Approximation and Online Algorithms in Scheduling and Coloring

Dissertation

zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften
(Dr. rer. nat.)
der Technische Fakultät
der Christian-Albrechts-Universität zu Kiel

Aleksei V. Fishkin

Kiel 2003

1. Gutachter: Prof. Dr. Klaus Jansen
2. Gutachter: Prof. Dr. Anand Srivastav
3. Gutachter: Prof. Dr. Evripidis Bampis
Tag der mündlichen Prüfung: 16.06.2003
Zum Druck genehmigt: 16.06.2003

PREFACE

This thesis includes part of my research from the past three years. I am indebted to my teachers, colleagues, and friends for their help and support. First of all this concerns my advisor Klaus Jansen. He has devoted many hours of his time for introducing me into the field of combinatorial optimization and approximation algorithms during my first year in Kiel, when I needed it the most. This was of great benefit to my work. Only with his help I was able to get my first results and work further as a researcher. I am grateful to him for his support, advising, and friendship. I am also sincerely appreciate Maxim Sviridenko for recommending me to Klaus. I wish to thank Maxim and Klaus for their trust and confidence in my research abilities. Without all of this my thesis would never have never been written.

I have been fortunate to meet many wonderful people who have contributed to my research. I would especially like to thank Lorant Porkolab and Evripidis Bampis. Lorant supervised and later edited my first works on approximation schemes for scheduling problems. His ability to listen, his understanding, his confidence, and support will always be an inspiration. Collaboration with Evripidis in several projects gave me an impetus to produce some results in online scheduling. I have learned a lot from his calm and diligent approach to research.

I have spent three productive years at Kiel University and thank all my colleagues in the group "Theory of Parallelism" at the Institute of Theoretical Computer Science and Applied Mathematics for a pleasant environment and a stimulating working atmosphere. Long evenings of discussions that I have spent with Jiří Fiala at the University guesthouse have turned into a joint work on the graph labeling problem. My trip to Prague, as a guest at his marriage, our jokes and our friendship stay with me forever. I am also happy to meet Monaldo Mastrolilli, Manuela Montangero, Guochuan Zhang, Jana Chlebíková and Hu Zhang. My colabaration with Monaldo and Guochuan resulted in several joint papers on scheduling, which are partially included here. I would like to thank Marian Margraf for working with me on different topics, Brigitte Preuss for her help with the paperwork, and them both for improving my German.

My special thanks to Martin Skutella for several helpful discussions and for his help with every small request I have made of him.

I made my research as a member of the graduate school 357 "Effiziente Algorith-

men und Mehrskalenmethoden” supported by the Deutsche Forschungsgemeinschaft. I am also grateful for the funding support provided by the EU project ARACNE “Research Training Network” (HPRN-CT-199-00112), the EU projects APPOL I and II “Thematic Network” (IST-1999-14084, IST-2001-32007), the DAAD project Procope “Scheduling of Malleable Tasks”, and the EU-Project CRESCCO “Critical Resource Sharing for Cooperation in Complex Systems” (IST-2001-33135).

Finally, I would like to express my gratefulness to my mother. Her unconditional love and support, though sometimes irrational, plays an important role in my life. I dedicate this thesis to her for all that she has done for me.

Aleksei V. Fishkin

Kiel, October 2002.

CONTENTS

INTRODUCTION

It frequently happens that in attempting to obtain a solution to an important problem we realize that this problem is difficult. This observation is especially true for many optimization problems [AGG$^+$99, BEY98, CCPS98, FW98, GJ79, Hoc96, Hro01, NW88, Pap94, PS82].

Solving an optimization problem we want to have an algorithm that will find an optimal solution for any instance of the problem. It is commonly held opinion that an optimization problem has not been solved efficiently until a polynomial time (deterministic) algorithm has been obtained for it. Unfortunately, most real world optimization problems seem to be too hard to be solved efficiently and, in fact, even many simply stated problems are believed to be intractable. The theory of NP-*completeness* provides a mathematical foundation for this belief [Coo71, GJ79].

We can informally summarize it as follows. A decision problem is one whose solution is either "yes" or "no". There are two classes of decision problems NP and P. It holds that $\mathrm{P} \subseteq \mathrm{NP}$. Furthermore, all problems in P can be solved efficiently, whereas all problems in $\mathrm{NP} - \mathrm{P}$ are intractable. An NP-*complete problem* $\Pi \in \mathrm{NP}$ has the property: $\Pi \in \mathrm{P}$ if and only if $\mathrm{P} = \mathrm{NP}$.

It is now widely accepted that NP-complete problems cannot be solved efficiently and $\mathrm{P} \neq \mathrm{NP}$. However, the problem "P versus NP" still remains one of the most challenging problems in theoretical computer science which is also included in the list of Millennium Prize Problems [CMI00].

The decision versions of many combinatorial optimization problems have been shown to be NP-*complete* [Kar72]. We might say that such combinatorial optimization problems are NP-*hard*, since they are, in a sense, at least as hard as the NP-complete problems.

If an optimization problem is NP-hard, then there exists no algorithm which would compute optimal solutions in polynomial time, unless $\mathrm{P} = \mathrm{NP}$. But, we can ask for less. We could relax the requirement the running time be polynomial or we need not require the solutions be optimal. Indeed, we can use *heuristic* algorithms like Local Search [AL97] and *enumeration* algorithms like Branch-and-Bound [HS78]. However, in the *worst-case analysis* such algorithms are either not polynomial or produce very *sub-optimal* solutions.

Here we are interested in the design and analysis of *approximation algorithms* that always compute *near-optimal* solutions in polynomial time [AGG+99, Hoc96, Hro01].

Approximation Algorithms. An optimization problem can be either cost minimization or profit maximization. Informally, an optimization problem Π of cost minimization consists of a set $\mathcal{I}$ of instances (inputs) and a cost function C. Associated with each instance $I \in \mathcal{I}$ is a set of feasible solutions (outputs) $F(I)$. For each instance I and a feasible solution $S \in F(I)$, the cost associated with I and S is $C(I,S) \in \mathbb{R}^+$. The kind of optimization problems we typically concerned with are of cost minimization problems; therefore, the discussion here is primarily in terms of cost problems. It is not difficult to develop the analogous concepts for profit maximization problems.

Let ALG be any algorithm for a cost minimization problem Π. Let $\mathrm{ALG}[I]$ denote a feasible solution produced by ALG given the instance I, and let

$$\mathrm{ALG}(I) \;=\; C(I, \mathrm{ALG}[I])$$

denote the cost incurred by ALG. An *optimal algorithm* OPT is such that for each instance I,

$$\mathrm{OPT}(I) \;=\; \min_{S \in F(I)} C(I,S).$$

An algorithm ALG is a ρ-approximation algorithm for a cost minimization problem Π if for all instances I,

$$\mathrm{ALG}(I) \;\leq\; \rho \cdot \mathrm{OPT}(I).$$

The running time of ALG is polynomial in the instance size $|I|$. The value of $\rho \geq 1$ is called the *approximation ratio* or *performance ratio* or *worst-case ratio* of ALG and in general can be a function of $|I|$.

A family of approximation algorithms, $\{A_\varepsilon\}_{\varepsilon>0}$, for a cost minimization problem Π is called a *polynomial time approximation scheme* or a PTAS, if algorithm A_ε is a $(1+\varepsilon)$-approximation algorithm and its running time is polynomial in the size of the instance for a fixed ε. If the running time of each A_ε is polynomial in the size of the instance and $1/\varepsilon$, then $\{A_\varepsilon\}_{\varepsilon>0}$ is called a *fully polynomial time approximation scheme* or a FPTAS.

For any given NP-hard optimization problem, we wish to determine whether it possesses a ρ-approximation algorithm, or a PTAS, or even a FPTAS. Thus, on one hand, *positive (approximability) results* in the area of approximation concern

the design and analysis of good polynomial time approximation algorithms and schemes, and on the other hand, the *negative (inapproximability) results* disprove the existence of such algorithms.

So far we assumed that the input instance of an optimization problem is completely known prior to the beginning of the computations. On the basis of this complete input, an algorithm produces an optimal or near-optimal solution as the output. In other words, we assumed problems and algorithms be *offline*.

Optimization problems in which the input is received in an online manner and in which the output must be produced online are called *online problems*. The complication is that each online output influences on the cost of the overall solution. This suggest a "natural" partition of optimization problems into online and offline problems. Another field of our interest is the design and analysis of *online algorithms* which solve online problems [BEY98, FW98].

Online Algorithms. Here we only consider a special class of online optimization problems. For an online optimization problem Π each instance $I \in \mathcal{I}$ appears as a finite sequence $I = i_1, i_2, \ldots, i_n$ and a corresponding feasible solution $S \in F(I)$ is a finite sequence $S = s_1, s_2, \ldots, s_n$. An online algorithm ALG for Π must produce a feasible solution in stages such that at the jth stage ($j = 1, 2, \ldots, n$) the algorithm is presented with the jth component of the instance and must produce the jth component of a feasible solution before the rest of the instance is made known.

Let OPT be an optimal *offline* algorithm for a cost minimization problem Π. An online algorithm ALG is *c-competitive* (or "attains a competitive ratio of c") if for each instance I,

$$\mathrm{ALG}(I) \leq c \cdot OPT(I).$$

The smallest c such that ALG is c-competitive is called ALG's *competitive ratio*. Thus, a c-competitive algorithm is quaranteed to incur a cost no larger than c times the smallest possible cost for each input sequence. We assume that $c \geq 1$ and in general it can be a function of some problem parameters.

For any given online problem, we wish to find an c-competitive algorithm with the smallest ratio c. Accordingly, a *lower bound* on the competitive ratio implies that no online algorithm can have a competitive ratio less than this bound.

Notice that the competitiveness of an online algorithm is evaluated with respect to an offline optimal algorithm. Strictly speaking, it does not indicate the loss associated with the computational resources availability, but with not having complete

information of the input instance. Hence, c-competitive and ρ-approximation algorithms are not comparable, even if $c = \rho$. However, here we consider only "efficient" online algorithms and, in particular, algorithms that do terminate within a polynomial (in the relevant parameters) time.

Outline of the thesis

In the last three decades, approximation and online algorithms have become a major area of theoretical computer science and discrete mathematics, rich in its powerful techniques and methods [FW98, Hoc96]. Scheduling and coloring problems are among the most popular ones for which approximation and online algorithms have been analyzed. On one hand, motivated by the well-known difficulty to obtain good lower bounds for the problems, it is particularly hard to prove results on the online and offline performance of algorithms. On the other hand, the theoretically oriented studies of approximation and online algorithms for scheduling and coloring have also impact on the development of better algorithms for real world applications.

In this thesis we contribute in two ways. First, we design polynomial approximation algorithms, in particular PTASs, for two wide classes of NP-hard scheduling problems which concern minimizing the *average weighted completion time*. Second, we develop a number of online and approximation algorithms for several *multiprocessor task* scheduling problems and several generalizations of the coloring problem, namely *distance constrained labeling* problems, which are primarily motivated by applications. We answer theoretical questions, present new and improve on previous results, develop novel techniques and methods, and give pointers to some experimental results.

The main part of this thesis is divided into four chapters. One can find some relationship between them. However, each chapter is intended to be mostly self-contained, and we hope that the reader interested in a particular topic would have no problem in reading only the corresponding part.

CHAPTER 1: In this chapter we present new approximation results for scheduling to minimize the average weighted completion time with release dates. We are given n jobs, where job j ($j = 1,2,\ldots,n$) has a processing time p_j, a positive weight w_j and a release date r_j. We consider several variants but the objective in all of them is to minimize the average weighted completion time $\sum w_j C_j$, where C_j is the completion time of job j in the schedule. If there are no release dates and the goal is to schedule jobs on a single machine without preemptions, the problem

can be solved optimally in polynomial time by scheduling jobs in non-increasing order of w_j/p_j, what is known as Smith's rule [Smi56]. Introducing release dates makes the problem strongly NP-hard [LLKS93, CPW98].

Recently there has been significant progress in giving approximation algorithms with good performance quarantees for a variety of strongly NP-hard scheduling problems with the average weighted completion time objective and release dates, e.g. scheduling on a single machine, on parallel (related) machines, on a constant number of unrelated machines, ect. [AM01]. There are just few approximation approaches. One one hand, one can either use a linear programming (LP) relaxation or preemptive relaxation to find a schedule [Sch96, Shm98, Sku98], or use an algorithm which first defines the most profitable jobs and then packs them into contiguous intervals of geometrically increasing sizes [HSSW97, CPS$^+$96]. Indeed, a series of ideas along these lines have led to many nice and practical approximation algorithms. On the other hand, there seem to be fundamental barriers to turning these algorithms into *polynomial time approximation schemes* (PTASs): algorithms that, for any fixed accuracy $\varepsilon > 0$, find a schedule within a $(1+\varepsilon)$ factor of the optimum in polynomial time. In order to build a PTAS, a new approach was proposed in [ABC$^+$99]. This involves the input transformation technique, the idea of intervals, the idea of huge-tiny jobs, the weight-shifting technique, and dynamic programming [AM01, CK01].

At the same time, it was always hard to say whether one can use already known approximation algorithms for the makespan version of the problem. Chandra Chekuri's Ph.D. thesis [Che98] concludes with the following question:

" Is there a polynomial time algorithm that uses as a subroutine a procedure for minimizing makespan and outputs an approximate schedule for minimizing weighted completion time?"

Indeed, several algorithms of this kind are known [CPS$^+$96, QS00]. However, these results give only a partial answer to the question, and giving a fully positive answer - for instance, a PTAS that uses as a subroutine a PTAS for minimizing makespan - remained an interesting challenge.

Here we also combine several ideas – the input transformation technique, the idea of intervals, the idea of huge-tiny jobs, the weight-shifting technique, an LP relaxation (formulation), rounding, a PTAS for minimizing makespan, and dynamic programming – into one approximation method. In particular, we demonstrate the power of our method on the job shop scheduling and multiprocessor (dedicated and parallel) tasks scheduling problems. This improves and generalizes a number of results presented in [CPS$^+$96, CLL98, ABC$^+$99, ABKM00, ABF$^+$00, FJP01b].

Interestingly, here we just relay on the properties of the known PTASs [JSOS99, CM99, JP99b] modifying them in some parts. Our method uses a PTAS for the makespan version of the problem not as a subroutine, but as a main part of a special proof technique. However, the techniques developed can be used for making the PTASs faster and simpler [FJM01].

In addition, our method can also lead to the design of PTASs for many variant of the multistage scheduling problem with the average weighted completion time objective and release dates, e.g. open shop, flow shop, dag shop, and their preemptive or multiprocessor versions. Thus, we not only give the answer to the above question, but also make a major step to understanding the approximability of shop scheduling problems with the sum of completion time objective, a question mentioned by Schuurman & Woeginger [SW99].

In Section 1.3 we first consider the problem of scheduling jobs on a single machine: Given n jobs, where each job j ($j = 1,2,\ldots,n$) has a processing time p_j, a release date r_j, and a weight w_j, schedule the jobs on a single machine so that the average weighted completion time $\sum w_j C_j$ is minimized, where C_j denotes the completion time of job j.

As we mentioned before, the problem is strongly NP-hard [LLKS93, CPW98] and there are a number of polynomial (efficient) approximation algorithms [Sch96, Sku98, Che98]. Here we start with reviewing and refining a PTAS proposed in [ABC$^+$99]. We outline the main steps of our method. We repeat and modify some lemmas and proofs in [ABC$^+$99] adding some new features and introducing the basic techniques. This provides a gentle introduction to the subject and simplifies the presentation of subsequent results.

Next, in Section 1.4, we consider the following job shop scheduling problem. We are given a set of m machines $M = \{1,2,\ldots,m\}$ and a set of n jobs $J = \{1,2,\ldots,n\}$. Each job j ($j = 1,2,\ldots,n$) has a weight w_j, a release date r_j, and consists of $\mu \geq 2$ operations $o_{1j},\ldots,o_{\mu j}$ that have to be processed in the given order. Each operation o_{ij} ($i = 1,2,\ldots,\mu$) requires a machine $\tau_{ij} \in M$ and has a processing time p_{ij}. Each job can be processed only by one machine at a time and each machine can process only one job at a time. The goal is to find a non-preemptive feasible schedule which minimizes $\sum w_j C_j$.

The problem is hard even if it is assumed that there are no weights (all $w_j = 1$) and no release dates (all $r_j = 0$). The problem with two machines and at most two operations per job is already strongly NP-hard [GJS76], and if the number of machines m and the number of operations per job μ are arbitrary, the general job shop scheduling problem is APX-hard [HSW98]. However, there is a $(5.78+\varepsilon)$-approximation algorithm for the problem in the case when m and μ are fixed [CPS$^+$96], and a $\tilde{O}(\log^2 m\mu)$- approximation algorithm for the problem with

arbitrary (not fixed) m and μ [QS00].

Here we present a PTAS for the problem which computes for any fixed values m, μ and $\varepsilon > 0$ accuracy, a $(1+\varepsilon)$-approximate schedule in $O(n \log n)$ time. Notice that our PTAS for the problem with fixed m and μ, cannot be extended to a FPTAS [GJ79] (a PTAS which is also polynomial in $1/\varepsilon$), and it does not generalize to the case when both m and μ are not fixed. It remains an open question whether one can obtain a PTAS when only one of the parameters, either m or μ, is fixed.

Finally, in Section 1.5, we address multiprocessor scheduling problems, where a set of n tasks has to be executed by a set of m processors such that each processor can work on at most one task at a time and a task can (or may need to be) processed simultaneously by several processors. The objective is to minimize the average weighted completion time $\sum w_j C_j$.

In the *dedicated* model, each task requires a simultaneous use of a prespecified set of processors. In the *parallel* model, each task requires a prespecified number of processors. In the *general* model, each task can have a number of *alternative modes*, where each processing mode is specified by a subset of processors and the execution time of the task on that particular processor set.

Both dedicated and parallel versions are strongly NP-hard even if it is assumed that there are no weights, no release dates and two processors $m = 2$ [XL99, CLL98]. In [FJP01b] we have proven that scheduling dedicated tasks with unit processing times on an arbitrary number (not fixed) of processors m cannot be approximated within a factor of $m^{\frac{1}{2}-\varepsilon}$, neither for some $\varepsilon > 0$, unless P = NP; nor for any $\varepsilon > 0$, unless NP = ZPP.

There are only few known approximation results for scheduling multiprocessor tasks. Furthermore, they only address the case when it is assumed that there are no weights and no release dates. It has been known, that there exist a 2-approximation algorithm for scheduling dedicated tasks on two processors [CLL98], a 32- approximation algorithm for scheduling parallel tasks on arbitrary number of processors [TLWY94], and - as we have shown recently - a PTAS for scheduling dedicated tasks on a fixed number of processors [ABF+00].

By combining various ideas we provide here generalizations of several results and obtain PTASs for both parallel and dedicated models (with release dates) that compute for any fixed value of m and accuracy $\varepsilon > 0$, $(1+\varepsilon)$-approximate schedules in $O(n \log n)$ time. We also prove that there is a PTAS for the general model, but in the case when there are no release dates. However, we conjecture the existence of a PTAS in the case when general multiprocessor tasks have release dates.

CHAPTER 2: In this chapter we consider the following *distance constrained labeling* problem [Lee98]. We are given a graph $G = (V,E)$ and a non-increasing sequence of *distance constraints* $p_1, p_2, \ldots, p_k$. The goal is to find an $L_{(p_1,\ldots,p_k)}$-*labeling* $c : V(G) \to \{1,2,\ldots,L\}$ of G such that $|c(u) - c(v)| \geq p_i$ whenever the vertices u and v are at graph distance i, for $i = 1,2,\ldots,k$, and the number of labels L is minimized.

The problem is a generalization of the well-known coloring problem, where given a graph the goal is to find an assignment of colors (numbers $1,2,3,\ldots$) to the vertices of the graph such that any two adjacent vertices have distinct colors and the number of colors is minimized [JT95, SU97]. The most intensively studied case of distance constrained labeling is $k = 2$ and $(p_1, p_2) = (2,1)$. The hardness and approximability of an $L_{(2,1)}$-labeling was explored for different graph classes, e.g. *cycles*, *paths*, *trees* and *co-graphs* [BKTvL00, CK96, GM96, GY92, vdHLS98, FKK99, MS]. For general graphs, it is expected that for every fixed k-tuple of *distance constraints* $(p_1,\ldots,p_k)$ whether exists a bound L_0 such that it is NP-hard to decide where there exists an $L_{(p_1,\ldots,p_k)}$-labeling with the number of colors $L \leq L_0$. So far, this conjecture has been only proven for $k = 2$ and $p_1 \geq 2\,p_2$ [Fia00].

The intersection graph G of a set D of disks in the plane is called a *disk graph*, and D is called the *disk representation* of G. If all disks of D have unit diameter, G is called a *unit disk graph*, and if the disk *diameter ratio* is bounded by some constant σ, G is called a σ-*disk graph*.

The recognition problem of a (unit, σ-) disk graph is NP-hard [BK96, BK98, HK01]. There are a number of offline and online approximation algorithms for coloring of (unit) disk graphs [BK98, Pee91, Mal97, Pee91].

Here we consider the problem of distance constrained labeling of σ-disk graphs. We present a number offline and online algorithms for the case of general distance constraints and for the case when $k = 2$ and $(p_1, p_2) = (2,1)$. We derive upper and lower bounds on the approximation and competitive ratio of the algorithms presented.

First, for each fixed k-tuple of distance constraints $(p_1,\ldots,p_k)$ we give an online $L_{(p_1,\ldots,p_k)}$-labeling algorithm which requires the disk representation of a σ-disk graph, i.e. it works on sets of disks with the disk diameter ratio at σ as the input. The algorithm is based on the so-called *hexagonal tiling*, *circular labeling*, and *first-fit* techniques. We derive an upper bound on its competitive ratio and show that for each fixed k-tuple $(p_1,\ldots,p_k)$ of distance constraints and each fixed diameter ratio σ, the algorithm is constant competitive. As an example, we demonstrate the algorithm for the case $k = 2$ and $(p_1, p_2) = (2,1)$.

Next, we derive lower bounds for online coloring and labeling. We consider the case when the disk representation of a disk graph is known. We start with simple

lower bounds for unit disk graphs. Then, we switch to disk graphs and prove that in the case when either the disk representation of disk graphs is not given or the diameter ratio is not bounded, no online labeling algorithm with a constant competitive ratio exists. In addition, we find a lower bound on any general online labeling algorithm for σ-disk graphs. By using this result we show that our online labeling algorithm is asymptotically optimal for the class of unit disk graphs.

Finally, we deal with the offline setting. We explore the case when $k = 2$ and $(p_1, p_2) = (2,1)$. We present two approximation algorithms for unit disk graphs. The first algorithm is based on the so-called *cutting* technique, which uses the disk representation of unit disk graphs. The second algorithm is *robust*, what is, it does not require the disk representation, and it either outputs a feasible labeling or shows that the input is not a unit disk graph. For the case of arbitrary distance constraints, we present a general offline labeling algorithm for σ-disk graphs which does not require the disk representation. For each fixed σ and k the approximation ratio of the algorithm is constant.

CHAPTER 3: In this chapter we address the following *dedicated* variant of the multiprocessor tasks scheduling problem. We are given a set of n tasks $T = \{1,\ldots,n\}$ and a set of m processors $M = \{1,2,\ldots,m\}$. Each task $j \in T$ has a processing time p_j, a release date r_j and a prespecified set of processors $\text{fix}_j \subseteq M$. Preemptions are not allowed. Each processor can work on at most one task at a time, each task $j \in T$ must be processed simultaneously by all processors of fix_j. The goal is to find a non-preemptive feasible schedule which minimizes the *makespan* $C_{\max} = \max_j C_j$.

The three-processor problem is already strongly NP-hard, and there exists no polynomial approximation algorithm with performance ratio smaller than $\frac{4}{3}$, unless P=NP [HdVV94]. Furthermore, in [FJP01b] we proved that the problem cannot be approximated within a factor of $m^{\frac{1}{2}-\varepsilon}$, neither for some $\varepsilon > 0$, unless P=NP; nor for any $\varepsilon > 0$, unless NP=ZPP. However, there is a polynomial time approximation scheme (PTAS) for the variant of the problem with fixed number of processors m [ABKM97]. The best known low time complexity algorithms are a $\frac{7}{6}$-approximation algorithm and a $\frac{9}{8}$-approximation algorithm for the three-processor problem presented in [Goe95, CH01].

In the online context, there are several results known for the *parallel* variant of the multiprocessor task scheduling problem [FST94, FKST93, BM98]. Up to our best knowledge, no online algorithms with guaranteed competitive ratio are known for the dedicated variant of the multiprocessor model considered here. Although some algorithms have been recently proposed in the literature, their performance is only simulated and is not analyzed analytically [CDI01b, CDI01a].

Here we present several results, important from both theoretical and practical points of view. First, we deal with the problem where tasks have unit processing times. We propose an online algorithm for the variant in which tasks arrive over-list and an online algorithm for the variant in which tasks arrive over-time. The competitive ratio of these algorithms is bounded by $2\sqrt{m}$ and $(2\sqrt{m}+1)$ respectively. Next, we switch to the general problem with arbitrary processing times. We show that any online algorithm which schedules tasks arriving over-list and leaves no unnecessary idle time cannot be better than m-competitive. In some sense, this leaves no hope for contracting any good online algorithm based on the first-fit technique. To simplify the problem we consider the case when all tasks are released at the beginning of scheduling. By using our *split-round* technique, we first give an offline $2k$-approximation algorithm for the problem where the maximum task size $\Delta_{\max}$ is bounded by some constant k, and modify it to an offline $3\sqrt{m}$-approximation algorithm for the general case. Then, by using the so-called *active-passive-bins* scheduling technique [SWW95], we outline an online algorithm for the variant in which tasks arrive over-time and the existence of a task is unknown before its release date. The competitive ratio of the algorithm is bounded by $6\sqrt{m}$.

CHAPTER 4: In this last chapter we address the following multiprocessor scheduling problem. We are given a set of n tasks $T = \{1,2,\ldots,n\}$ and a set of m processors $M = \{1,2,\ldots,m\}$. Each task j ($j = 1,2,\ldots,n$) has a unit processing time $p_j = 1$ and a due date d_j. Each processor can work on at most one task at a time and a task can (or may need to be) processed simultaneously by several processors. In the *dedicated* model, for each task $j \in T$ there is given a prespecified set $\mathrm{fix}_j \subseteq M$ which indicates the task must be processed by all the the processors of fix_j. In the *parallel* model, the multiprocessor architecture is disregarded and for each task $j \in T$ there is given a prespecified number $size_j \in M$ which indicates that the task can be processed by any subset of processors of the cardinality equal to $size_j$. Here we assume that all tasks are available at time zero and the goal is to find a feasible schedule which maximizes the *throughput* $\sum \bar{U}_j$, where $\bar{U}_j = 0$ if task j completes after d_j, and $\bar{U}_j = 1$ otherwise.

There are a lot of results known for the classical (non-multiprocessor) job scheduling problems, where the objective is either to minimize the (weighted) number of late (tardy) jobs or to maximize the (weighted) number of on time (early) jobs, see e.g. [Bru98, DP95, HPW00, KIM78, Law76, Law82, LM69, LKB77, Mon82, Moo68, RW98]. In the multiprocessor setting, the previous research has mainly focused on the objectives of minimizing *the makespan* and *the sum of completion times*. As a rule, scheduling multiprocessor tasks with unit processing times is a strongly *NP*-hard problem [Llo81, HdVV94]. However, a number of differ-

ent approximation algorithms have been recently proposed in [ABKM97, BM98, CH01, CLL98, FST94, FKST93, Goe95, Llo81, TLWY94]. Up to our knowledge, no results are known for the multiprocessor tasks scheduling problem which concern either minimizing the number of late (tardy) tasks or maximizing the number of on time (early) tasks.

Here, focusing on the throughput objective, we present the first results in this direction. We derive the complexity results and present several approximation algorithms, for both parallel and dedicated variants of the problem.

We start with the parallel variant of the problem. We prove that the problem is strongly NP-hard and propose two simple greedy algorithms. Then, we present an improved algorithm with the worst-case ratio at most $3/2-1/(2m-2)$ (here $m \geq 2$). Finally, we consider the dedicated variant. For the case when all tasks have a common due date, we adopt the complexity result for MAXIMUM CLIQUE [Has99]. We prove that for any given $\varepsilon > 0$ the problem cannot be approximated within $m^{1/2-\varepsilon}$ unless NP = ZPP, where m is number of processors. However, for this special case, we are able to show that the worst-case ratio of a greedy algorithm does not exceed $\sqrt{m}+1$. To grip on the case of individual due dates, we generalize this algorithm and demonstrate that bound $\sqrt{m}+1$ remains valid.

Interestingly, there are a number of different relations to some well-known combinatorial problems. Just beyond the mentioned relation to MAXIMUM CLIQUE, BIN PACKING and MULTIPLE KNAPSACK correspond to the parallel variant of our problem.

Last notes. Throughout the thesis we use the standard three-field $\alpha|\beta|\gamma$ scheduling notation introduced in [GLLK79, LLKS93, Dro96]. We give a short version of this scheme in Appendix A on page 193. In Chapter 1 we use a rounding procedure included into Appendix B on page 197. For the sake of convenience, we also give all main definitions used from graph theory in Appendix C on page 201, and from complexity theory in Appendix D on page 205.

We assume that the reader is familiar with the basic concepts of combinatorial optimization, complexity theory, approximation and online algorithms which can, for instance, be found in the following books [AGG+99, BEY98, CCPS98, FW98, GJ79, Hoc96, Hro01, NW88, PS82, Pap94]. There are a number of books on machine scheduling [CCLL95, Pin95], on graph theory [Bol98, BR99, Wes01], and on linear programming [BT97, Sai95]. We would also recommend the following surveys [AM01, CPW98, Dro96, Kos99, SS00].

Parts of this thesis have been published or will be published in [ABF+00, FJP01b, FJP01a, FFF01, FJM01, BCF+02, FJM02, FZ02].

CHAPTER 1

ON MINIMIZING AVERAGE WEIGHTED COMPLETION TIME

1.1 INTRODUCTION

In this chapter we present new approximation results for several NP-hard scheduling problems in which jobs have release dates and the objective is to minimize the average weighted completion time. We are primarily interested in the design of PTASs: algorithms that, for any fixed accuracy $\varepsilon > 0$, find a solution within a $(1+\varepsilon)$ factor of the optimum in polynomial time. Here we combine a number of ideas into one approximation method. Our approach is built on the input transformation technique [ABC$^+$99] which makes the use of intervals with geometrically increasing sizes [HSSW97, CPS$^+$96], LP formulation [Sch96, Shm98, Sku98, Che98], LP rounding [JSOS99], and approximation algorithms (PTASs) for makespan minimization [ABKM97, CM99, JP99b, JSOS99]. We demonstrate the power of our method on the job shop scheduling problem and the multiprocessor (dedicated and parallel) tasks scheduling problem. In particular, this improves and generalizes a number of results presented in [CPS$^+$96, CLL98, ABC$^+$99, ABKM00, ABF$^+$00, FJP01b]. As a consequence, we make a major step to the answer to the questions mentioned in [Che98, SW99].

Scheduling on a Single Machine. We consider the following problem of scheduling jobs on a single machine: Given n jobs, where job j ($j = 1,2,\ldots,n$) has a processing time p_j, a positive weight w_j and a release date r_j, schedule the jobs on a single machine so as to minimize $\sum w_j C_j$, where C_j is the completion time of job j. The problem is denoted by $1|r_j|\sum w_j C_j$.

If there are no release dates and the goal is to schedule jobs on a single machine without preemptions, $1||\sum w_j C_j$, the problem can be solved optimally in polynomial time by scheduling jobs in non-increasing order of w_j/p_j, what is known as Smith's rule [Smi56]. Introducing release dates makes the problem strongly NP-hard [LLKS93, CPW98].

There are a number of nice and practical algorithms for $1|r_j|\sum w_j C_j$ which are

based on the LP and preemptive relaxations [Sch96, Shm98, Sku98, Che98]. However, there seem to be fundamental barriers to turning any of these algorithms into a PTAS: an algorithm which given $\varepsilon > 0$ accuracy will output a $(1+\varepsilon)$-schedule in polynomial time. From one side, there is always a gap between the objective values of an LP relaxation and an optimal schedule. Hence, any algorithm that compares itself to the LP optimum inherits this gap, and so it cannot be a PTAS. From another side, as it was shown in [TU99], any algorithm for $1|r_j|\sum C_j$ that starts with the preemptive relaxation created by shortest remaining processing time (SRPT) algorithm [LLKS93] cannot find an approximation ratio better than $e/(e-1)$. This matches the best known upper bound [CMNS97].

In order to build a PTAS a new approach was proposed in [ABC+99]. This employs the well-known input transformation technique [SW02], the idea of intervals with geometrically increasing sizes [HSSW97, CPS+96], and the weight-shifting and merging techniques [AM01]. The approach to approximation is to perform several "transformations" that simplify the input problem without dramatically increasing the objective value, such that the final result is amenable to a fast dynamic programming solution.

Informally, in order to find a $(1+\varepsilon)$-approximate schedule one proceeds as follows. First, all processing times and release dates are rounded to integer powers of $(1+\varepsilon)$. This transformation increases the objective function by at most a factor of $1+\varepsilon$. Furthermore, this breaks the time line into contiguous *geometrically increasing intervals* $I_x = ((1+\varepsilon)^x, (1+\varepsilon)^{x+1}]$, $x \in \mathbb{Z}$, where release dates only happen at the beginning of intervals. Then, jobs are divided into *tiny* and *huge* ones. In particular, a job is *huge* with respect to an interval if its length is at least ε^2 times the size of the interval, and *tiny* otherwise. For one interval I_x, huge jobs of the same size (a power of $1+\varepsilon$) are prioritized by decreasing weights w_j, and all tiny jobs are prioritized by decreasing ratio w_j/p_j (Smith's rule). Next, the weight-shifting technique is applied. If many jobs released at interval I_x, it is known that some of them with low priority will have to wait to be processed. Shifting refers to the process of moving the excess jobs to the next interval I_{x+1}. Finally, the merging technique is used. The set of tiny jobs released at interval I_x is partitioned into a constant number of subsets with roughly equal total processing time, and the jobs of each subset are grouped into one single tiny job. From one side, both techniques increase the objective value by at most a factor of $1+O(\varepsilon)$. From another side, now it is quite simple to enumerate all reasonable $(1+\varepsilon)$-schedules by using dynamic programming.

Here we start with reviewing and refining a PTAS for $1|r_j|\sum w_jC_j$ presented in [ABC+99]. We repeat and modify some lemmas and proofs in [ABC+99], introducing some basic techniques and adding some new features. By such means we outline main parts of our method: (1) Structuring (2) Compacting, and (3)

Dynamic Programming. Similarly to the above approach [ABC+99], in (1) and (2) we show how one can simplify the input problem, and in (3) we show how one can solve the final problem by using dynamic programming. This provides a gentle introduction to the subject and simplifies the presentation of subsequent results.

Job Shop Scheduling. We consider the following job shop scheduling problem. We are given a set of n jobs $J = \{1,2,\ldots,n\}$ and a set of m machines $M = \{1,2,\ldots,m\}$. Each job j ($j = 1,2,\ldots,n$) has a positive weight w_j, a release date r_j, and consists of a sequence of $\mu \geq 2$ operations $o_{1j}, o_{2j}, \ldots, o_{\mu j}$ that must run in the given order. Each operation o_{ij} ($\mu = 1,2,\ldots,\mu$) has a processing time p_{ij} and requires a machine $\tau_{ij} \in M$. Each job can be processed only by one machine at a time and each machine can process only one job at a time. Here we assume that m and μ are fixed, and the goal is to find a non-preemptive feasible schedule which minimizes $\sum w_j C_j$, where C_j denotes the completion time of job j. The problem is denoted by $Jm|op \leq \mu, r_j|\sum w_j C_j$.

The above problem is an important generalization of single machine scheduling problem $1|r_j|\sum w_j C_j$. However, it seems to be harder for approximating even if in the case when there are unit weights and no release dates. It was shown that two machine problem $J2|op \leq 2|\sum C_j$ with at most two operations per job is already strongly NP-hard [GJS76], and general problem $J||\sum C_j$ with arbitrary m and μ is APX-hard [HSW98].

For a long period only a simple $O(m)$-approximation algorithm for $J||\sum C_j$ has been known [GS78]. Until recently, a $(5.78+\varepsilon)$ - approximation algorithm for problem $Jm|op \leq \mu, r_j|\sum w_j C_j$ with fixed m and μ, and an $\tilde{O}(\log^2(m\mu))$ - approximation algorithm for problem $J|r_j|\sum w_j C_j$ with arbitrary m and μ have been proposed in [CPS+96] and [QS00], respectively. Both algorithms follow the same approach. One first uses a subroutine for "assigning" jobs to intervals with geometrically increasing sizes, and then a subroutine for "packing" jobs in single intervals. However, subroutines are different. In the first case, there are a *dual* algorithm and a $(2+\varepsilon)$-approximation algorithm [SSW94], whereas in the second case, there are an LP relaxation and an $\tilde{O}(\log^2(m\mu))$ - approximation algorithm [GPSS97].

Here we improve on the result by Chakrabarti et al. and prove that there is a PTAS for problem $Jm|op \leq \mu, r_j|\sum w_j C_j$ which computes for any fixed m, μ and $\varepsilon > 0$ accuracy, a $(1+\varepsilon)$-approximate schedule in $O(n \log n)$ time. Notice that our PTAS for the problem with fixed m and μ, cannot be extended to a fully PTAS [GJ79] (a PTAS which is also polynomial in $1/\varepsilon$), and it does not generalize to the case when both m and μ are not fixed. It remains an open question whether one can

obtain a PTAS when only one of the parameters, either m or μ, is fixed.

In order to build a PTAS we follow the main parts of our method: (1) Structuring (2) Compacting, and (3) Dynamic Programming. We make almost no chances in parts (2) and (3) presented for the single machine problem $1|r_j|\sum w_jC_j$, but we generalize part (1) in a non-trivial way.

In the case of single machine it is easy to structure the input problem. In contrast, a job in the job shop problem is a chain of operations, that creates a lot of difficulties. Here we have to define *main* and *negligible* operations of a job. All negligible operations are very small ones, and we can round their processing times to zero. All main operations are very big, and we can round them such that the input instance becomes simpler. Only after this, by a non-trivial proof we are able to show that any job can be completed within a constant number of intervals.

Another important difference is that for problem $1|r_j|\sum w_jC_j$ tiny jobs can be prioritized by Smith's rule. When the job shop problem is addressed, several aspects become complex and some important generalizations are needed. We first partition all jobs into a constant number of subsets sharing similar characteristics, called *profile*. This introduces a special structure on a schedule in which each tiny job has a *pattern*. Then, by using intervals and patterns we are able to modify Smith's rule, but for tiny jobs of the same profile. In order to prove this, similarly to ideas in [CPS$^+$96, QS00], we first use a subroutine for "assigning" tiny jobs to intervals and patterns, and then as a subroutine for "packing" jobs in single intervals. From one side, for assigning we can use an LP formulation and a special rounding procedure. From another side, packing jobs in each single interval corresponds to the makespan version of the job shop scheduling problem which is known to be NP-hard [GJ79]. To copy with that, we follow ideas of the PTAS which first presented in [JSOS99] and later modified in [FJM01]. This the most difficult step of part (1) of our method.

Interestingly, we can generalize all above mentioned ideas and techniques. By following our method we can prove that there are PTASs for many variants of the shop scheduling problem, e.g. open shop, flow shop, dag shop, and their preemptive versions.

Multiprocessor Task Scheduling. We address non-preemptive multiprocessor scheduling problems, where a set of n tasks $T = \{1,2,\ldots,n\}$ has to be executed by a set of m processors $M = \{1,2,\ldots,m\}$ such that each processor can work on at most one task at a time and a task can (or may need to be) processed simultaneously by several processors. Each task $j \in T$ has a processing time p_j, a positive weight w_j and a release date r_j. Here we assume that m is fixed and the goal is to find a non-preemptive schedule which minimizes $\sum w_jC_j$.

In the *dedicated* model, denoted by $Pm|\,\mathrm{fix}_j, r_j|\sum w_jC_j$, each task $j \in T$ requires the simultaneous use of a prespecified set of processors $\mathrm{fix}_j \subseteq M$. In the *parallel* model, denoted by $Pm|\,\mathrm{size}_j, r_j|\sum w_jC_j$, the multiprocessor architecture is disregarded and for each task $j \in T$ there is given a prespecified number $\mathrm{size}_j \in M$ which indicates that j can be processed by any subset of processors of cardinality size_j. In the *general* model, $Pm|\,\mathrm{set}_j, r_j|\sum w_jC_j$, each task can have a number of *alternative modes* and for each task $j \in T$ there is an associated function $p_{\cdot j} : 2^M \longrightarrow \mathbb{Z}^+ \cup \{+\infty\}$ which gives the execution time $p_{\tau j}$ of task j in terms of a set of processors $\tau \subseteq M$ that are *allotted* to j.

In the multiprocessor setting, problem $P|\,\mathrm{fix}_j|\sum C_j$ was first studied in [HdVV94] and shown to be strongly NP-hard even when all tasks have unit processing times. If the number of processors is fixed, then problem $Pm|\,\mathrm{fix}_j, p_j = 1|\sum C_j$ with unit processing times becomes polynomial-time solvable [BK96] even if the tasks have release dates. The negative result was strengthened in [CLL98], where the authors proved that already problem $P2|\,\mathrm{fix}_j|\sum C_j$ is strongly NP-hard. The problem of scheduling parallel multiprocessor tasks on two processors $P2|\,\mathrm{size}_j|\sum w_jC_j$ was also shown to be strongly NP-hard [XL99]. Contrasting this with the fact that for identical parallel machines, in the non-multiprocessor model, only general problem $P||\sum C_j$ is strongly NP-hard, while $Pm||\sum C_j$ is just weakly NP-hard, indicates that computing optimal or approximate schedules for multiprocessor tasks is likely to be much harder than the corresponding (classical) non-multiprocessor variants.

There are only few known approximation results for multiprocessor scheduling with the average completion time objective. Furthermore, they only address unweighted versions where it is also assumed that all tasks are released at the beginning. It has been known, that there exist a 2-approximation algorithm for $P2|\,\mathrm{fix}_j|\sum C_j$ [CLL98], a 32-approximation algorithm for $P|\,\mathrm{size}_j|\sum C_j$ [TLWY94], and - as we have recently shown - a PTAS for $Pm|\,\mathrm{fix}_j|\sum C_j$ [ABF$^+$00].

Here we provide generalizations of several results and prove the following: There are PTASs for both $Pm\ |\mathrm{fix}_j, r_j|\ \sum w_jC_j$ and $Pm|\,\mathrm{size}_j, r_j|\ \sum w_jC_j$ that compute for any fixed m and $\varepsilon > 0$ accuracy, $(1+\varepsilon)$-approximate solutions in $O(n\log n)$ time. If the number of processors m is not fixed, then the problem becomes harder from the point of view of computing approximate solutions for any given relative accuracy. It turns out that the classical *graph coloring* problem can be reduced to the problem of scheduling dedicated tasks of unit length. Based on this reduction it was proved in [HdVV94] that for $P|\,\mathrm{fix}_j, p_j = 1|C_{\max}$ there exists no polynomial approximation algorithm with performance ratio smaller than $4/3$, unless P=NP. (This can be strengthened by using inapproximability results from [LY94] and [FK98]). Following the same line of ideas, one can construct a reduction from the *minimum color sum* problem (introduced in [Kub89]), where the sum

of the assigned colors is minimized (instead of the largest one), hence differing from the classical coloring problem only in the objective. By using this reduction along with the hardness results [BNBH$^+$98, BNHK$^+$99] for approximating the minimum color sum problem, one can obtain the following negative (inapproximability) results:

Theorem 1.1.1. *Problem* $P|\,\mathrm{fix}_j, p_j = 1|\sum C_j$ *cannot be approximated within a factor of* $m^{\frac{1}{2}-\varepsilon}$*, neither for some* $\varepsilon > 0$*, unless* P= NP*; nor for any* $\varepsilon > 0$*, unless* NP=ZPP.

The problem of scheduling general multiprocessor tasks $Pm|\,\mathrm{set}_j|\,\sum w_jC_j$ (without release dates) can be also viewed as a generalization of two well (but mainly independently) studied scheduling problems: scheduling tasks on unrelated parallel machines $Rm||\sum w_jC_j$ and multiprocessor task scheduling with dedicated processors $Pm|\,\mathrm{fix}_j|\sum w_jC_j$. In the case of unrelated machines, for each task (job) there are m processing modes, each with a single processor (machine). In the case of dedicated processor sets, each task has only a single processing mode but including (typically) several processors. Since both of the above special cases are strongly NP-hard [BCJS74, CLL98] for general weights and $m \geq 2$, even if there are only a constant number of processors, it is natural to study how closely the optimum can be approximated by efficient algorithms. Focusing on the case where m is fixed, we integrate many of the above mentioned recent results that have shown the existence of PTASs for the two special cases, by providing the following generalization: There is a PTAS for $Pm|\,\mathrm{set}_j|\sum w_jC_j$ that computes, for any fixed m and $\varepsilon > 0$ accuracy, a $(1+\varepsilon)$-approximate schedule in $O(n\log n)$ time.

This work was motivated by [ABC$^+$99], where it was also announced that there is a PTAS for scheduling on unrelated parallel machines with release dates $Rm|r_j|\sum w_jC_j$, and our very recent work [FJP01b], where we have shown the existence of a PTAS for $Pm|\,\mathrm{fix}_j, r_j|\sum w_jC_j$. Our original goal was to provide a generalization for all previous results on scheduling problems involving a fixed number of processors (machines), release dates and the average weighted completion time objective. Here we do not achieve this goal completely, but our result provide hopefully a major step towards it. We conjecture that there is a PTAS for $Pm|\,\mathrm{set}_j, r_j|\sum w_jC_j$.

In order to build PTASs, here we also follow three main parts of our method: (1) Structuring (2) Compacting, and (3) Dynamic Programming. Interestingly, similar to the job shop problem, we make almost no chances in parts (2) and (3), but in part (1) we add some novel non-trivial ideas which were not used before.

There is one very important feature of the multiprocessor tasks scheduling problem which contrasts it from all previously considered problems. Here, a task can

require more than one processor. Thus, while tasks run in a schedule, more "earlier" tasks can intersect in a very irregular way blocking more "later" tasks. This creates some difficulties in using the already developed techniques. However, in order to be able to cope with multiprocessor tasks we can enforce regular *gaps* in a schedule. A gap is an interval in time where all the processors are idle. This simple idea leads to PTASs for the dedicated and parallel tasks scheduling problems with release dates and a PTAS for the general multiprocessor tasks scheduling problem without release dates.

Similarly to the one machine case, we can round tasks and classify tasks as huge and tiny. Similarly to the job shop case, we can define profiles and prioritizing tiny tasks of the same profile by Smith's rule. However, in order to prove the later result, we again use an LP formulation, a rounding procedure, and a PTAS for the makespan version of the problem [ABKM97, CM99, JP99b].

Unfortunately, our method fails in the case of general multiprocessor tasks with release dates. Rounding of general multiprocessor tasks without release dates is quite simple, but as soon as release dates are introduced, the problem becomes more complex. Here we need some new ideas for defining profiles, tiny and huge tasks, and Smith's rule.

Organization of the Chapter. The rest of the chapter is organized as follows. In the next section we give a short overview of the input transformation technique, which can be also found in [SW02]. In Section 1.3 we consider the problem of scheduling on a single machine. In Section 1.4 we consider the job shop problem, and in Section 1.5 we discuss the multiprocessor tasks scheduling problem. In Section 1.6 we give concluding remarks.

1.2 DESIGNING A PTAS: THE INPUT TRANSFORMATION TECHNIQUE

Suppose we want to find a PTAS for a strongly NP-hard scheduling problem. How we should proceed?

We know that any approximation algorithm A should take an instance I, process it for some time, and finally output an approximate (near-optimal) solution $App(I)$. Indeed, all known approximation schemes are based on the diagram depicted in Figure 1.1 on the following page consisting of three well-separated parts: The input I on the left, the output $App(I)$ on the right, and the algorithm A in the middle.

However, one can find a number of different approaches to the construction of approximation schemes. One of the standard approaches is the so-called technique of

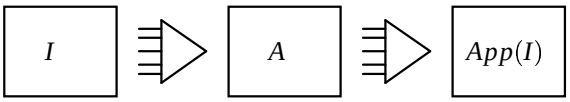


Figure 1.1: Solving of the input.

simplifying of the instance. Here, one first turns a difficult instance into a more *simpler* instance which is easier to tackle, and then uses an optimal (or approximate) solution for this instance in finding an approximate solution the original instance.

Less formally, we add a bit more *structure* to the diagram in Figure 1.1. For an illustration see Figure 1.2. Here, due to the NP-hardness of our scheduling problem, instance I is very complicated and irregularly shaped, and it would be difficult to go directly form I to an approximate solution $App(I)$. Hence, one takes the detour via a simplified instance $I^{\#}$ for which it is easy to obtain an optimal solution $OPT(I^{\#})$ or an approximate solution $App(I^{\#})$. Finally, one translates $OPT(I^{\#})$ or $App(I^{\#})$ into an approximate solution $App(I)$ for the original instance I.

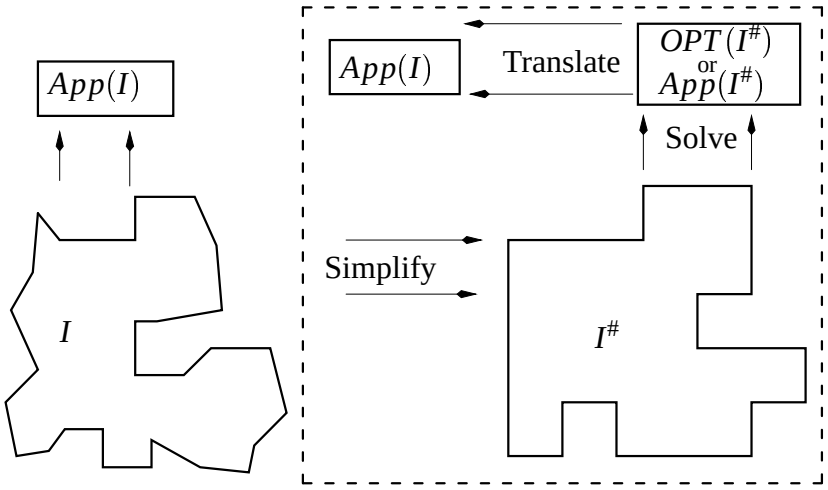


Figure 1.2: Simplifying of the instance

The approach described can be presented by the following two-phase procedure:

(A) Simplify: Simplify instance I to an instance $I^{\#}$. The simplification depends on the desired accuracy $\varepsilon > 0$ of approximation; the closer ε is to zero, the

closer instance $I^{\#}$ should resemble instance I. The time needed to transform I into $I^{\#}$ must be polynomial in the size of I (it can be exponential in $1/\varepsilon$).

(B) Solve: Determine an optimal solution $OPT(I^{\#})$ or a near-optimal solution $App(I^{\#})$ for $I^{\#}$, and then translate $OPT(I^{\#})$ or $App(I^{\#})$ back into an approximation solution $App(I)$. It should be done in polynomial time in the size of I (it can be exponential in $1/\varepsilon$), and solution $App(I)$ should stay close to $OPT(I)$.

Of course, finding the right simplification in step **A** is an art. On one hand, if $I^{\#}$ is too close to I, then $I^{\#}$ can be still hard to solve to optimality. On the other hand, if $I^{\#}$ is too far away from $I^{\#}$, then solving $I^{\#}$ will not tell us anything about how to solve I. The following techniques for simplifying the input instance often work well.

Rounding. The simplest way of adding structure to the input is to round some of the numbers in the input. For instance, we may round all job lengths to perfect powers of two.

Merging. Another way of adding structure is to merge small pieces into larger pieces of primitive shape. For instance, we may merge a large number of tiny jobs into a single job with processing time equal to the processing time of all tiny jobs.

Cutting. Yet another way of adding structure is to cut away irregular shaped pieces from the instance. For example, we may remove a small set of jobs with a broad spectrum of processing times from the instance.

In the following sections, we will combine these techniques, though named differently, into one approximating method which consists of three main parts in our method: (1) Structuring, (2) Compacting, and (3) Dynamic Programming. Here, parts (1) and (2) correspond to phase **(A)**, and part (3) corresponds to phase **(B)** of the input transformation technique.

Informally, given a fixed accuracy $\varepsilon > 0$ and an instance I of size n we proceed as follows. In parts (1) and (2), we construct an instance $I^{\#}$ from an instance I of the problem in $O(n \log n)$ time. In part (3), we find a near-optimal solution for $I^{\#}$ by using dynamic programing in $O(n)$ time. The objective value of the found solution is within a factor of $1 + O(\varepsilon)$ of the original optimum $OPT(I)$, and the algorithm derived is a PTAS with $O(n \log n)$ running time (it can be exponential in $1/\varepsilon$).

1.3 SCHEDULING ON A SINGLE MACHINE: A REFINED PTAS

In this section we consider the following problem of scheduling jobs on a single machine: Given n jobs, where job j ($j = 1, \ldots, n$) has a processing time p_j, a positive weight w_j and a release date r_j, schedule the jobs on a single machine so as to minimize $\sum w_j C_j$, where C_j denotes the completion time of job j. For an illustration see Figure 1.3.

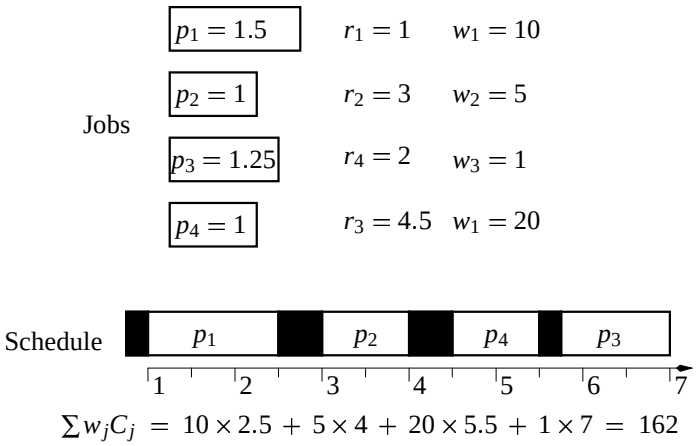


Figure 1.3: A schedule for 4 jobs

A very simple idea can be to use a dynamic programming algorithm which works over release dates and enumerates all possible schedules for the jobs released at each distinct date. However, there are two potential difficulties which can lead to an exponential in $O(n)$ running time. First, it can happen that "too many" jobs get released at the same time. Second, it can happen that some jobs wait "too long" time for processing.

Indeed, in order to "speed up" the running time we will avoid both these extreme cases. We use the technique of simplifying of the instance. Our main goal is to structure the problem such that there is at most a "constant" number of jobs at each distinct release date, and then to find a near-optimal schedule in which every job completes "close" to its release date.

Informally, we perform several *transformations* that simplify the input problem. Many of our transformations are thought modifications applied to the optimal schedule to argue that some schedule nearly as good has very simple structure. For example, we can modify an optimal schedule such that all starting times $S_j \geq \varepsilon p_j$ and the objective value increases by at most a factor of $1+\varepsilon$. Then, we say that *with* $1+\varepsilon$ *loss*, we can *assume* that all $S_j \geq \varepsilon p_j$ in any schedule. Others our

transformations are actual simplifying modifications of the instance that run in polynomial time and do not increase the objective value too much. For example, in $O(n)$ time we can round all processing times p_j to integer powers of $1+\varepsilon$. Then, we say that *with* $1+\varepsilon$ *loss* and in $O(n)$ time, we can *enforce* all p_j be integer powers of $1+\varepsilon$.

As we discussed, there are three main parts of our approximation method: (1) Structuring, (2) Compacting, and (3) Dynamic Programming. In part (1), Sections 1.3.1,1.3.2, 1.3.3,1.3.4,1.3.5,1.3.6, we derive several transformations that add more structure to the problem. In part (2), Sections 1.3.7,1.3.8, we provide two main transformations for compacting the problem instance. In part (3), Section 1.3.9,1.3.10, we give a dynamic programming algorithm for finding a near-optimal schedule.

As a preliminary step of part (1), in Section 1.3.1 we perform basic structuring of instances and schedules. The first transformation is *geometric-rounding*: we round all release dates r_j and processing times p_j to integer powers of $1+\varepsilon$, and perturb weights w_j such that all w_j/p_j are distinct (Lemmas 1.3.2 and 1.3.3). This guarantees that there are only a small number of distinct processing times and release dates to worry about, and lets us break the time line into *geometrically increasing intervals*, where release dates only happen at the beginning of intervals, that is useful for dynamic programming. Our second transformation is *schedule-stretching*: we multiply all job completion times by $1+\varepsilon$ and increase the starting times to match. From one side, this increases the objective function by a factor of $1+\varepsilon$. From another side, assuming that every job does not start too early (Lemma 1.3.6), we can define a relationship between release dates and processing times. Furthermore, assuming that no job can cross too many intervals (Lemma 1.3.4), we can define *crossing jobs*.

As an intermediate step of part (1), in Section 1.3.2 we classify jobs into *tiny* and *huge* ones. In particular, a job is *huge* in an interval if its length is at least ε^2 times the size of the interval, and *tiny* otherwise. Accordingly, there is at most a constant number of huge job sizes, that are powers of $1+\varepsilon$ (Lemma 1.3.7). Our next transformation is *time-stretching*: we add small amounts of idle time into intervals. This can be used to "clean up" the schedule. With $1+\varepsilon$ loss, we can assume that no tiny job crosses an interval in a schedule (Lemma 1.3.8).

As the final step of part (1), we perform several transformations that structure scheduling of tiny jobs. In total, Section 1.3.3, we can assume that all tiny jobs "obey" *Smith's rule*: if two tiny jobs k and j such that $w_j/p_j < w_k/p_k$ are available in an interval, then job k of greater value w_k/ℓ_k completes not later than job j with respect to intervals (Lemma 1.3.9). Informally, in order to prove this, we first use a subroutine for "assigning" tiny jobs to intervals, and then as a subroutine for

"packing" jobs in single intervals. In Section 1.3.4, we take an optimal schedule and formulate an linear programming (LP) which gives a fractional assignment of tiny jobs to intervals. In Section 1.3.5, we round an optimal LP solution to an integral assignment in which tiny jobs obey Smith's rule. Rounding does not increase the objective value and can only lead to small increase in each "interval load" (Lemma 1.3.11). In Section 1.3.6, we apply the time-stretching technique and reschedule tiny jobs in the optimal schedule with respect to the found integral assignment. These transformations increase the objective value by at most a factor of $1+7\varepsilon$.

After these three steps of part (1), we can simply proceed throughout part (2) of our method described in Sections 1.3.7 and 1.3.8. The first transformation here is *weight-shifting* (Lemma 1.3.13). If many jobs are released at one date, we know that some of them will have to wait to be processed. Shifting refers to the process of moving the excess jobs in the current interval to the next interval. For tiny jobs, we prioritize them in order of decreasing ratio w_j/p_j, and retaining only those that can be executed. For large jobs of each particular size (power of $1+\varepsilon$), we prioritize them in order of decreasing weights w_j and simply retain the maximum number that could be potentially scheduled. Our second transformation here is *merging* (Lemma 1.3.14). We first take the set of tiny jobs released at the same date. Then, we partition this ordered set into a constant number of roughly equal subsets. Merging refers to the process of grouping the tiny jobs of each subset into one single tiny job. Both sifting and merging can be done with $1+O(\varepsilon)$ loss and in $O(n\log n)$ time. Hence, we can enforce that there is at most a constant number of jobs released at each interval, and their total processing time is a small multiple of the size of the interval.

At this point, in Sections 1.3.9 and 1.3.10, we complete the algorithm by proceeding part (3) of our method. We first use time-stretching. We show that every job can be completed within a constant number of intervals after its release date in a near-optimal schedule (Lemma 1.3.15). Our next idea here is to find such a near-optimal schedule by using dynamic programming. Accordingly, we define special *block* structure over intervals and derive recurrent equations (1.8) on page 38 which define the dynamic programming framework. The corresponding dynamic programming algorithm for our problem runs in $O(n)$ time (Lemma 1.3.16). In total, we prove the following result:

Theorem 1.3.1. *There is a PTAS for $1|r_j|\sum w_jC_j$ that computes for any fixed $\varepsilon>0$ accuracy, a $(1+\varepsilon)$-approximate schedule in $O(n\log n)$ time.*

1.3.1 Basic Structuring

To simplify notations we will use throughout this section that $1/\varepsilon$ is integer (and in particular $\varepsilon < 1/4$). We use C_j and S_j to denote the completion and start time of job j, OPT to denote the objective value of the optimal schedule. For a job set X, we use $p(X)$ to denote the total processing time of the jobs of X.

Geometric Rounding. First, we use *geometric rounding* to create a well-structured set of processing times and release dates.

Lemma 1.3.2. *With $1+\varepsilon$ loss and in $O(n)$ time, we can enforce all r_j and p_j be integer powers of $1+\varepsilon$.*

Proof. For an illustration see Figure 1.4. Take an optimal schedule of objective value OPT, Figure 1.4 a). Then, multiply all r_j and p_j by $1+\varepsilon$. The optimal objective value is $(1+\varepsilon)OPT$, Figure 1.4 b). Use εr_j and εp_j to *round* (decrease) $p_j(1+\varepsilon)$ and $r_j(1+\varepsilon)$ to the next *lower* integer powers of $1+\varepsilon$. This can only decrease the objective value of the optimal schedule. □

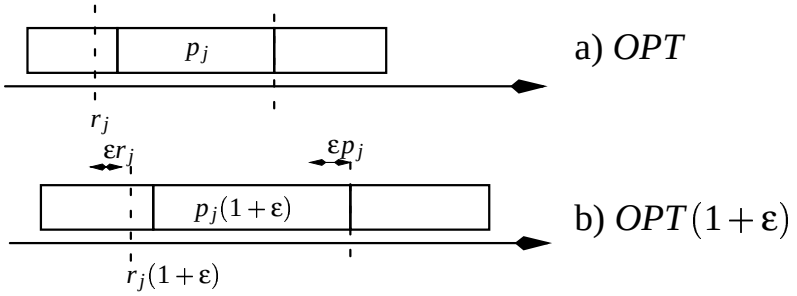


Figure 1.4: Geometric rounding

Intervals. For an arbitrary integer x, define $R_x = (1+\varepsilon)^x$. Now, all release dates are of the form $R_x = (1+\varepsilon)^x$ for some integer x. For an illustration see Figure 1.5 on the next page. We partition $(0,\infty)$ into disjoint intervals of the form $I_x := [R_x, R_{x+1})$. We will use $|I|_x$ to refer to the size $(R_{x+1} - R_x)$ of interval I_x.

There are two very simple properties of intervals

$$|I|_x \;=\; \varepsilon R_x \;\text{ and }\; |I|_{x+1} \;=\; (1+\varepsilon)|I|_x.$$

This means that the size of each interval is ε times its start time, and the size of two consecutive intervals differs by a factor of $1+\varepsilon$.

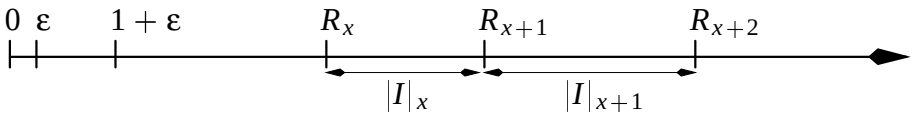


Figure 1.5: Intervals

Weights. Similarly, we can perturb weights.

Lemma 1.3.3. *With* $1+\varepsilon$ *loss and in at most* $O(n \log n)$ *time, we can enforce all* w_j/p_j *be distinct.*

Proof. Multiply all w_j by $1+\varepsilon$, and then *decrease* (round) some of them by small values until all w_j/p_j are distinct. □

Schedule-Stretching. Next, we show that jobs cannot neither start nor be released too early. This will help in the later analysis.

Lemma 1.3.4. *With* $1+\varepsilon$ *loss, we can assume that all starting times* $S_j \geq \varepsilon p_j$ *in a schedule.*

Proof. For an illustration see Figure 1.6 on the facing page. Take an optimal schedule of objective value OPT, Figure 1.6 a). Multiply all C_j by $1+\varepsilon$ and *increase* S_j to *match* without changing job processing times, Figure 1.6 b). The objective value of the final schedule is at most $(1+\varepsilon)OPT$, and all starting times

$$(1+\varepsilon)C_j - p_j \geq (1+\varepsilon)p_j - p_j$$

are at least εp_j. □

As a consequence, we can move all release dates as follows:

Lemma 1.3.5. *With* $1+\varepsilon$ *loss and in* $O(n)$ *time, we can enforce* $r_j \geq \varepsilon p_j$ *for all jobs* j*.*

Proof. By Lemma 1.3.4 all $S_j \geq \varepsilon p_j$ in an optimal schedule. Then, w.l.o.g. we can simply move each release date r_j by εp_j. □

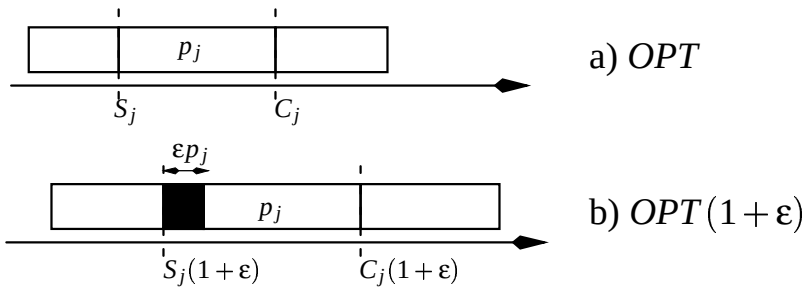


Figure 1.6: Stretching of a schedule

Crossing Jobs. Now we can prove that no job can cross too many intervals:

Lemma 1.3.6. *With* $1+\varepsilon$ *loss, we can assume that each job crosses at most*

$$s^* = \left\lceil \log_{1+\varepsilon}(1+\frac{1}{\varepsilon}) \right\rceil$$

intervals in a schedule.

Proof. Take a schedule. By Lemma 1.3.4 all $S_j \geq \varepsilon p_j$. Consider a crossing job j. For an illustration see Figure 1.7. Let $S_j \in I_x = [R_x, R_{x+1})$. Then, completion time

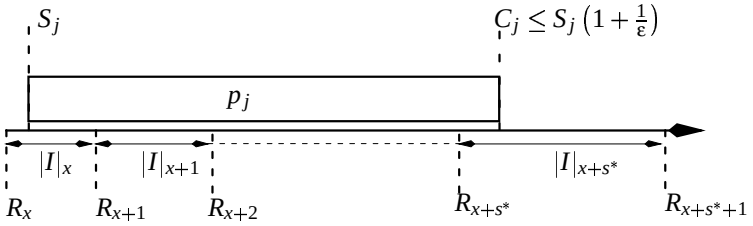


Figure 1.7: A crossing job j

$$C_j \;=\; S_j \;+\; p_j \;\leq\; S_j\left(1+\frac{1}{\varepsilon}\right),$$

and processing time

$$p_j \;\leq\; S_j/\varepsilon \;\leq\; R_{x+1}/\varepsilon.$$

Let $s^* = \lceil \log_{1+\varepsilon}(1+\frac{1}{\varepsilon}) \rceil$. Consider s^* intervals which follow I_x. Their total length can be bounded as

$$\sum_{i=x+1}^{x+s^*} |I|_i \;=\; R_{x+s^*+1} - R_{x+1} \;=\; R_{x+1}((1+\varepsilon)^{s^*} \;-\; 1) \;\geq\; R_{x+1}/\varepsilon.$$

Hence, these s^* intervals cover $[S_j, C_j]$. □

Notice that we do not use the above fact in this section. However, this will be used later in a generalized form.

1.3.2 Huge and Tiny Jobs

Now we introduce two different types of jobs. We say that job j is *huge* in an interval I_x if its processing time $p_j \geq \varepsilon^2 |I|_x$, and *tiny* otherwise. We will write H_x and T_x to denote sets of huge and tiny jobs released at R_x (H for huge and T for tiny).

Lemma 1.3.7. *There are at most $2/\varepsilon^4$ distinct sizes (p_j powers of $1+\varepsilon$) in H_x.*

Proof. We have $p_j \leq r_j/\varepsilon = R_x/\varepsilon$ for any job j (Lemma 1.3.5). Then, for a huge job j we have

$$\varepsilon^2 |I|_{x(j)} \;=\; \varepsilon^3 R_{x(j)} \;\leq\; p_j \;\leq\; R_{x(j)}/\varepsilon.$$

Let $p_j = R_{x(j)}(1+\varepsilon)^s$, for some integer s (Lemma 1.3.2). Then, we have

$$\varepsilon^3 R_{x(j)} \;\leq R_{x(j)}(1+\varepsilon)^s \;\leq R_{x(j)}/\varepsilon.$$

Hence,

$$s \in [-3\log 1/\varepsilon, \log 1/\varepsilon],$$

and k is integer. From

$$4\lceil \log 1/\varepsilon \rceil + 1 \leq 2\log 1/\varepsilon^4 \leq 2/\varepsilon^4$$

we can state that there are at most $2/\varepsilon^4$ distinct huge job sizes s (powers of $1+\varepsilon$). □

Time-Stretching. Now we introduce the *time-stretching* technique. For an illustration see Figure 1.8 on the facing page. The formal description is given in the proof of Lemma 1.3.8. Less formally, we can describe the technique as follows. Take an optimal schedule, w.l.o.g. in intervals $I_1, I_2, I_3, \ldots$, see Figure 1.8 a). Increase the size of each of I_x, $x = 1, 2, \ldots$ by $1+\varepsilon$. This creates $\varepsilon|I|_x$ idle time in each interval I_x, and increases the objective function value at most $1+\varepsilon$ factor, see Figure 1.8 b).

By using this simple idea we can prove the following:

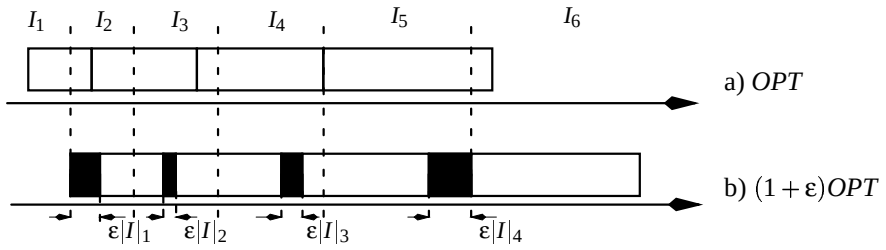


Figure 1.8: Time-stretching

Lemma 1.3.8. *With* $1+\varepsilon$ *loss, we can assume that no tiny job crosses an interval in a schedule.*

Proof. For an illustration see Figure 1.9. Take an optimal schedule of value *OPT*. Start from its first interval and go further up to the end. Assume that we meet a tiny job j_b in an interval I_b such that $R_b \in (S_{j_b}, C_{j_b}]$. In other words, job j_b crosses I_b, see Figure 1.9 a), and its processing time $p_{j_b} < \varepsilon|I|_b$. Then, start this job j_b exactly at R_b by moving all later jobs, see Figure 1.9 b).

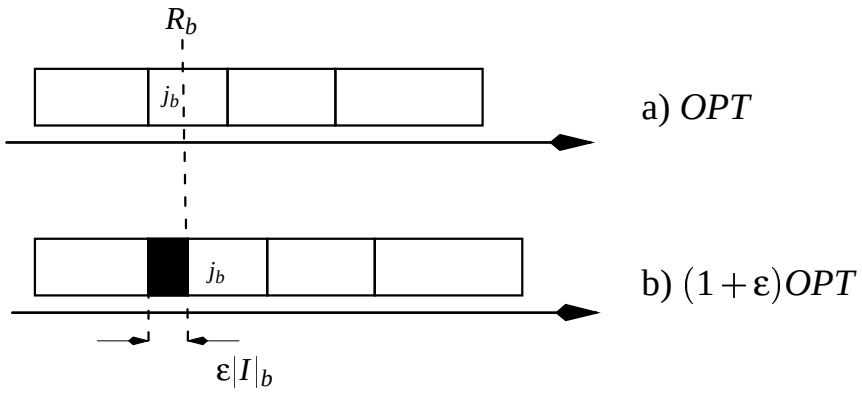


Figure 1.9: A job j_b with $R_b \in (S_{j_b}, C_{j_b}]$

Clearly, no tiny job crosses an interval in the final schedule. Observe also that this procedure increases *OPT* by at most a factor of $1+\varepsilon$. For an illustration see Figure 1.10 on the next page. Consider any job j in the optimal schedule. Let $C_j \in I_x = [R_x, R_{x+1})$, see Figure 1.10 a). Then, we have moved this jobs by at most

$$\sum_{b<x} p_{j_b} \;\leq\; \sum_{b<x} \varepsilon|I|_b \;=\; \varepsilon|I|_x \sum_{t\geq 1} \frac{1}{(1+\varepsilon)^t} \;\leq\; \varepsilon|I|_x \frac{1}{\varepsilon} \;=\; |I|_x \;=\; \varepsilon R_x.$$

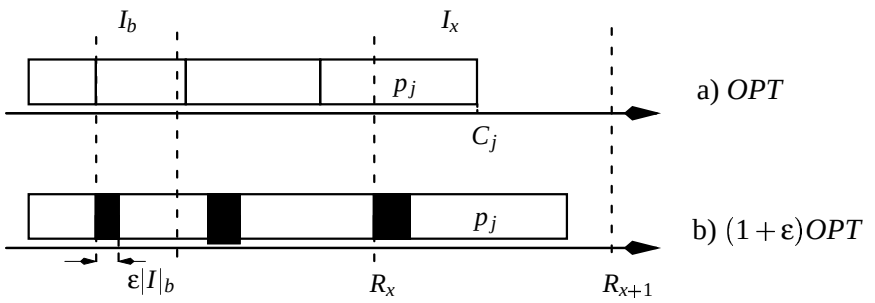


Figure 1.10: A job j with $C_j \in I_x$

Since $\varepsilon R_x \leq \varepsilon C_j$, the completion time of job j increases by at most a factor of $1+\varepsilon$, see Figure 1.10 b). □

1.3.3 Scheduling Tiny Jobs: Smith's Rule

We first need to introduce some notations. By Lemma 1.3.8 no tiny job crosses an interval in a schedule. Then, for a tiny job j we define two indices $x(j) \leq y(j)$ such that $R_{x(j)} = r_j$ and $S_j, C_j \in I_{y(j)}$. Less formally, job j is released at $R_{x(j)}$ and completely scheduled in interval $I_{y(j)}$.

Smith's Rule. Let j and k be two tiny jobs such that $x(k) \leq x(j)$ (here $r_k \leq r_j$) and $\frac{w_k}{p_k} > \frac{w_j}{p_j}$ (see Lemma 1.3.3). We say that tiny jobs obey *Smith's rule* in a schedule if $y(k) \leq y(j)$ ($S_k \leq S_j$) for all such pairs of jobs j and k. In other words, if jobs k, j are available in an interval, then job k with greater value w_k/p_k starts not later than job j with respect to intervals $I_{y(k)}$ and $I_{y(j)}$.

Now we can prove the following:

Lemma 1.3.9. *With $1+7\varepsilon$ loss, we can assume that all tiny jobs obey Smith's rule in a schedule.*

Proof. The proof is given in Sections 1.3.4,1.3.5,1.3.6. Informally, we first use a subroutine for "assigning" tiny jobs to intervals, and then as a subroutine for "packing" jobs in single intervals. Here, assigning is based on an LP formulation and a rounding procedure, whereas packing procedure is based on the time-stretching technique. We can briefly sketch the proof as follows.

We take an optimal schedule. For each tiny job j we find index $y(j)$. By using $y(j)$, for each interval I_y we find a set Y_y of tiny jobs processed inside I_y.

Next, by using the values of $D_y = p(Y_y)$ we formulate a linear program (LP) which defines a fractional assignment of tiny jobs j to intervals I_y. Then, we round an optimal LP solution to an integral assignment.

By using this integral assignment, for each tiny job j we can define new index $y(j)$, and for each interval I_y we can define a new set $\widetilde{Y}_y$ of tiny jobs assigned to I_y. From one side, tiny jobs j obey Smith's rule with respect to these $y(j)$. From another side, the rounding procedure does not increase the objective value, but increases the value of each D_y by at most $\varepsilon^2|I|_y$.

We notice that

$$p(\widetilde{Y}_y) \;\leq\; D_y \;+\; \varepsilon^2|I|_y \;=\; p(Y_y) \;+\; \varepsilon^2|I|_y.$$

We simply take the optimal schedule and use time-stretching. We add $\varepsilon|I|_y$ idle time in each interval I_y. Then, we replace the tiny jobs of Y_y by the tiny jobs of $\widetilde{Y}_y$ inside each interval I_{y+1}. This gives us a schedule with the objective function value at most $(1+7\varepsilon)OPT$ in which all tiny jobs obey Smith's rule. □

1.3.4 Assigning to Intervals: LP formulation

Consider an optimal schedule of objective value OPT. Then, for each tiny job j we define two indices $x(j) \leq y(j)$ such that $R_{x(j)} = r_j$ and $S_j, C_j \in I_{y(j)}$. In addition, for each interval I_y, we define set Y_y of tiny jobs scheduled inside I_y, i.e. $Y_y = \{j | y(j) = y\}$.

Now we can formulate the fractional assignment problem for all tiny jobs as the following LP:

$$\begin{aligned}
\textbf{Minimize} \quad & F(\xi) \;=\; \textstyle\sum_j w_j \sum_{y \geq x(j)} \xi_{jy} \cdot R_y \\
\text{s.t. (1)} \quad & \Delta_y = \textstyle\sum_j \xi_{jy} \cdot p_j \;\leq\; D_y = p(Y_y) \;\text{ for all } y, \\
(2) \quad & \textstyle\sum_{y \geq x(j)} \xi_{jy} \;=\; 1 \;\text{ for all } j, \\
(3) \quad & \xi_{jy} \;\geq\; 0 \;\text{ for all } y \geq x(j) \text{ and } j,
\end{aligned} \tag{1.1}$$

where

ξ_{jy}**:** the yth fraction of tiny job j assigned to interval $I_y = [R_y, R_{y+1})$,

D_y**:** the load in interval $I_y = [R_y, R_{y+1})$,

Δ_y**:** the fractional load in interval $I_y = [R_y, R_{y+1})$, and

$F(\xi)$**:** the fractional average weighted completion time of tiny jobs.

Accordingly, the LP constraints have the following meaning: **(1)** the value of fractional load Δ_y in each interval I_y is at most D_y, and **(2)-(3)** each tiny job j is assigned completely to intervals I_y, $y \geq x(j)$.

Informally, in the optimal schedule we allow tiny jobs be fractionally processed in several intervals. From one side, we assign each yth fraction of job j be completed in interval $I_y = [R_y, R_{y+1})$, and take R_y as its "completion time". From another side, preserving "loads" $D_y = p(Y_y)$ in all intervals I_y, we keep the schedule of huge jobs without changes.

We can bound the fractional average weighted completion time as follows:

Lemma 1.3.10. *For any optimal solution ξ of the LP it holds that $F(\xi) \leq OPT'$, where OPT' is the total weighted completion time of all tiny jobs in the optimal schedule.*

Proof. Consider all tiny jobs j in an optimal schedule. We define indices $x(j) \leq y(j)$ such that $R_{x(j)} = r_j$ and $S_j, C_j \in I_{y(j)}$. Since $y(j) \geq x(j)$, for an assignment $\xi = (\xi_{jy})$ we can define $\xi_{jy} = 1$ if $y = y(j)$ and $\xi_{jy} = 0$ otherwise. Then, ξ is feasible. Furthermore, since $C_j \in I_{y(j)} = [R_{y(j)}, R_{y(j)+1})$ we have

$$F(\xi) \;=\; \sum_j w_j \cdot R_{y(j)} \;\leq\; \sum_j w_j \cdot C_j \;=\; OPT'.$$

□

1.3.5 Assigning to Intervals: LP Rounding

Let $\xi = (\xi_{jy})$ be an optimal LP solution. If ξ is integral, then for each tiny job j there is exactly one $y(j)$ such that $y(j) \geq x(j)$ and $\xi_{y(j)j} = 1$. In other words, an integral LP solution can be treated as an integral assignment of tiny jobs to intervals. Here we prove the following result:

Lemma 1.3.11. *An optimal solution ξ of the LP can be rounded into an integral assignment of tiny jobs to intervals such that*

(1) *$F(\xi)$ does not increase;*

(2) *the value of each Δ_y increases by at most $\varepsilon^2 |I|_y$;*

(3) *tiny jobs j obey the modified Smith's rule with respect to intervals $I_{y(j)}$ given by rounded ξ.*

Proof. Let $\xi = (\xi_{jy})$ be an optimal LP solution. Let j and k be two tiny jobs with $x(k) \leq x(j)$ and $\frac{w_j}{p_j} < \frac{w_k}{p_k}$ (that is $w_j p_k < w_k p_j$). Assume that there exist two fractions $\xi_{jy(j)} > 0$, $\xi_{ky(k)} > 0$ with $y(j) < y(k)$. Informally, this means that there are two fractions which break Smith's rule.

Then, there exist values t_j and t_k such that

$$0 < t_j \leq \xi_{jy(j)} \text{ and } 0 < t_k \leq \xi_{ky(k)}, \tag{1.2}$$

and

$$t_j p_j = t_k p_k. \tag{1.3}$$

We define a new solution $\xi' = (\xi'_{jy})$ as follows. First, we exchange the $y(j)$th and $y(k)$th fractions of jobs j and k

$$\begin{aligned} \xi'_{jy(j)} &= \xi_{jy(j)} - t_j \qquad \xi'_{jy(k)} = \xi_{jy(k)} + t_j \\ \xi'_{ky(k)} &= \xi_{ky(k)} - t_k \qquad \xi'_{ky(j)} = \xi_{ky(j)} + t_k. \end{aligned} \tag{1.4}$$

After this, we define the rest of ξ' as in ξ. For an illustration see Figure 1.11.

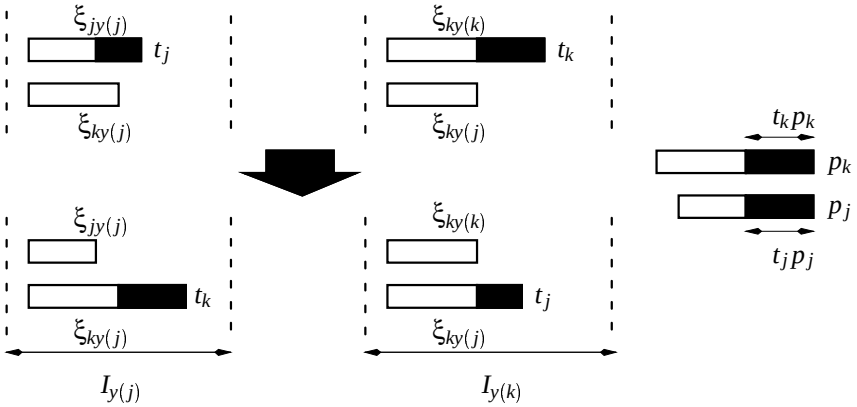


Figure 1.11: The $y(j)$,$y(k)$th fractions of jobs k and j

Due to (1.2), (1.3) and (1.4) we have that solution ξ' is feasible. Furthermore,

$$\begin{aligned} F(\xi') - F(\xi) &= R_{y(j)}(w_k t_k - w_j t_j) + R_{y(k)}(w_j t_j - w_k t_k) \\ &= (R_{y(j)} - R_{y(k)})(w_k t_k - t_j w_j). \end{aligned} \tag{1.5}$$

From $y(j) < y(k)$ we have $R_{y(j)} - R_{y(k)} < 0$. From $w_j p_k < w_k p_j$ and (1.3) we have $(w_k t_k - w_j t_j) > 0$. Thus, from (1.5) we conclude that

$$F(\xi') < F(\xi).$$

This is a contradiction to the optimality of ξ.

In the following we round ξ into an integral solution. We will use the above property of ξ. First, for each tiny job j we find the earliest index $y(j)$ with $\xi_{jy(j)} > 0$. Then, we simply put $\xi_{jy(j)} = 1$ for each such $y(j)$, and $\xi_{jy} = 0$ for others y. Clearly, ξ is integral.

Let j and k be two tiny jobs with $\frac{w_j}{p_j} < \frac{w_k}{p_k}$ and $x(k) \leq x(j)$. Then, due to the above property of ξ, it holds $y(k) \leq y(j)$ for any such pairs of jobs j and k. Thus, all tiny jobs j obey Smith's rule with respect to $y(j)$ given by ξ.

One can see that we cannot increase the value of $F(\xi)$. From another side, we can violate some constrains (2) of the LP. However, we can again use the above property of ξ. For an interval I_y, there is at most one tiny job, say j_y, which does not "fit" into D_y. One can also think that j_y "crosses" I_y. Since $p_{j_y} \leq \varepsilon^2 |I|_y$, each load Δ_y increases by at most $\varepsilon^2 |I|_y$. □

Now we use an integral assignments ξ. For each tiny job j we define new index $y(j) \geq x(j)$, respectively. Then, by using $y(j)$, for each interval I_y, we define the set $\widetilde{Y}_y$ of tiny jobs that are assigned to I_y. Then, we can prove the following:

Lemma 1.3.12. *For all tiny jobs j it holds that $y(j) \geq x(j)$ and*

$$\sum w_j R_{y(j)} \leq OPT'. \tag{1.6}$$

Furthermore, for each interval I_y it holds

$$p(\widetilde{Y}_y) \leq p(Y_y) + \varepsilon^2 |I|_y. \tag{1.7}$$

Proof. By Lemmas 1.3.10 and 1.3.11, for all tiny jobs it holds that

$$\sum w_j R_{y(j)} = F(\xi) \leq OPT'.$$

To define sets $\widetilde{Y}_y$, we have used a rounded solution ξ for the LP. Hence, it holds that

$$\Delta_y = p(\widetilde{Y}_y).$$

Then, by the LP formulation and Lemma 1.3.11 it holds that

$$y(j) \geq x(j)$$

and

$$\begin{aligned}\Delta_y &\leq D_y + \varepsilon^2|I|_y \\ &\leq p(Y_y) + \varepsilon^2|I|_y.\end{aligned}$$

□

1.3.6 Packing in Single Intervals

Here, by using the results of Lemmas 1.3.11 and 1.3.12, we complete the proof of Lemma 1.3.9.

We first take an optimal schedule of value OPT. Then, for each tiny job j we define two indices $x(j) \leq y(j)$ such that $R_{x(j)} = r_j$ and $S_j, C_j \in I_{y(j)}$. In addition, for each interval I_y, we define set Y_y of tiny jobs scheduled inside I_y, i.e. $Y_y = \{j | y(j) = y\}$, and define $D_y = p(Y_y)$.

Next, we formulate the LP and round an optimal solution. Let ξ be an integral assignment as defined in Lemma 1.3.11. Then, for each tiny job j we define new index $y(j)$ such that $y(j) \geq x(j)$ and $\xi_{jy(j)} = 1$. For each interval I_y, we also define $\widetilde{Y}_y = \{j | y(j) = y\}$.

From one side, all tiny jobs j obey Smith's rule with respect to these new $I_{y(j)}$. From another side, by Lemma 1.3.12 it holds (1.6) and (1.7).

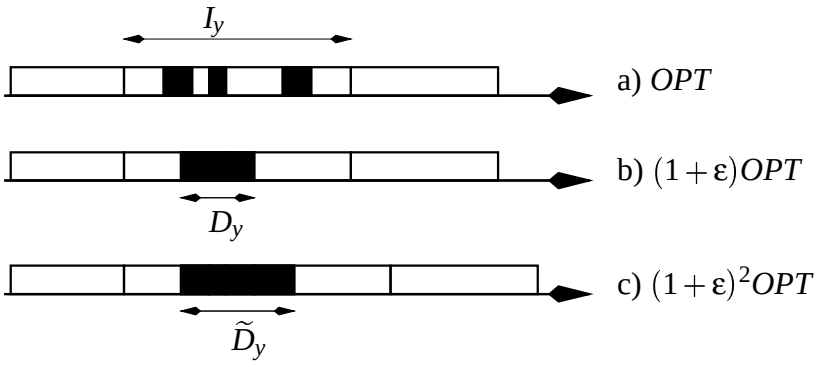


Figure 1.12: Scheduling inside interval I_y

Now we proceed as follows. For an illustration see Figure 1.12. We take the optimal schedule, see Figure 1.12 a), and move jobs processed inside each interval I_y such that the tiny jobs of Y_y are placed together in a gap of value $D_y = p(Y_y)$,

Figure 1.12 b). This increases the objective function value by at most a factor of $1+\varepsilon$.

Next, we apply time-stretching to the schedule, Figure 1.12 c). In each interval I_y, we add $\varepsilon|I|_y$ idle time to the gap of value D_y. The objective function value increases by at most a factor of $1+\varepsilon$, but there is a gap of length $D_y+\varepsilon|I|_y$ in each interval I_{y+1} of the schedule.

Finally, we reschedule all tiny jobs. For each interval I_{y+1}, we simply complete the jobs of $\widetilde{Y}_y$ inside a gap of length

$$\widetilde{D}_y \;=\; p(\widetilde{Y}_y) \;\leq\; D_y+\varepsilon^2|I|_y.$$

Once can see that each tiny job j completes at $C_j \in I_{y(j)+1}$. By Lemma 1.3.12 the average weighted completion time of all tiny jobs can be bounded as follows

$$\sum_j C_j \;\leq\; \sum_j (1\,+\,\varepsilon)\,R_{y(j)+1} \;=\; (1\,+\,\varepsilon)^2 OPT',$$

where OPT' is the total weighted completion time of all tiny jobs in the optimal schedule. Hence, the objective function value of the final schedule is at most

$$(1+\varepsilon)^3 OPT \;\leq\; (1+7\varepsilon)OPT.$$

This completes the proof of Lemma 1.3.9.

1.3.7 Weight-Shifting

Consider some release date R_x. Assume that there are "too many" huge jobs in H_x and tiny jobs in T_x. Which jobs have more higher priority and should be scheduled first?

By Lemma 1.3.3 all values w_j/p_j are distinct. From one side, by Lemma 1.3.7 the huge jobs of H_x have at most $2/\varepsilon^4$ distinct sizes (processing times). Hence, we can prioritize the huge jobs of H_x having the same size by ordering them in decreasing order of weights w_j. From another side, by Lemma 1.3.9 the tiny jobs of T_x obey Smith's rule. Hence, we can prioritize tiny jobs of T_x by ordering them in decreasing order of ratio w_j/p_j.

Indeed, there are at most $|I|_x$ available time in interval I_x. The processing time of any huge job in H_x is at least $\varepsilon^2|I|_x$. Hence, for each size we can select the first $2/\varepsilon^2$ high priority huge jobs and move the others huge job in H_x to the next release date R_{x+1}. Similarly, we can select tiny jobs with higher priority up to at most $2|I|_x$ total processing time, and move the others tiny jobs in T_x to R_{x+1}. As a result of the procedure we can prove the following:

Lemma 1.3.13. *With $1+O(\varepsilon)$ loss and in $O(n\log n)$ time, we can enforce $p(T_x) \leq 2|I|_x$ and $|H_x| \leq 4/\varepsilon^6$ for all x.*

1.3.8 Merging

By Lemma 1.3.13, for each release date R_x there is at most a constant number of jobs in H_x, but in T_x. However, jobs in T_x are tiny and their total processing time is bounded by $2|I|_x$. Our next transformation is merging tiny jobs together.

For each release date R_x, we partition the ordered set of tiny jobs T_x into at most $4/\varepsilon^2$ subsets of roughly equal size $\approx \varepsilon^2|I|_x/2$, but less than $\varepsilon^2|I|_x$. Then, we merge the jobs of each such subset into a new tiny job. Then, by using arguments similar to the ones in Lemma 1.3.8 we can prove the following:

Lemma 1.3.14. *With $1+O(\varepsilon)$ loss and in $O(n\log n)$ time, we can enforce $|T_x \cup H_x| \leq 8/\varepsilon^6$ for all x.*

Proof. Take an instance as in Lemma 1.3.13. Take an optimal schedule of value OPT. Let D_b be the time used to execute tiny jobs in interval I_b. Take a release date R_x. The tiny jobs of T_x are scheduled by Smith's rule. We replace the tasks of T_x by the new created jobs. For an interval I_b with $b > x$, it may happen that some new tiny job, say $j_x \in T_x$, does not fit into D_b. However, the processing time of each such job j_x, $x < b$, is at most $\varepsilon^2|I|_x$. Hence, the total overload in D_b caused by all jobs j_x, $x < b$ is at most

$$\sum_{x<b} p_{j_x} \;\leq\; \sum_{x<b} \varepsilon^2|I|_x = \varepsilon^2|I|_b \sum_{k\geq 1} \frac{1}{(1+\varepsilon)^k} \leq \varepsilon^2|I|_b/\varepsilon = \varepsilon|I|_b.$$

Then, we increase the value of D_b by adding $\varepsilon|I|_b$ idle time, and complete all new tiny jobs. The time-stretching technique implies that the value of the final schedule is at most $(1+\varepsilon)OPT$. □

1.3.9 Blocks

Lemmas 1.3.5 and 1.3.14 imply that the total number (if there is any) of jobs released at each R_x is bounded by $8/\varepsilon^8$. Thus, by Lemma 1.3.5 the total processing time of these jobs is bounded by $8|I|_x/\varepsilon^{10}$. Recall that we are dealing with intervals of increasing sizes. Thus, we can find a constant d^* such that $8|I|_x/\varepsilon^{10}$ is bounded by $\varepsilon^2|I|_{x+d^*}$. Hence, in interval I_{x+d^*} one can treat all jobs released at R_x as one tiny job. Again, the ideas of Lemma 1.3.8 and Lemma 1.3.14 lead to the following:

Lemma 1.3.15. *With $1+\varepsilon$ loss, we can assume that in a schedule each job completes within d^* intervals after its release date.*

We partition the time line into a sequence of *blocks*, where each block consists of d^* consecutive intervals.

1.3.10 The Dynamic Programming Framework

The basic idea here is to use dynamic programming with blocks as units. By lemma 1.3.15, the jobs of block i run either in block i or $i+1$. A *pseudo-schedule* $S^{(i)}$ describes a possible placement of the jobs of block i. For each job it is enough to define two intervals in which the job starts and completes.

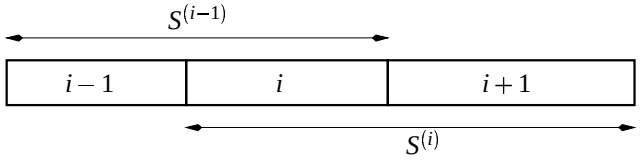


Figure 1.13: Pseudo-schedules $S^{(i)}$ and $S^{(i-1)}$ for block i

The dynamic programming entry $E(i,S^{(i)})$ stores the minimum weighted completion time achievable by completing all the jobs released before or in block i while leaving pseudo-schedule $S^{(i)}$ for block $i+1$. Given all table entries for $i-1$, the values for i can be computed as follows.

$$E(i,S^{(i)}) = \min_{S^{(i-1)}} \{E(i-1,S^{(i-1)}) + W(i,S^{(i-1)},S^{(i)})\}, \tag{1.8}$$

where $W(i,S^{(i-1)},S^{(i)})$ is the minimum weighted completion time achievable by scheduling the jobs in intervals of block i with respect to the *incoming* pseudo-schedule $S^{(i-1)}$ and the *outgoing* pseudo-schedule $S^{(i)}$, respectively. For an illustration see Figure 1.13.

By Lemmas 1.3.14,1.3.15 there is at most a constant number of jobs released in a block. Hence, both the feasibility test and computation of $W(i,S^{(i-1)},S^{(i)})$ can be done in $O(1)$ time. There are n jobs, each of which can cross at most s^* intervals (Lemma 1.3.6). Hence, there are at most $O(n)$ relevant blocks and we have the following:

Lemma 1.3.16. *The entire table $E(\cdot)$ can be computed in $O(n)$ time.*

Combining this result with Lemma 1.3.14 we complete the proof of Theorem 1.3.1.

1.4 A PTAS FOR THE JOB-SHOP SCHEDULING PROBLEM

In this section we consider the following job-shop scheduling problem. We consider the following job shop scheduling problem. We are given a set of n jobs $J = \{1,\dots,n\}$ and a set of m machines $M = \{1,\dots,m\}$. Each job j ($j = 1,\dots,n$) has a positive weight w_j, a release date r_j, and consists of a sequence of $\mu \geq 2$ operations $o_{1j}, o_{2j}, \dots, o_{\mu j}$ that must run in the given order. Each operation o_{ij} ($\mu = 1,\dots,\mu$) must run without interruption on a required machine $\tau_{ij} \in M$, during p_{ij} time units. Each job can be processed only by one machine at a time and each machine can only process one job at a time. Here we assume that m and μ are fixed, and the goal is to find a feasible schedule which minimizes average weighted completion time $\sum w_j C_j$, where C_j denotes the completion time of job j. For an illustration see Figure 1.14.

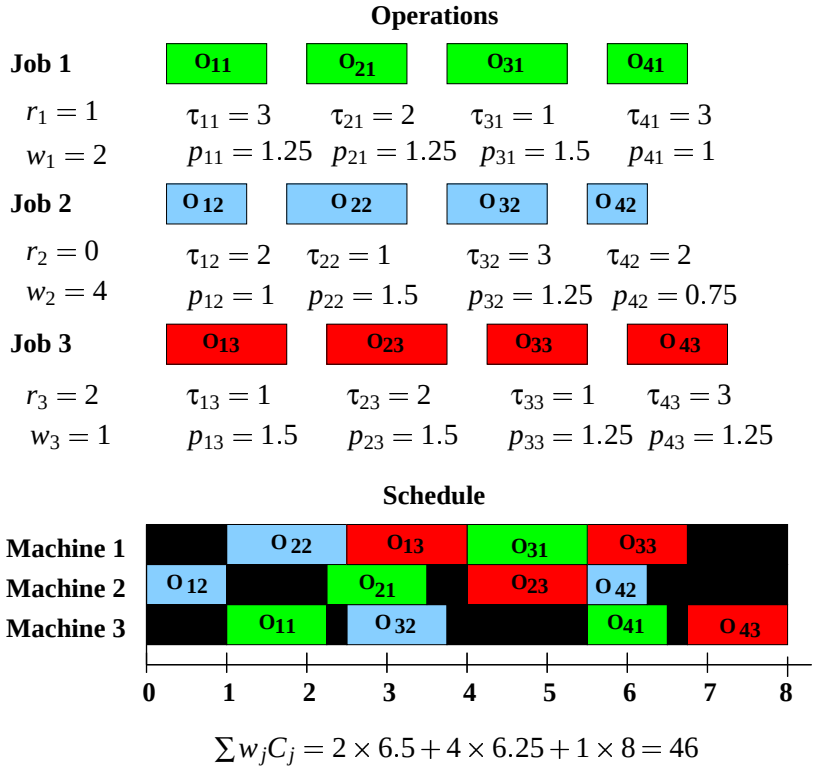


Figure 1.14: A schedule for 3 jobs

The above problem is a generalization of $1|r_j|\sum w_j C_j$. Here we also follow the main parts of our method: (1) Structuring, (2) Compacting, and, (3) Dynamic Programming. Indeed, we use ideas and techniques described in the single machine

case. We will make almost no changes in parts (2) and (3), Sections 1.4.9 and 1.4.10, but in part (1), Sections 1.4.1 – 1.4.8, we generalize the previous ideas for structuring in a non-trivial way.

In Section 1.4.1 we use already known rounding and stretching (Lemmas 1.4.2 – 1.4.5). We round release dates, weights and introduce intervals. However, we only deal with operations, that provides the basic structuring step of part (1). To complete the main structuring step, we use some new ideas throughout next three sections.

In Section 1.4.2, we partition operations of any job into two classes, the *main* and the *negligible* operations: the sum of processing times of negligible operations are less than $\varepsilon^{4\mu}$ times the processing time of any main operation. We show that for a job j the processing time of a main operation can be bounded from below by $\varepsilon^{5\mu^2} \cdot \ell_j$, where $\ell_j = \sum_{i=1}^{\mu} p_{ij}$ is the length of job j (Lemma 1.4.6).

In Section 1.4.3 by using different combinations of known techniques, we show that processing time of any negligible operation can be rounded to zero, and processing time of any main operation can be rounded to a ε^2 multiple of a small constant fraction of the total job length (Lemma 1.4.7). In addition, we round release dates (Lemma 1.4.8). By using this, we can conclude that no job crosses too many intervals (Lemma 1.4.9). Informally, this means that any job is processed "locally" in a schedule.

In Section 1.4.4, we give some notations and definitions that help in structuring instances and schedules. We first specify job *profiles*. Each profile φ is a 2μ-tuple which consists of two μ-tuples $\tau = (\tau_i)$ and $\pi = (\pi_i)$. If a job j has profile φ, then each operation o_{ij} $(i = 1, \ldots, \mu)$ requires $\pi_i \cdot \ell_j$ processing time on machine τ_i. We prove that there is a constant number of possible profiles (Lemma 1.4.10). Next, as in the single machine case, we classify jobs into *tiny* and *huge* ones. In particular, a job is huge in an interval if its length is at least ε^2/q^* times the size of interval, and huge otherwise. Here, parameter $q^* = q_1^* \cdot q_2^*$, where the values of q_1^*, q_2^* are defined later in Sections 1.4.7,1.4.8, respectively. For tiny jobs, we prove that no tiny operation crosses an interval (Lemma 1.4.11). For huge jobs, we prove that there is at most a constant number of huge job "sizes" at each release date (Lemma 1.4.12). Finally, for tiny jobs we introduce *local profiles* and *patterns*. Informally, each profile represents a set of all μ operations, whereas each local profiles represents a subset of the operations. Then, each pattern is a collection local profiles, that represents a way in which a tiny job can be processed inside several intervals of a schedule. We prove that there is at most a constant number of distinct local profiles and patterns (Lemma 1.4.13).

As the final step of part (1), similarly to the single machine case, we perform several transformations that structure scheduling of tiny jobs. In total, Section 1.4.5,

we can assume that tiny jobs of one profile φ "obey" Smith's rule: if two jobs k and j of profile φ with $w_j/\ell_j < w_k/\ell_k$ are available in an interval, then job k of greater value w_k/ℓ_k completes not later than job j with respect to intervals and patterns (Lemma 1.4.14). However, our non-trivial proof of this result spans over Sections 1.4.6,1.4.7,1.4.8.

Indeed, we proceed as in the single machine case. Informally, we first use a subroutine for "assigning" tiny jobs to intervals and patterns, and then as a subroutine for "packing" jobs in single intervals. In Section 1.4.6, we take an optimal schedule and formulate an LP(φ) which defines a fractional assignment of tiny jobs of profile φ to patterns and intervals. In Section 1.4.7, we round an optimal LP solution to an integral assignment in which tiny jobs of profile φ obey Smith's rule. We define the value of parameter q_1^* such that rounding does not increase the objective value and can only lead to small increase in "loads" on local profiles (Lemma 1.4.16). In Section 1.4.8, we obtain a near-optimal schedule in which tiny jobs of each profile φ obey Smith's rule. First, by combining integral assignments over all profiles φ we find new sets of tiny operations for every single interval of the optimal schedule. Then, we apply the time-stretching technique to the optimal schedule and create some idle time inside intervals. Finally, we pack new sets of tiny operations by rescheduling operations inside single intervals. This procedure increase the objective value by at most a factor of $1+7\varepsilon$.

In contrast to the single machine case, here "packing" jobs in each single interval corresponds to the makespan version of the job shop scheduling problem which is known to be NP-hard [GJ79]. To copy with that, we follow ideas of the PTAS which was first presented in [JSOS99] and later modified in [FJM01]. We define the value of parameter q_2^* such that the PTAS can output a feasible schedule inside any single interval (Lemma 1.4.18).

After we have proved Smith's rule for tiny jobs, the rest of our method follows straightforward. Part (2) Compacting, Section 1.4.9, and part (3) Dynamic Programming, Section 1.4.10, are similar to the single machine case. In total, we prove the following main result:

Theorem 1.4.1. *There is a PTAS for $Jm|op \le \mu, r_j|\sum w_jC_j$ that computes for any fixed m, μ and $\varepsilon > 0$ accuracy, a $(1+\varepsilon)$-approximate schedule in $O(n\log n)$ time.*

1.4.1 Basic Structuring

To simplify notations we will use that $1/\varepsilon$ is integer (in particular $\varepsilon < 1/2^{m\mu}$). For an operation o_{ij}, we use S_{ij} and C_{ij} to denote the start and completion time of o_{ij}. For a job j, we use C_j and S_j to denote the completion and start time, and use

$\ell_j = \sum_{i=1}^{\mu} p_{ij}$ to denote the *length* of job j. For a job set X, we use $D(X) := \sum_{j \in X} \ell_j$ to denote the total length of X. Following previous notations, OPT denotes the objective value of the optimal schedule, I_x denotes interval $[R_x, R_{x+1})$ and $|I|_x = R_{x+1} - R_x$ denotes its size, where $R_x = (1+\varepsilon)^x$ for integer $x \in \mathbb{Z}$.

As in the single machine case, we first use geometric rounding and stretching. Following the same line of ideas we can prove:

Lemma 1.4.2. *With $1+\varepsilon$ loss and in $O(n)$ time, we can enforce all ℓ_j and r_j be integer powers of $1+\varepsilon$.*

Hence, all release dates r_j are of the form $R_x = (1+\varepsilon)^x$ for some integer x.

Lemma 1.4.3. *With $1+\varepsilon$ loss and in time $O(n)$, we can enforce all w_j/ℓ_j be distinct.*

Lemma 1.4.4. *With $1+\varepsilon$ loss, we can assume that all $S_{ij} \geq \varepsilon p_{ij}$ in a schedule.*

Lemma 1.4.5. *With $1+\varepsilon$ loss, we can assume that each operation crosses at most $s^* = \lceil \log_{1+\varepsilon}(1+\frac{1}{\varepsilon}) \rceil$ intervals in a schedule.*

1.4.2 Main and Negligible Operations

Consider a job j. Let $\ell_j = \sum_{i=1}^{\mu} p_{ij}$ be its length. Let operations o_{ij} ($i = 1, \ldots, \mu$) be indexed by $i_1, i_2, \ldots, i_\mu$ such that $p_{i_1 j} \geq p_{i_2 j} \geq \ldots \geq p_{i_\mu j}$. Then, if there exist some $k \in \{1, \ldots, \mu\}$ such that

$$\varepsilon^{4\mu} \cdot p_{i_k j} > \sum_{s=k+1}^{\mu} p_{i_s j}, \tag{1.9}$$

then we select the smallest value of k and define operations $o_{i_{k+1} j}, \ldots, o_{i_\mu j}$ be *negligible*, and operations $o_{i_1 j}, \ldots, o_{i_k j}$ be *main*. For an illustration see Figure 1.15.

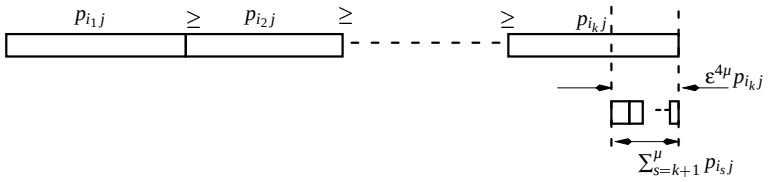


Figure 1.15: Main and negligible operations

Lemma 1.4.6. *Each main operation o_{ij} has processing time $p_{ij} \geq \varepsilon^{5\mu^2} \cdot \ell_j$.*

Proof. W.l.o.g. we rename the operations of job j such that processing times $p_{1j} \geq p_{2j} \geq \ldots \geq p_{\mu j}$. Let $\ell_j = \sum_{i=1}^{\mu} p_{ij}$ be the length of job j, and let $\rho = \varepsilon^{4\mu}$. Assume that $k = 1$. Then, the first operation is main and the other $\mu - 1$ operations are negligible. Let

$$\ell_j(1) \;=\; \ell_j \;-\; p_{1j}$$

Then, from (1.9) we have

$$\rho \cdot p_{1j} \;>\; \ell_j(1).$$

Thus,

$$p_{1j} \;>\; \ell_j(1)/\rho \;=\; (\ell_j \;-\; p_{1j})/\rho.$$

Finally, we have that

$$p_{1j}\left(1+\frac{1}{\rho}\right) \;>\; \ell_j \;\text{ and }\; p_{1j} > \left(\frac{\rho}{1+\rho}\right)\ell_j.$$

From $\varepsilon \leq 1/2$ and $\rho = \varepsilon^{4\mu}$ we conclude

$$\left(\frac{\varepsilon^{4\mu}}{1 \;+\; \varepsilon^{4\mu}}\right) \;\geq\; \frac{\varepsilon^{4\mu}}{2} \;\geq\; \varepsilon^{5\mu^2}.$$

Assume that $k \geq 2$. Then, the first $k \geq 2$ operations are main and the other $\mu - k$ operations are negligible. Let

$$\ell_j(t) \;:=\; \ell_j \;-\; \sum_{i=1}^{t} p_{ij}, \text{ for } t \;=\; 1,\ldots,k. \tag{1.10}$$

Accordingly, it holds that

$$\ell_j(1) \;=\; \ell_j \;-\; p_{1j} \;\text{ and }\; \ell_j(t) \;=\; \ell_j(t-1) \;-\; p_{tj}, \text{ for } t \;=\; 2,\ldots,k. \tag{1.11}$$

In (1.9) the value of k is smallest. Hence, it follows that

$$p_{kj} \;>\; \ell_j(k)\,/\,\rho \;\text{ and }\; p_{tj} \;\leq\; \ell_j(t)\,/\,\rho, \;\text{ for } t = 1,\ldots,k-1. \tag{1.12}$$

Since

$$\ell_j(1) \;=\; \ell_j \;-\; p_{1j} \;\text{ and }\; p_{1j} \;\leq\; \ell_j(1)/\rho$$

we have

$$\ell_j(1) \;\geq\; \ell_j \;-\; \ell_j(1)/\rho.$$

Thus,

$$\ell_j(1)\cdot\left(1 \;+\; \frac{1}{\rho}\right) \;\geq\; \ell_j \;\text{ and }\; \ell_j(1) \;\geq\; \left(\frac{\rho}{1+\rho}\right)\cdot\ell_j. \tag{1.13}$$

Similarly, from (1.11), (1.12) we have

$$\ell_j(t) \;=\; \ell_j(t-1) \;-\; p_{tj} \;\text{ and }\; p_{tj} \;\leq\; \ell_j(t)\,/\,\rho.$$

Thus,

$$\ell_j(t) \;\geq\; \left(\frac{\rho}{1\;+\;\rho}\right)\cdot\ell_j(t-1),\;\; \text{for } t = k-1,\ldots,2. \tag{1.14}$$

Summarizing, from

$$p_{kj} > \ell_j(k)\,/\,\rho = (\ell_j(k-1) - p_{kj})\,/\,\rho$$

it follows that

$$p_{kj}\cdot\left(1 \;+\; \frac{1}{\rho}\right) \;>\; \ell_j(k-1).$$

Then, using (1.14) and (1.13) we get

$$\begin{aligned}
p_{kj} \;&>\; \left(\frac{\rho}{1\;+\;\rho}\right)\cdot\ell_j(k-1)\\
&\geq\; \left(\frac{\rho}{1\;+\;\rho}\right)^2\cdot\ell_j(k-2)\\
&\;\;\vdots\\
&\geq\; \left(\frac{\rho}{1+\rho}\right)^{k-1}\ell_j(1)\\
&\geq\; \left(\frac{\rho}{1+\rho}\right)^{k}\cdot\ell_j.
\end{aligned}$$

Finally, $p_{tj} \geq p_{kj}$, $t = 1,\ldots,k$, and $k \leq \mu$ it implies that

$$p_{tj} \;\geq\; \left(\frac{\rho}{1+\rho}\right)^{k}\cdot\ell_j \;\geq\; \left(\frac{\rho}{1\;+\;\rho}\right)^{\mu}\cdot\ell_j,\;\; \text{for } t \;=\; 1,\ldots,k.$$

From $\varepsilon \leq 1/2$ and $\rho = \varepsilon^{4\mu}$ we conclude

$$\left(\frac{\varepsilon^{4\mu}}{1\;+\;\varepsilon^{4\mu}}\right)^{\mu} \geq \left(\frac{\varepsilon^{4\mu}}{2}\right)^{\mu} \;=\; \frac{\varepsilon^{4\mu^2}}{2^{\mu}} \geq \varepsilon^{4\mu^2+\mu} \geq \varepsilon^{5\mu^2}.$$

□

1.4.3 Main Structuring

By combining several techniques we eliminate all negligible operations and round all main operations.

Lemma 1.4.7. *With* $1+3\varepsilon$ *loss and in* $O(n)$ *time, for all jobs* j *we can enforce all operation processing times*

$$p_{ij} = \pi_{ij} \cdot \ell_j, \tag{1.15}$$

where

$$\pi_{ij} \in \left\{ z \cdot \varepsilon^{5\mu^2+2} \mid z = 0, 1 \ldots, \frac{1}{\varepsilon^{5\mu^2+2}} \right\}. \tag{1.16}$$

Proof. Assume that $p_{ij} = 0$ for all negligible operations o_{ij}. Consider an optimal schedule of objective value OPT. Each operation crosses at most s^* intervals (Lemma 1.4.5), and there are μ operations per job.

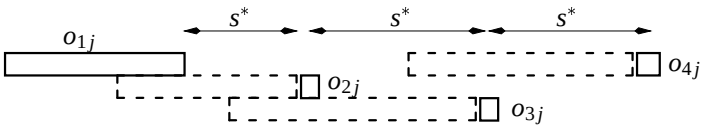


Figure 1.16: Main operation o_{1j} and negligible operations o_{2j}, o_{3j}, o_{4j}

First, we "shrink" the schedule as follows. Take a job j and consider its operations. We let all the main operations of j at their positions. For the negligible operations of j, we reschedule them as closer to the main operations as possible. For example, if there is a sequence of negligible operations which follow a main operation, then we simply reschedule these negligible operations in a first-fit manner. (See Figure 1.16.) Since processing times $p_{ij} = 0$ for all negligible operations o_{ij}, this does not increase the objective value. Repeating this procedure for all jobs j, we get a schedule in which each negligible operation o_{ij} is scheduled at most $s^*\mu$ intervals away from some main operation of job j.

Next, take an interval I_x. Let N_x be the set of negligible operations fall into I_x. For each operation $o_{ij} \in N_x$ we take the closest main operation of job j. Let M_x be the set of such main operations. Consider μs^* consecutive intervals from the left side and the right side of I_x. By using the above property, the main operations of

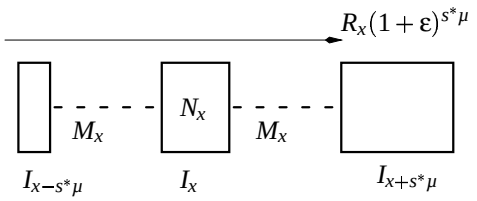


Figure 1.17: Interval I_x with sets N_x and M_x

M_x run (and complete) within these $2\mu s^*$ intervals. (See Figure 1.17.) There is at most

$$m \sum_{y=x-s^*\mu}^{x+s^*\mu} \leq 2mR_x(1+\varepsilon)^{\mu s^*}$$

$$\leq 2mR_x\left(1+\frac{1}{\varepsilon}\right)^{\mu} \quad (\text{here } s^* = \log_{1+\varepsilon}(1+1/\varepsilon)) \tag{1.17}$$

$$\leq 2m\left(\frac{2}{\varepsilon}\right)^{\mu}|I|_x/\varepsilon \leq |I|_x/\varepsilon^{2\mu} \quad (\text{using } 2^{\mu+m} \leq 1/\varepsilon)$$

time available on m machines. Hence, the total processing time of M_x is bounded by $|I|_x/\varepsilon^{2\mu}$. Recall (1.9), for a job j the total processing time of the negligible operations of j is not larger than $\varepsilon^{4\mu}$ times the size of a main operation of j. Thus, the total processing time of the negligible operations of N_x is at most $\varepsilon^{4\mu} \cdot (|I|_x/\varepsilon^{2\mu}) \leq \varepsilon^2|I|_x$. By using time-stretching we add $\varepsilon|I|_x$ time in each interval I_x and restore all the negligible operations of N_x. The objective value of the final schedule is at most $(1+\varepsilon)OPT$. Thus, with $1+\varepsilon$ loss and in $O(n)$ time, we can enforce $p_{ij} = 0$ for all negligible operations o_{ij}.

Now we can round main operations as follows. We first take an optimal schedule of objective value OPT. Then, we multiply all C_{ij} by $1+\varepsilon$ and increase S_{ij} to match (without changing operation processing times). The objective value of the final schedule is at most $(1+\varepsilon)OPT$, and there is at least εp_{ij} idle time before each operation o_{ij}. (See Lemma 1.3.4.)

For all main operations o_{ij} we have $p_{ij} \geq \varepsilon^{5\mu^2} \cdot \ell_j$ (Lemma 1.4.6). Then, we can use $\varepsilon p_{ij} \geq \varepsilon^2 \cdot (\varepsilon^{5\mu^2}) \cdot \ell_j$ idle time to round p_{ij} to the next ε^2 multiple value of $\varepsilon^{5\mu^2} \cdot \ell_j$. Thus, we get $p_{ij} := \pi_{ij} \cdot \ell_j$, where $\pi_{ij} \in \{z \cdot \varepsilon^{5\mu^2+2} \mid z = 1, \ldots, 1/\varepsilon^{5\mu^2+2}\}$. □

We can also move release dates.

Lemma 1.4.8. *With $1+\varepsilon$ loss and in $O(n)$ time, we can enforce r_j be at least $\varepsilon^{10\mu^2} \cdot \ell_j$ for all jobs j.*

Proof. We follow the same ideas as in the proof of Lemma 1.4.7. First, we take an optimal schedule of objective value OPT. Next, we multiply all C_{ij} by $1+\varepsilon$ and increase S_{ij} to match (without changing operation processing times). The objective value of the final schedule is at most $(1+\varepsilon)OPT$, and there is at least εp_{ij} idle time before each operation o_{ij}. (See Lemma 1.3.4.)

Take a job j. If the first operation o_{1j} is main, we increase r_j to enforce $r_j \geq \varepsilon^2(\varepsilon^{5\mu^2}\ell_j)$. If o_{1j} is negligible, then it is at most μs^* intervals away from some main operation of j. We increase r_j to enforce

$$r_j \geq \varepsilon^2(\varepsilon^{5\mu^2}\ell_j)/(1+\varepsilon)^{s^*\mu} = \varepsilon^{5\mu^2+2}\ell_j/(1+1/\varepsilon)^{\mu} \geq \varepsilon^{10\mu^2}\ell_j.$$

□

Crossing Jobs. Now we can prove that no job can cross too many intervals.

Lemma 1.4.9. *With $1+\varepsilon$ loss, each job crosses at most a constant number of intervals e^*.*

Proof. Take an optimal schedule of objective value OPT. Consider all jobs those main operations complete in an interval I_x. By Lemmas 1.4.5 and 1.4.7 we can bound the total length of these jobs by $h^*|I|_x$, where h^* is some constant. (In some sense, we follow the ideas in Section 1.3.10.) Thus, after a constant number of intervals, say e^*, this total length is only at most $\varepsilon^2|I|_{x+e^*}$.

For an interval I_b, consider all the non-completed jobs that start e^* intervals before. Their total length

$$\sum_{x<b-e^*} \varepsilon^2|I|_x = \varepsilon^2|I|_b \sum_{k\geq 1} \frac{1}{(1+\varepsilon)^k} \leq \varepsilon^2|I|_b/\varepsilon = \varepsilon|I|_b.$$

Then, we complete each of the jobs by scheduling its operations in a *first-fit* manner within μs^* intervals following I_b. (Since each operation crosses at most s intervals, for any job j in J one can always find a sequence of "gaps" corresponding to machine order $\tau_{1j},\tau_{2j},\ldots,\tau_{\mu j}$. See also Figure 1.16 on page 45 and the proof of Lemma 1.4.7.) Again, by using time-stretching the objective value of the final schedule is at most $(1+\varepsilon)OPT$. □

1.4.4 Profiles, Huge and Tiny Jobs, Local Profiles and Patterns

Here we introduce some definitions that will be used throughout next three sections. We first define job profiles. Two jobs of the same profile have the same set of required processors and their operations form the same set of multiples, but they can differ in length. Next, we define huge and tiny jobs, which depends on the value of parameter $q^* = q_1^* \cdot q_2^*$. The idea behind is to define the value of q_i^* ($i = 1, 2$) later in Sections 1.4.7,1.4.8, respectively. Finally, we define local profiles and patterns for tiny jobs. This allows us to formulate Smith's rule for tiny jobs in Section 1.4.5.

Profiles. Consider a job j. By Lemma 1.4.7, each operation o_{ij} ($i = 1, \ldots, \mu$) has processing time $\pi_{ij} \cdot \ell_j$ and requites machine τ_{ij}. Then, μ-tuples $\pi_j := (\pi_{ij})_{i=1}^{\mu}$ and $\tau_j := (\tau_{ij})_{i=1}^{\mu}$ are called the *execution* and *machine* profile of a job j, respectively.

We say that two jobs have the same profile $\varphi = (\pi, \tau)$, if they have the same execution profile π and machine profile τ. Notice that two jobs of profile φ can only differ in their length and release dates. Furthermore, as a consequence of Lemma 1.4.7 we can prove the following:

Lemma 1.4.10. *The number of distinct profiles is bounded by a constant* ν^*.

Huge and Tiny Jobs. We say that a job j is *huge* in an interval I_x if its length $\ell_j \geq \varepsilon^2 |I|_{x(j)}/q^*$, and *tiny* otherwise. The value of parameter $q^* = q_1^* \cdot q_2^* \gg 1$ is defined later in Lemmas 1.4.16 and 1.4.18, respectively. We will write H_x and T_x to denote sets of huge and tiny jobs released at R_x (H for huge and T for tiny).

As in the single machine case, we can use time-stretching to "clean up" a schedule. Similarly to Lemma 1.3.8, but regarding operations, we can prove the following:

Lemma 1.4.11. *With* $1+\varepsilon$ *loss, we can assume that no tiny operation crosses an interval in a schedule.*

Furthermore, similarly to Lemma 1.3.7, by using Lemmas 1.4.2 and 1.4.8 we can prove the following:

Lemma 1.4.12. *There is at most a constant number* $z^* = z^*(q_1^*, q_2^*)$ *of distinct sizes (*ℓ_j *powers of* $1+\varepsilon$*) in* H_x.

Local Profiles and Patterns. Take an optimal schedule. Consider a tiny job j. Let $x(j)$ be the index for which $R_{x(j)} = r_j$. Let $y(j)$ and $z(j)$ be whose indices for which $S_j \in I_{y(j)}$ and $C_j \in I_{z(j)}$. Then, job j runs in intervals $I_{y(j)}, \ldots, I_{z(j)}$.

By Lemma 1.4.11, the operations of tiny job j do not cross intervals. Hence, the set O_j of all operations $o_{1j}, \ldots, o_{\mu j}$ "splits" into a constant number of subsets, $O_j^{y(j)}, \ldots, O_j^{z(j)}$, where each subset $O_j^x \subseteq O_j$ consists of operations which "fall" into interval I_x, for $x = y(j), \ldots, z(j)$.

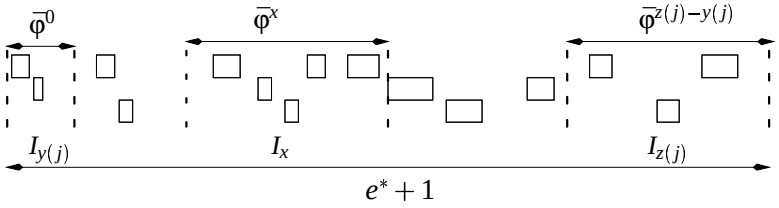


Figure 1.18: Local profiles of job j

Assume that tiny job j has some profile $\varphi = (\pi, \tau)$, where two μ-tuples $\pi = (\pi_i)_{i=1}^{\mu}$ and $\tau = (\tau_i)_{i=1}^{\mu}$. Then, we have that $\pi_i = \pi_{ij}$ and $\tau_i = \tau_{ij}$, for all operations $o_{ij} \in O_j$. Informally, we can say that the operations of set O_j "form" profile $\varphi = (\pi, \tau)$. If we restrict ourselves to the operations of set O_j^x, we can define a $2|O_j^x|$-tuple $\bar{\varphi}^x = (\bar{\pi}^x, \bar{\tau}^x)$ such that $\bar{\pi}_i^x = \pi_{ij}$ and $\bar{\tau}_i^x = \tau_{ij}$, for all operations $o_{ij} \in O_j^x$. In this case, we can also say that the operations of set O_j^x "form" the *local profile* $\bar{\varphi}^x$ in interval I_x, $x = y(j), \ldots, z(j)$.

In other words, tiny operations "locally" form "profiles" for jobs. Notice that every local profile is just a prototype of the profile which corresponds to a subset of the operations. However, the operations of two jobs with different profiles can form the same local profile in an interval.

We say that a tiny job j has *pattern* $f(j) = < \bar{\varphi}^0, \bar{\varphi}^1, \ldots, \bar{\varphi}^{|f(j)|} >$ in a schedule if tiny job j starts in $I_{y(j)}$ and completes in $I_{y(j)+|f(j)|}$, and in each interval $I_{y(j)+k}$ the operations of tiny job j form local profile $\bar{\varphi}^k$, for $k = 0, \ldots, |f(j)|$. For an illustration see Figure 1.18. Notice some local profiles $\bar{\varphi}^k$ can be empty, but the combination of all local profiles gives the profile of tiny job j.

By Lemmas 1.4.9 any tiny job crosses at most a constant number e^* of intervals. By Lemma 1.4.10 there is at most a constant number ν^* of profiles. Hence, we can prove the following:

Lemma 1.4.13. *There is at most a constant number $\bar{\nu}^*$ of distinct local profiles, and at most a constant number f^* of distinct patterns.*

1.4.5 Scheduling Tiny Jobs: Smith's Rule

We first need to introduce some notations. Consider an optimal schedule. Then, for a tiny job j we can define two indices $x(j) \leq y(j)$ and pattern $f(j)$ such that job j is released at $R_{x(j)}$, starts at $S_j \in I_{y(j)}$ having pattern $f(j)$ and completes at $C_j \in I_{y(j)+|f(j)|}$.

Smith's rule. Let k and j be two tiny jobs with $x(k) \leq x(j)$ (here $r_k \leq r_j$) and $\frac{w_j}{\ell_j} < \frac{w_k}{\ell_k}$ (see Lemma 1.4.3). We say that tiny jobs *obey* Smith's rule if $y(k) + |f(k)| \leq y(j) + |f(j)|$ for all such pairs of jobs j and k. In other words, if the two jobs are available in an interval, then job k of greater value w_k/ℓ_k completes not later than job j with respect to intervals $I_{y(k)}$, $I_{y(j)}$ and patterns $f(k)$, $f(j)$. For an illustration see Figure 1.19.

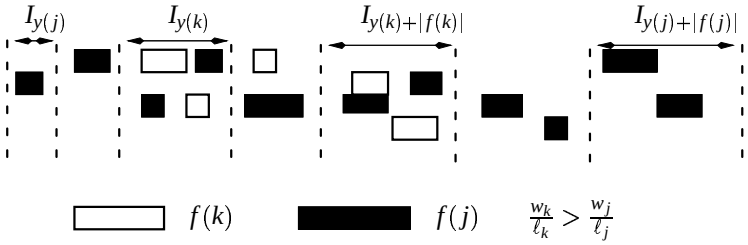


Figure 1.19: Smith's rule for jobs k and j

We are interested in the following result:

Lemma 1.4.14. *The value of parameter $q^* = q_1^* \cdot q_2^*$ can be defined such that with $1 + 7\varepsilon$ loss, for each profile φ we can assume that tiny jobs of profile φ obey Smith's rule in a schedule.*

Proof. The proof is given in Sections 1.4.6,1.4.7,1.4.8. In fact, we follow the same line of ideas as in the single machine case. As before, we first use a subroutine for "assigning" tiny jobs to intervals, and then as a subroutine for "packing" jobs in single intervals. From one side, assigning and packing here are also based on an LP formulation, a rounding procedure, and the time-stretching technique. From another side, we have to generalize both the LP formulation and rounding procedure for dealing with operations defined by profiles, patterns and local profiles, and we have to combine the time-stretching technique with a PTAS for the makespan version of the problem [JSOS99, FJM01]. We can briefly sketch the proof as follows.

We first take an optimal schedule of objective value OPT, and consider one profile φ. For each tiny job j of profile we find index $y(j)$ and pattern $f(j)$. By using $y(j)$ and $f(j)$, for each interval I_x and local profile $\bar{\varphi}$ we define sets $Y_x^{\bar{\varphi}}(\varphi)$ of tiny jobs of profile φ which operations form $\bar{\varphi}$ in I_x.

Next, by using $D(Y_x^{\bar{\varphi}}(\varphi))$, we formulate a linear program (named LP(φ)) which defines a fractional assignment of tiny jobs of φ to intervals and patterns. Then, by using Lemmas 1.4.13 and the rounding procedure from Appendix B on page 197, we round an optimal LP solution to an integral assignment.

By using this integral assignment, for each tiny job j we can define new index $y(j)$ and new pattern $f(j)$. Then, for each interval I_x and local profile $\bar{\varphi}$ we can define new sets $\widetilde{Y}_x^{\bar{\varphi}}(\varphi)$ of tiny jobs of profile φ which operations are assigned to $\bar{\varphi}$ in I_x.

From one side, the rounding procedure does not increase the objective value for tiny jobs. From another side, tiny jobs j of profile φ obey Smith's rule with respect to these new $y(j)$ and $f(j)$. Our main idea here is to define the value of parameter q_1^* in the definition of tiny jobs such that

$$D(\widetilde{Y}_x^{\bar{\varphi}}(\varphi)) \;\leq\; D(Y_x^{\bar{\varphi}}(\varphi)) \;+\; \varepsilon^2|I|_x/(2\bar{\nu}\nu q_2^*). \tag{1.18}$$

Indeed, we can combine integral assignments over all profiles φ. Thus, we can find new indices $y(j)$ and patterns $f(j)$ for all tiny jobs j. Our next idea is to modify the optimal schedule such that in a near-optimal schedule for each profile φ tiny jobs of φ obey Smith's rule with respect to these $y(j)$ and $f(j)$.

Consider interval I_x in the optimal schedule. Let $Y_x^{\bar{\varphi}}$ be the union of sets $Y_x^{\bar{\varphi}}(\varphi)$ over all profiles φ. Then, the operations of tiny jobs in $Y_x^{\bar{\varphi}}$ form local profile $\bar{\varphi}$ in I_x, and

$$D(Y_x^{\bar{\varphi}}) \;=\; \sum_{\varphi} D(Y_x^{\bar{\varphi}}(\varphi)). \tag{1.19}$$

Now let $\widetilde{Y}_x^{\bar{\varphi}}$ be the union of new sets $\widetilde{Y}_x^{\bar{\varphi}}(\varphi)$ over all profiles φ. Then, the operations of tiny jobs in $\widetilde{Y}_x^{\bar{\varphi}}$ also form local profile $\bar{\varphi}$ in I_x. Furthermore, by Lemmas 1.4.10 and (1.18), (1.19) we can bound

$$D(\widetilde{Y}_x^{\bar{\varphi}}) \;\leq\; D(Y_x^{\bar{\varphi}}) \;+\; \varepsilon^2|I|_x/(2\bar{\nu}q_2^*). \tag{1.20}$$

We simply take the optimal schedule and use time-stretching. We add $\varepsilon|I|_x$ idle time on machines in each interval I_x. Then, we replace the tiny operations of $Y_x^{\bar{\varphi}}$ by the tiny operations of $\widetilde{Y}_x^{\bar{\varphi}}$ inside each single interval I_{x+1}. This gives us a schedule

with the objective function value at most $(1+7\varepsilon)OPT$ in which for each profile φ tiny jobs of profile φ obey Smith's rule.

However, this last packing procedure is a form of the makespan version of the problem, which is known to be NP-hard. To copy with that, we use (1.20) and a PTAS for the makespan version of the problem [JSOS99, FJM01]. Our idea here is to define the value of parameter q_2^* in the definition of tiny jobs such that $Y_x^{\bar{\varphi}}$ can be replaced by $\widetilde{Y}_x^{\bar{\varphi}}$ inside interval I_{x+1}. □

1.4.6 Assigning to Intervals and Patterns: LP formulation

Consider an optimal schedule of objective value OPT. For each tiny job j we define two indices $x(j) \le y(j)$ and profile $f(j)$ such that job j is released at $R_{x(j)}$, starts at $S_j \in I_{y(j)}$ having pattern $f(j)$ and completes at $C_j \in I_{y(j)+|f(j)|}$.

Now we use these $y(j)$ and $f(j)$ in dealing with each interval I_x and local profile $\bar{\varphi}$. First, considering all tiny jobs, we define the set $Y_x^{\bar{\varphi}}$ of tiny jobs which operations form local profile $\bar{\varphi}$ in I_x. Next, considering tiny jobs of profile φ, we define the set $Y_x^{\bar{\varphi}}(\varphi)$ of tiny jobs of profile φ which operations form local profile $\bar{\varphi}$ in I_x. Clearly,

$$Y_x^{\bar{\varphi}} = \cup_{\varphi} Y_x^{\bar{\varphi}}(\varphi) \tag{1.21}$$

and

$$D(Y_x^{\bar{\varphi}}) \;=\; \sum_{\varphi} D(Y_x^{\bar{\varphi}}(\varphi)). \tag{1.22}$$

For each tiny job j of profile φ we define an *assignment* $\xi^{(j)} = (\xi_y^{(f,j)})$ such that

$$\sum_{f} \sum_{y \ge x(j)} \xi_y^{(f,j)} \;=\; 1, \tag{1.23}$$

where $\xi_y^{(f,j)} \in [0,1]$ is the (y,f)*th fraction* of job j which assigned to interval I_y and pattern f.

Let $f = (\bar{\varphi}^0, \bar{\varphi}^1, \ldots, \bar{\varphi}^{|f|})$. Then, $\xi_y^{(f,j)} > 0$ means that the fractional length $\xi_y^{(f,j)} \cdot \ell_j$ is assigned to each kth interval I_{y+k} on local profile $\bar{\varphi}^k$, for $k = 0, \ldots, |f|$. For an illustration see Figure 1.20 on the facing page.

Consider an interval I_x, local profile $\bar{\varphi}$ and pattern $f = (\bar{\varphi}^0, \bar{\varphi}^1, \ldots, \bar{\varphi}^{|f|})$. Then, we can define

$$d_x(f, \bar{\varphi}) \;=\; \sum_{\substack{k=0,\ldots,|f| \\ \bar{\varphi}^k = \bar{\varphi}}} \left(\sum_{j : x(j) \le y = x-k} \xi_y^{(f,j)} \cdot \ell_j \right). \tag{1.24}$$

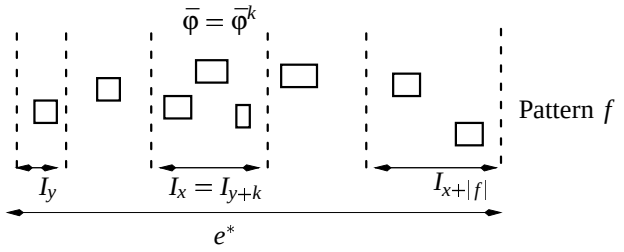


Figure 1.20: Pattern f and its load in an interval I_x

Informally, this value is equal to the "load" of pattern f on local profile $\bar{\varphi}$ in interval I_x. We take all kth local profiles $\bar{\varphi}^k$ ($k = 0, \ldots, |f|$) in pattern f that are equal to $\bar{\varphi}$, and then we sum up fractions $\xi_y^{(f,j)} \cdot \ell_j$ over all jobs j for which $y = x - k \geq x(j)$. In other words, we sum up all (y, f)th fractions which are assigned k intervals before I_x and have the kth local profile $\bar{\varphi}^k = \bar{\varphi}$.

The *fractional load* on local profile $\bar{\varphi}$ in interval I_x is defined as follows

$$\Delta_x^{\bar{\varphi}}(\varphi) \;=\; \sum_f d_x(f, \bar{\varphi}). \tag{1.25}$$

The *fractional weighted completion time* of a tiny job j of profile φ is defined as follows

$$w_j \sum_f \sum_{y \geq x(j)} \xi_y^{(f,j)} \cdot R_{y+|f|}. \tag{1.26}$$

Here, $R_{y+|f|}$ is the (y, f)th *fractional completion time* of job j. Then, the *fractional average weighted completion time* of tiny jobs of profile φ is defines as follows

$$F_\varphi(\xi) = \sum_{j \in T(\varphi)} w_j \sum_f \sum_{y \geq x(j)} \xi_y^{(f,j)} \cdot R_{y+|f|}. \tag{1.27}$$

We formulate the *fractional assignment problem* for the tiny jobs of profile φ as the following LP(φ):

$$\begin{aligned}
\textbf{Minimize} \quad & F_\varphi(\xi) \\
\textbf{s.t. (1)} \quad & \Delta_x^{\bar{\varphi}}(\varphi) \;\leq\; D_x^{\bar{\varphi}}(\varphi) = D(Y_x^{\bar{\varphi}}(\varphi)), \;\text{ for all } \bar{\varphi} \text{ and } x, \\
\textbf{(2)} \quad & \textstyle\sum_f \sum_{y \geq x(j)} \xi_y^{(f,j)} \;=\; 1, \;\text{ for all } j, \\
\textbf{(3)} \quad & \xi_y^{<f,j>} \;\geq\; 0, \;\text{ for all } f,\, y \geq x(j) \text{ and } j,
\end{aligned} \tag{1.28}$$

where the constraints have the following meaning: **(2)-(3)** each tiny job j is assigned completely to patterns f and intervals I_y, $y \geq x(j)$; **(1)** in each interval I_y the fractional load $\Delta_x^{\bar{\varphi}}(\varphi)$ on each local profile $\bar{\varphi}$ is at most $D_x^{\bar{\varphi}}(\varphi)$.

Informally, in the optimal schedule we allow all tiny jobs of profile φ be fractionally assigned to distinct intervals and patterns. However, preserving all "loads" $D_x^{\bar{\varphi}}(\varphi) = D(Y_x^{\bar{\varphi}}(\varphi))$ on local profiles $\bar{\varphi}$ in every single interval I_x, we keep the schedule of huge jobs without changes.

We can bound the fractional average weighted completion time as follows:

Lemma 1.4.15. *For any optimal solution ξ of the LP(φ) it holds that*

$$F_\varphi(\xi) \leq OPT'(\varphi), \tag{1.29}$$

where $OPT'(\varphi)$ is the average weighted completion time of the tiny jobs of profile φ in the optimal schedule. Accordingly, it also holds that

$$\sum_\varphi F_\varphi(\xi) \leq OPT', \tag{1.30}$$

where OPT' is the average weighted completion time of all tiny jobs in the optimal schedule.

Proof. Consider all tiny jobs j of profile φ in an optimal schedule. Let $y(j)$, $x(j)$ and $f(j)$ be such that job j is released at $R_{x(j)}$, starts at $S_j \in I_{y(j)}$ having pattern $f(j)$ and completes at $C_j \in I_{y(j)+|f(j)|}$.

Since $y(j) \geq x(j)$, for a solution ξ we can define $\xi_y^{(f,j)} = 1$ if $y = y(j)$, $f = f(j)$, and $\xi_y^{(f,j)} = 0$ otherwise. Then, ξ is feasible. Furthermore, since $C_j \in I_{y(j)+|f(j)|}$ we have

$$R_{y(j)+|f(j)|} \leq C_j$$

and

$$F_\varphi(\xi) = \sum_j w_j R_{y+|f(j)|} \leq \sum_j C_j = OPT'(\varphi).$$

Finally, from

$$OPT' = \sum_\varphi OPT'(\varphi)$$

we also have

$$\sum_\varphi F_\varphi(\xi) \leq OPT'.$$

□

1.4.7 Assigning to Intervals and Patterns: LP Rounding

For each profile φ, let ξ be an optimal solution for the LP(φ). Assume that ξ is integral. Then, for each tiny job j of profile φ we can find exactly one index $y(j)$ and pattern $f(j)$ such that $\xi^{f(j)}_{y(j)j} = 1$ and $y(j) \geq x(j)$. In other words, an integral LP solution can be treated as an integral assignment of tiny jobs to intervals and patterns.

Here we prove the following result:

Lemma 1.4.16. *Define the value of $q_1^* := 24\nu^*(\overline{\nu}^* e^*)^2$. Then, for each profile φ, an optimal solution ξ of the LP(φ) can be rounded into an integral assignment of tiny jobs of profile φ to intervals and patterns such that*

(1) *$F_\varphi(\xi)$ does not increase;*

(2) *the value of each $\Delta_x^{\overline{\varphi}}(\varphi)$ increases by at most $\varepsilon^2|I|_x/(2\overline{\nu}^*\nu^* q_2^*)$;*

(3) *tiny jobs j of profile φ obey the modified Smith's rule with respect to intervals $I_{y(j)}$ and patterns $f(j)$ given by rounded ξ.*

Proof. Let ξ be an optimal LP(φ) solution. Let j and k be two jobs of profile φ with $x(k) \leq x(j)$ and $\frac{w_j}{\ell_j} < \frac{w_k}{\ell_k}$ ($w_j\ell_k < w_k\ell_j$). Assume that there exist two fractions $\xi^{f(j)}_{jy(j)} > 0$, $\xi^{f(k)}_{ky(k)} > 0$ with $y(j) + |f(j)| < y(k) + |f(k)|$. Informally, this means that jobs j and k do not obey Smith's rule.

Then, there exist values t_j and t_k such that

$$0 < t_j \leq \xi^{f(j)}_{jy(j)} \text{ and } 0 < t_k \leq \xi^{f(k)}_{ky(k)}, \tag{1.31}$$

and

$$t_j\ell_j = t_k\ell_k. \tag{1.32}$$

We define a new solution ζ as follows. First we exchange the $y(j)$th and $y(k)$th fractions of jobs j and k:

$$\begin{aligned} \zeta^{f(j)}_{jy(j)} &= \xi^{f(j)}_{jy(j)} - t_j \quad & \zeta^{f(k)}_{jy(k)} &= \xi^{f(k)}_{jy(k)} + t_j \\ \zeta^{f(k)}_{ky(k)} &= \xi^{f(k)}_{ky(k)} - t_k \quad & \zeta^{f(j)}_{ky(j)} &= \xi^{f(j)}_{ky(j)} + t_k. \end{aligned} \tag{1.33}$$

After this, we define the rest of ζ as in ξ.

Due to (1.31), (1.32) and (1.33) solution ζ is feasible. Furthermore,

$$\begin{aligned} F(\zeta) - F(\xi) &= R_{y(j)+|f(j)|}(w_k t_k - w_j t_j) + R_{y(k)+|f(k)|}(w_j t_j - w_k t_k) \\ &= (R_{y(j)+|f(j)|} - R_{y(k)+|f(k)|})(w_k t_k - t_j w_j). \end{aligned} \tag{1.34}$$

From $y(j)+|f(j)| < y(k)+|f(k)|$ we have $R_{y(j)+|f(j)|} - R_{y(k)+|f(k)|} < 0$. From $w_j \ell_k < w_k \ell_j$ and (1.32) we have $(w_k t_k - w_j t_j) > 0$. Hence, from (1.34) we conclude

$$F(\zeta) < F(\xi).$$

This is a contradiction to the optimality of ξ.

In the following we round ξ into an integral solution. We will use the above property of ξ. We will not increase the value of $F(\xi)$, but the values of $\Delta_x^{\bar{\varphi}}(\varphi)$.

One can see that for each interval I_y variables $\xi_y = (\xi_y^{(f,j)})$ appear in at most $\bar{\nu}^*(e^*+1)$ constraints **(1)** of the LP corresponding to intervals $I_y, I_{y+1}, \ldots, I_{y+e^*}$. W.l.o.g. we can reassign the values of ξ_y such that there are at most $\bar{\nu}^*(e^*+1)$ jobs j with fractions $\xi_y^{(f,j)} > 0$ for at least two different patterns f, see Appendix B on page 197. Then, for each such job j we can select one pattern having the smallest length, and then round the values of $\xi^{(j)}$. We use this property as follows. We start from the first interval of the schedule and go further up to the end. For each interval I_y, we perform the rounding procedure for the values of ξ_y such that for each job j there is at most one pattern f for which $\xi_y^{(f,j)} > 0$. This completes the first step of rounding.

One can see that that the value of $F(\xi)$ does not increase. Furthermore, the above stated property of ξ (Smith's rule) is still valid. However, rounding the values of ξ violates constraints **(2)** of the LP(φ). Recall that the length of any pattern f is bounded by e^*, see Figure 1.20 on page 53. Consider one interval I_x and the values of $\Delta_x^{\bar{\varphi}}(\varphi)$ in **(2)** of the LP(φ). By the above rounding procedure, only the jobs which we have rounded in e^*+1 foregoing intervals I_y, $y = x, \ldots, x-e^*$, increase the values of $\Delta_x^{\bar{\varphi}}(\varphi)$. There are at most $(e^*+1)(\bar{\nu}^*(e^*+1))$ such jobs. Each of these jobs is tiny and released before I_x, i.e. its length is at most $\varepsilon^2 |I|_x / q^*$. Thus, the total increase in the value of each $\Delta_x^{\bar{\varphi}}(\varphi)$ is at most

$$(e^*+1)(\bar{\nu}^*(e^*+1)\varepsilon^2 |I|_x / q^*). \tag{1.35}$$

At the second step, we enforce each tiny job to be completed within e^*+1 consecutive intervals. First, one can see the following fact. For each interval I_y there

is at most one job k with $\xi_y^{<f_y,k>} > 0$ and $\zeta_x^{<f_x,k>} > 0$, where $x + |f_x| \geq y + e^*$. If there are two such jobs, say jobs j and k such that $w_j/\ell_j < w_k/\ell_k$, then we can replace a fraction of job j by a fraction of job k in interval I_y. This will be a contradiction to the above stated property of ξ (Smith's rule). We use this fact as follows. We start from the first interval of the schedule and go further up to the end. In each interval I_y we find such a job k (if it exists) and round the values of $\xi^{(k)}$.

Similar to the first step, $F(\xi)$ does not increase and the above stated property of ξ (Smith's rule) is still valid. The total increase in the value of each $\Delta_x^{\bar{\varphi}}(\varphi)$ is at most

$$(e^* + 1)\varepsilon^2 |I|_x / q^*. \tag{1.36}$$

At the third step, we round ξ into an integral solution. First, let us observe the following fact. For an interval I_y, let X_y be the set of all jobs j that are not assigned to previous intervals and for which $\xi_y^{(f,j)} > 0$, i.e. the set of tiny jobs that are fractionally assigned for the first time. Then, by the second rounding step, the jobs of X_y complete within $e^* + 1$ intervals $I_y, I_{y+1}, \ldots, I_{y+e^*}$. W.l.o.g. we can reassign the values of $\xi^{(j)}$ for each job $j \in X_y$ such that there are at most $\bar{\nu}^*(e^* + 1)$ jobs in X_y with non-unique assignment to the intervals $I_y, I_{y+1}, \ldots, I_{y+e^*}$ and patterns f, see Appendix B on page 197. We use this fact as follows. We start from the first interval of the schedule and go further up to the end. In each interval I_y we round the values of $\xi^{(j)}$ for all jobs $j \in X_y$.

As before, $F(\xi)$ does not increase and the above stated property of ξ (Smith's rule) is still valid. The total increase in the value of each $\Delta_x^{\bar{\varphi}}(\varphi)$ is at most

$$(e^* + 1)(\bar{\nu}^*(e^* + 1)\varepsilon^2 |I|_x / q^*). \tag{1.37}$$

Summing up (1.35), (1.36) and (1.37), after three steps of rounding the value of each $\Delta_x^{\bar{\varphi}}(\varphi)$ increases by at most

$$3(e^* + 1)\bar{\nu}^*\varepsilon^2 |I|_x / q^* \leq 3(2e^*)^2 \bar{\nu}^*\varepsilon^2 |I|_x / q^*. \tag{1.38}$$

In $q^* = q_1^* \cdot q_2^*$ we define

$$q_1^* := 24\nu^*(\bar{\nu}^* e^*)^2.$$

Hence, we can bound (1.38) by

$$\varepsilon^2 |I|_x / (2\bar{\nu}^*\nu^* q_2^*).$$

Now ξ is integral solution of LP(φ). Then, for each tiny job j of profile φ we can find $y(j)$ and $f(j)$ such that $\xi_{y(j)}^{f(j)j} = 1$ and $y(j) \geq x(j)$. Since the above stated property of ξ remains valid throughout the rounding procedure, tiny jobs j of profile φ obey Smith's rule with respect to these $y(j)$ and $f(j)$ given by ξ. □

Now we combine integral assignments ξ for all profiles φ. For each tiny job j we define index $y(j) \geq x(j)$ and pattern $f(j)$, respectively. Then, by using $y(j)$ and $f(j)$, for each interval I_x and local profile $\bar{\varphi}$, we define sets $\widetilde{Y}_x^{\bar{\varphi}}$ of tiny jobs that are assigned to $\bar{\varphi}$ in I_x. Then, we can prove the following:

Lemma 1.4.17. *For all tiny jobs j it holds that $y(j) \geq x(j)$ and*

$$\sum w_j R_{y(j)+|f(j)|} \;\leq\; OPT'. \tag{1.39}$$

Furthermore, for each interval I_x and local profile loc it holds

$$D(\widetilde{Y}_x^{\bar{\varphi}}) \;\leq\; D(Y_x^{\bar{\varphi}}) \;+\; \varepsilon^2 |I|_x/(2\bar{\nu}q_2^*). \tag{1.40}$$

Proof. By Lemmas 1.4.15 and 1.4.16, for all tiny jobs it holds that

$$\sum w_j R_{y(j)+|f(j)|} \;=\; \sum_{\varphi} F_{\varphi}(\xi) \;\leq\; OPT'.$$

To define sets $\widetilde{Y}_x^{\bar{\varphi}}$, we have used rounded solutions ξ for the LP(φ) over all profiles φ. Hence, it holds that

$$\sum_{\varphi} \Delta_x^{\bar{\varphi}}(\varphi) \;=\; D(\widetilde{Y}_x^{\bar{\varphi}}).$$

Then, by the LP(φ) formulation and Lemma 1.4.16 it holds that

$$y(j) \geq x(j)$$

and

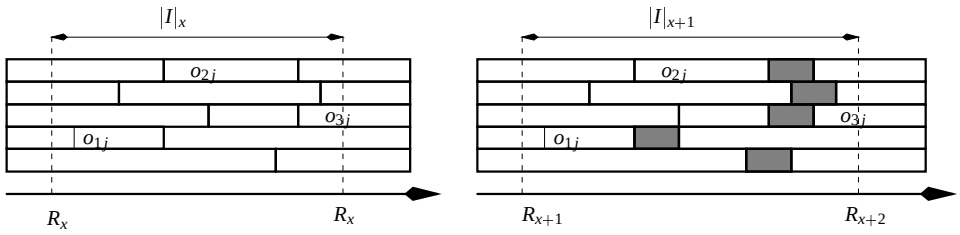
$$\begin{aligned} \Delta_x^{\bar{\varphi}}(\varphi) \;&\leq\; D^{\bar{\varphi}}(\varphi) \;+\; \varepsilon^2 |I|_x/(2\bar{\nu}^*\nu^* q_2^*) \\ &=\; D(Y_x^{\bar{\varphi}}(\varphi)) \;+\; \varepsilon^2 |I|_x/(2\bar{\nu}^*\nu^* q_2^*). \end{aligned}$$

By Lemma 1.4.10 the number of distinct profiles is bounded by ν^*. Hence, combining with (1.21) and (1.22) we have (1.40). □

1.4.8 Packing in Single Intervals: A PTAS for Makespan

Here, by using the results of Lemmas 1.4.16 and 1.4.17, we complete the proof of Lemma 1.4.14.

First, we take an optimal schedule of value OPT. For an illustration see Figure 1.21. Consider an interval I_x. Each operation that runs in I_x can be either tiny or huge, and each huge operation can be either crossing or non-crossing, see Figure 1.21 a). There are at most $2m$ huge crossing operations, and non-crossing operations form local profiles $\bar{\varphi}$ with respect to jobs. Then, we can define the set K_x of huge crossing jobs, and sets $N_x^{\bar{\varphi}}$ and $Y_x^{\bar{\varphi}}$ of non-crossing huge jobs and tiny jobs, respectively.



a) OPT b) $(1+\varepsilon)OPT$

Figure 1.21: Operations in interval I_x and I_{x+1}

Now we apply time-stretching to the optimal schedule. We add $\varepsilon|I|_x$ idle time on machines in each interval I_x, see Figure 1.21 b). This increases the objective function value by at most a factor of $1+\varepsilon$. Furthermore, the operations of set K_x and sets $N_x^{\bar{\varphi}}$, $Y_x^{\bar{\varphi}}$ run in interval I_{x+1}.

Next, we use the results of Lemmas 1.4.16 and 1.4.17. For each tiny job j we find new $y(j)$ and $f(j)$, and for each interval I_x we find new sets $\widetilde{Y}_x^{\bar{\varphi}}$ of tiny jobs. We simply reschedule all tiny jobs in the schedule. Inside each interval I_{x+1} we replace the tiny operations of sets $Y_x^{\bar{\varphi}}$ by the tiny operations of sets $\widetilde{Y}_x^{\bar{\varphi}}$.

Then, each tiny job j completes at $C_j \in I_{y(j)+f(j)+1}$. By Lemmas 1.4.16 and 1.4.17 we have that

$$\sum_j C_j \;\leq\; \sum_j (1+\varepsilon)\, R_{y(j)+f(j)+1} \;\leq\; (1+\varepsilon)^2 OPT'. \tag{1.41}$$

In order to get a feasible schedule, we reschedule the operations of set K_x and sets $N_x^{\bar{\varphi}}$, $\widetilde{Y}_x^{\bar{\varphi}}$ inside each single interval I_{x+1}. This also increases the objective function

value by at most a factor of $1+\varepsilon$, and the objective function value of the final schedule is at most

$$(1+\varepsilon)^3 OPT \;\leq\; (1+7\varepsilon)OPT.$$

To complete the proof of Lemma 1.4.14, in the rest of this section we prove the following result:

Lemma 1.4.18 ([JSOS99, FJM01]). *Define the value of* $q_2^* = (4\mu m+1)^{\left\lceil \frac{2me^*}{\varepsilon^7} \right\rceil}$. *Then, the operations of set* K_x *and sets* $N_x^{\bar{\varphi}}, \widetilde{Y}_x^{\bar{\varphi}}$ *can be scheduled inside each single interval* I_{x+1}.

General Problem. In order to prove the above stated lemma we need to solve the following problem. We are given job sets K_x, $N_x^{\bar{\varphi}}$, $Y_x^{\bar{\varphi}}$, $\widetilde{Y}_x^{\bar{\varphi}}$ (over all $\bar{\varphi}$) and their corresponding operations. It is known:

(i) there exists a schedule for the operations of set K_x and sets $N_x^{\bar{\varphi}}$, $Y_x^{\bar{\varphi}}$ inside interval I_x;

(ii) the length of sets $Y_x^{\bar{\varphi}}$ and $\widetilde{Y}_x^{\bar{\varphi}}$ differs by at most $\varepsilon^2|I|_x/2\bar{\nu}^* q_2^*$;

(iii) for each job j in $Y_x^{\bar{\varphi}}$ or $\widetilde{Y}_x^{\bar{\varphi}}$ it holds $\ell_j \leq \varepsilon^2|I|_x/q_2^*$.

Can we define the value of parameter q_2^* such that there exists a schedule for the operations of set K_x and sets $N_x^{\bar{\varphi}}, \widetilde{Y}_x^{\bar{\varphi}}$ inside interval I_{x+1}?

Let us first consider this problem informally. On an upper level, we can reformulate it as follows: We are a set of jobs J_x. There is a schedule for J_x of length $|I|_x$. Can we find a schedule for J_x of length $|I|_{x+1} = (1+\varepsilon)|I|_x$?

Indeed, the answer is "yes". We can simply use a PTAS for the makespan version of the problem [JSOS99, FJM01]. We can take $\delta = \varepsilon/2$ and run the PTAS on J_x with δ. The output schedule has length at most

$$(1+\delta)OPT(J_x) \;<\; (1+\varepsilon)|I|_x.$$

Now assume that we have a new set Y_x of jobs. Can we find a schedule for $J_x \cup Y_x$ of length $|I|_{x+1} = (1+\varepsilon)|I|_x$? Intuitively, the answer is "yes" if we can guarantee that all jobs in Y_x are small enough with respect to length $|I|_x$.

For example, assume that the total length $D(Y_x)$ is at most $\varepsilon^2|I|_x/2$. Then, we can take $\delta := \varepsilon^2/2$, run the PTAS on J_x with δ, and then add the jobs of Y_x at the end of the output schedule. The final schedule has length at most

$$(1+\delta)OPT(J_x) + \varepsilon^2|I|/2 \;<\; (1+\varepsilon)|I|_x = |I|_{x+1}.$$

In fact, our problem is quite similar. We have our auxiliary parameter q_2. Due to (i), there exists a schedule for the operations of sets K_x, $N_x^{\bar{\varphi}}$, $Y_x^{\bar{\varphi}}$ of length $|I|_x$. Due to (ii) and (iii), if q_2 tends to the infinity, the difference between sets $\widetilde{Y}_x^{\bar{\varphi}}$ and $Y_x^{\bar{\varphi}}$ slowly disappears. The idea behind our approach is to define the value of q_2 such that given the operations of set K_x and sets $N_x^{\bar{\varphi}}$, $\widetilde{Y}_x^{\bar{\varphi}}$ the PTAS with $\delta := \varepsilon^2/2$ outputs a schedule which has length at most

$$(1 + \delta)|I|_x + \varepsilon^2|I|/2 < (1+\varepsilon)|I|_x = |I|_{x+1}.$$

Clearly, this provides a positive answer to the general problem.

Simplified Problem. Formally, in a local profile $\bar{\varphi} = (\bar{\pi}, \bar{\tau})$ both tuples $\bar{\pi}$ and $\bar{\tau}$ can correspond to less than μ operations. This information is very important when we deal with a schedule which spans several intervals. Here, we only deal with a schedule inside single interval.

To simplify further presentation, w.l.o.g. we introduce some dummy operations with zero processing times. We replace both $\bar{\tau}$ and $\bar{\pi}$ by μ-tuples $\pi = (\pi_i)_{i=1}^{\mu}$ and $\tau = (\tau_i)_{i=1}^{\mu}$. However, not abuse the notations too much, we will write $\bar{\varphi} = (\pi, \tau)$.

To simplify the description, we first consider the case when the set of crossing huge jobs $K_x = \emptyset$. We also define

$$J_x^{\bar{\varphi}} = N_x^{\bar{\varphi}} \cup Y_x^{\bar{\varphi}} \text{ and } J_x = \cup_{\bar{\varphi}} J_x^{\bar{\varphi}}.$$

Then, all jobs j in set $J_x^{\bar{\varphi}}$ have the same local profile $\bar{\varphi} = (\pi, \tau)$, but differ in length ℓ_j. Formally, each job j in $J_x^{\bar{\varphi}}$ consists of operations $o_{1j}, \ldots, o_{\mu j}$, where each operation o_{ij} has processing time $\pi_i \cdot \ell_j$ and requires machine τ_i, for $i = 1, \ldots, \mu$.

Due to (i), there exists a schedule for the operations of sets $J_x^{\bar{\varphi}}$ which length is at most $|I|_x$. By Lemma 1.4.9 any job crosses at most e^* intervals. Thus, the total length of the jobs in J_x is at most the total time available within e^* intervals following I_x. If $m \geq 2$ and $e^* \geq 2$ we can bound

$$D(J_x) \leq m(e^*|I|_{x+e^*}) \leq me^*(1 + \varepsilon)^{e^*}|I|_x \leq me^*2^{e^*}|I|_x \leq 2^{me^*}|I|_x. \quad (1.42)$$

In the following we will use the known PTAS for the makespan version of the problem [JSOS99, FJM01]. First, we deduce a schedule for the operations of sets $J_x^{\bar{\varphi}}$. By defining the value of parameter q_2^* we ensure that the schedule length is bounded by $|I|_{x+1} = (1+\varepsilon)|I|_x$. Then, we show that this bound remains valid even if we replace the jobs of $Y_x^{\bar{\varphi}}$ by the jobs of $\widetilde{Y}_x^{\bar{\varphi}}$ in $J_x^{\bar{\varphi}}$. Finally, for the general case when the set of crossing huge jobs $K_x \neq \emptyset$, we change the value of q_2^*.

Long and Short Jobs. We take all jobs in J_x and number them by $0,1,2,\ldots,n'$ in order of non-increasing lengths

$$\ell_0 \;\geq\; \ell_1 \;\geq\; \ell_2 \;\geq\; \ldots \;\geq\; \ell_{n'}. \tag{1.43}$$

Here, $n'+1$ is the total number of jobs in J_x.

Next, we introduce an integer k_x^* and a constant $q_2^* \geq 0$. We will specify their exact values later. In addition, we introduce sets U_x, L_x and S_x as follows. We first define

$$U_x = \{j | j \in J_x \text{ and } \ell_j \geq \varepsilon^2 |I|_x / q_2^*\}. \tag{1.44}$$

Informally, we put all jobs j from set J_x with length $\ell_j \geq \varepsilon^2 |I|_x / q_2^*$ into set U_x. Then, with respect to the number of jobs in U_x and the value of k_x^* we consider two cases:

(1) If $|U_x| \leq k_x^*$ then we define $L_x := U_x$ and $S_x := J_x \setminus U_x$;

(2) If $|U_x| > k_x^*$ then we put the first k_x^* jobs $0,1,\ldots,k_x^*-1$ into set L_x, and other jobs $k_x^*, k_x^*+1,\ldots,n'$ into set S_x.

For simplicity, we call the jobs in L_x *long* and the jobs in S_x *short*. Finally, we define

$$S_x^{\bar{\varphi}} \;=\; S_x \cap J_x^{\bar{\varphi}}.$$

Notice that in both cases (1) and (2) it holds that

$$L_x \subseteq U_x \;\text{ and }\; S_x \;=\; J_x \setminus L_x. \tag{1.45}$$

Furthermore, the number of long jobs

$$|L_x| \;:=\; \min\{\, k_x^*, |U_x| \,\}, \tag{1.46}$$

i.e. either $U_x = L_x$ or the first k_x^* longest jobs from U_x are in L_x.

Snapshots and Relative Schedules. We say that two operations o_{ij} and $o_{i'j'}$ are *compatible* if $\tau_{ij} \neq \tau_{i'j'}$. Then, a *snapshot* is a set of compatible operations (it can be empty). In the following we consider the operations of long jobs in L_x. A *relative schedule* of L_x is a sequence $M(1),\ldots,M(g)$ of g snapshots such that for each long job $j \in L_x$ holds

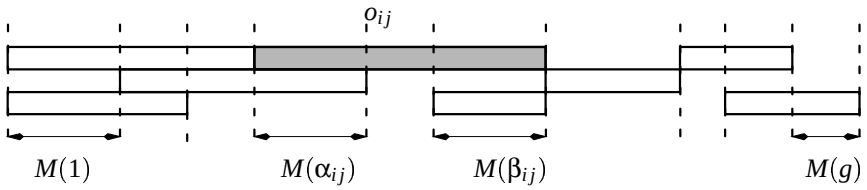


Figure 1.22: A relative schedule and an operation o_{ij}

- each operation o_{ij} $(i = 1, \ldots, \mu)$ occurs in a sequence of consecutive snapshots $M(\alpha_{ij}), \ldots, M(\beta_{ij})$, $1 \leq \alpha_{ij} \leq \beta_{ij} \leq g$, where $M(\alpha_{ij})$ and $M(\beta_{ij})$ are the first and last snapshots containing o_{ij},
- $\beta_{ij} < \alpha_{i+1,j}$ for two consecutive operations o_{ij} and $o_{i+1,j}$, and
- any two consecutive snapshots are different.

Less formally, a relative schedule corresponds to an execution order of the long operations of L_x. One can associate a relative schedule to each non-preemptive schedule of L_x by looking at every time in the schedule when a long operation starts or ends and creating a snapshot right after that time. For an illustration see Figure 1.22. Notice that for each long job there are μ operations. Hence, the number of snapshots

$$g \;\leq\; 2\mu|L_x|. \tag{1.47}$$

Free machines and Assignments. Assume that we are given a relative schedule $M(1), \ldots, M(g)$. Our next goal is to schedule the jobs in $S_x \cup L_x$ such that (a) all short operations of S_x are completed; (b) each long operation of L_x runs as it is defined in snapshots $M(1), \ldots, M(g)$.

For $s = 1, \ldots, g$ we define the set

$$F(s) \;:=\; M \setminus \left(\bigcup_{o_{ij} \in M(s),\, j \in L_x} \{\tau_{ij}\} \right) \tag{1.48}$$

of *free machines* in snapshot $M(s)$. Clearly, the short operations of S_x can only be processed on $F(s)$ in $M(s)$.

Take a local profile $\bar{\varphi} = (\pi, \tau)$, where $\pi = (\pi_i)_{i=1}^{\mu}$ and $\tau = (\tau_i)_{i=1}^{\mu}$. Consider sort jobs j in $S_x^{\bar{\varphi}}$. All ith operations o_{ij} require the same machine τ_i, for $i = 1, \ldots, \mu$.

Then, we define the set

$$A_{\bar{\varphi}} := \{< s_1, \ldots, s_\mu > \mid 1 \le s_1 \le \ldots \le s_\mu \le g \text{ and } \tau_i \in F(s_i), \text{ for } i = 1, \ldots, \mu\} \tag{1.49}$$

of possible *assignments* of operations o_{ij} to snapshots $M(s_i)$ in which machines τ_i are free, for $i = 1, \ldots, \mu$. Accordingly, for $s = 1, \ldots, g$, $\tau \in M = \{1, \ldots, m\}$ and $\vec{a} =< s_1, s_2, \ldots, s_\mu > \in A_{\bar{\varphi}}$ we define the set

$$B_{\bar{\varphi}}(s, \tau, \vec{a}) := \{i \mid s_i = s,\ \tau_i = \tau \text{ and } i = 1, \ldots, \mu\} \tag{1.50}$$

of all indices i which correspond to ith operations assigned by $\vec{a}$ to machine τ in snapshot $M(s)$.

Fractions and Loads. Take a local profile $\bar{\varphi} = (\pi, \tau)$, where $\pi = (\pi_i)_{i=1}^{\mu}$ and $\tau = (\tau_i)_{i=1}^{\mu}$. Consider set $S_x^{\bar{\varphi}}$ of short jobs. Let $D(S_x^{\bar{\varphi}}) = \sum_{j \in S_x^{\bar{\varphi}}} \ell_j$ be the total length of the jobs in $S_x^{\bar{\varphi}}$. Then, all ith operations of these jobs require the same machine τ_i and their total processing time is equal to $\pi_i \cdot D(S_x^{\bar{\varphi}})$, for $i = 1, \ldots, \mu$. We simply treat all jobs in $S_x^{\bar{\varphi}}$ as one job of local profile $\bar{\varphi}$ which has length $D(S_x^{\bar{\varphi}})$. We define $\xi^{\bar{\varphi}} = (\xi_{\vec{a}})_{\vec{a} \in A_{\bar{\varphi}}}$ such that

$$\sum_{\vec{a} \in A_{\bar{\varphi}}} \xi_{\vec{a}} = 1, \tag{1.51}$$

where $\xi_{\vec{a}} \in [0, 1]$ is the fraction of the jobs of $S_x^{\bar{\varphi}}$ assigned by $\vec{a} \in A_{\bar{\varphi}}$. Accordingly, for $s = 1, \ldots, g$ and $\tau \in \{1, \ldots, m\}$

$$L_{(s,\tau,\bar{\varphi})}(\xi) := \sum_{\vec{a} \in A_{\bar{\varphi}}} \left(\sum_{i \in B_{\bar{\varphi}}(s,\tau,\vec{a})} \xi_{\vec{a}} \cdot \pi_i \cdot D(S_x^{\bar{\varphi}}) \right) \tag{1.52}$$

is the *load* of $S_x^{\bar{\varphi}}$ on machine τ in snapshot $M(s)$.

Regarding all short jobs in $S_x = \cup_{\bar{\varphi}} S_x^{\bar{\varphi}}$,

$$L_{(s,\tau)}(\xi) = \sum_{\bar{\varphi}} L_{(s,\tau,\bar{\varphi})}(\xi) \tag{1.53}$$

is the *load* of S_x on machine τ in snapshot $M(s)$.

LP Formulation. Let $M(1),\dots,M(g)$ be a relative schedule of L_x. Then, we formulate the problem of scheduling the long jobs in L_x and the short jobs in S_x as the following LP:

$$\begin{aligned}
\textbf{Minimize} \quad & L(\xi,t) = \textstyle\sum_{s=1}^{g} t_s \\
\textbf{s.t. (1)} \quad & t_s \geq 0, \ \text{ for } s = 1,\dots,g, \\
\textbf{(2)} \quad & \textstyle\sum_{s=\alpha_{ij}}^{\beta_{ij}} t_s \geq p_{ij}, \ \text{ for } i = 1,\dots,\mu \text{ and all } j \in L_x, \\
\textbf{(3)} \quad & L_{(s,\tau)}(\xi) \leq t_s, \text{ for } s = 1,\dots,g \text{ and all } \tau \in M, \\
\textbf{(4)} \quad & \textstyle\sum_{\vec{a} \in A_{\bar{\varphi}}} \xi_{\vec{a}} = 1, \ \text{ for all } \bar{\varphi}, \\
\textbf{(5)} \quad & \xi_{\vec{a}} \geq 0, \ \text{ for all } \vec{a} \in A_{\bar{\varphi}} \text{ and all } \bar{\varphi},
\end{aligned}$$

where the variables have the following interpretation:

t_s**:** the *length* of snapshot $M(s)$, and

$L(\xi,t)$**:** the *total schedule length*.

Accordingly, the constraints have the following meaning: **(2)** each long operation o_{ij} fits into the corresponding snapshots $M(\alpha_{ij}),\dots,M(\beta_{ij})$, for $i = 1,\dots,\mu$; **(3)** the load on each machine τ in snapshot $M(s)$ does not exceed the snapshot length t_s; **(4)-(5)** all short jobs in $S_x^{\bar{\varphi}}$ are assigned completely.

Proposition 1.4.19. *There exists a relative schedule of L_x such that $L(\xi,t) \leq |I|_x$. Furthermore, if each value $D(S_x^{\bar{\varphi}})$ increases by at most $\varepsilon^2|I|_x/2\overline{\nu}^*$, then $L(\xi,t)$ increases by at most $\varepsilon^2|I|_x/2$, i.e. $L(\xi,t) \leq |I|_x + \varepsilon^2|I|_x/2$.*

Proof. Recall that there exists a schedule for the operations of jobs in sets $J_x^{\bar{\varphi}}$ which length is at most $|I|_x$. We can "copy" a relative schedule of L_x from this schedule. Due to the LP formulation, there exists a solution (ξ,t) of the LP such that $L(\xi,t) \leq |I|_x$.

We fix the values of ξ and increase each value of $D(S_x^{\bar{\varphi}})$ by $\varepsilon^2|I|_x/2\overline{\nu}^*$. Then, the values of $L_{(s,\tau,\bar{\varphi})}(\xi)$ increase as well, see (1.52), but constraints **(4)-(5)** of the LP remain valid.

Now we fix the values of ξ and solve the LP over variables $t = (t_s)$. Informally, we restrict the LP to constraints **(1)-(3)** and find new values for $t = (t_s)$. Let $(\xi,\tilde{t})$ be a new solution. In the following we show that $L(\xi,\tilde{t})$ is at most $L(\xi,t) + \varepsilon^2|I|_x/2$.

Indeed, by the above construction the objective value

$$L(\xi,\tilde{t}) = \sum_{s=1}^{g} \tilde{t}_s \;=\; \sum_{s=1}^{g} \tilde{L}_{(s,\tau_s)}(\xi), \tag{1.54}$$

where $\tilde{t}_s = \tilde{L}_{(s,\tau_s)}(\xi)$ for some machine $\tau_s \in M$, $s = 1,\ldots,g$. (See **(3)** in the LP.) Then, from (1.52) each load

$$\begin{aligned}\tilde{L}_{(s,\tau_s)}(\xi) \;&\leq\; \sum_{\bar{\varphi}} \sum_{\vec{d}\in A_{\bar{\varphi}}} \left(\sum_{i\in B_{\bar{\varphi}}(s,\tau_s,\vec{d})} \xi_{\vec{d}} \cdot \pi_i \cdot (D(S_x^{\bar{\varphi}}) \;+\; \varepsilon^2 |I|_x / 2\bar{\nu}^*) \right) \\ &=\; L_{(s,\tau_s)}(\xi) \;+\; \sum_{\bar{\varphi}} \sum_{\vec{d}\in A_{\bar{\varphi}}} \left(\sum_{i\in B_{\bar{\varphi}}(s,\tau_s,\vec{d})} \xi_{\vec{d}} \cdot \pi_i \right) \cdot \varepsilon^2 |I|_x / 2\bar{\nu}^*.\end{aligned} \tag{1.55}$$

We know that $\pi_i \leq 1$ (Lemma 1.4.7), and from (1.49), (1.50) it follows that the number of indices

$$|\cup_{s=1}^{g} B(s,\tau_s,\vec{d})| \;\leq\; \mu.$$

Hence,

$$\begin{aligned}\sum_{s=1}^{g} \left(\sum_{\bar{\varphi}} \sum_{\vec{d}\in A_{\bar{\varphi}}} \left(\sum_{i\in B_{\bar{\varphi}}(s,\tau_s,\vec{d})} \xi_{\vec{d}} \right) \right) \;&=\; \sum_{\bar{\varphi}} \sum_{\vec{d}\in A_{\bar{\varphi}}} \left(\sum_{s=1}^{g} \left(\sum_{i\in B_{\bar{\varphi}}(s,\tau_s,\vec{d})} \xi_{\vec{d}} \right) \right) \\ &\leq\; \sum_{\bar{\varphi}} \sum_{\vec{d}\in A_{\bar{\varphi}}} \mu \cdot \xi_{\vec{d}}.\end{aligned} \tag{1.56}$$

From **(3)-(4)** of the LP it follows

$$\sum_{\vec{d}\in A_{\bar{\varphi}}} \xi_{\vec{d}} = 1 \;\text{ and }\; \sum_{s=1}^{g} L_{(s,\tau_s)}(\xi) \leq \sum_{s=1}^{g} t_s.$$

The number of local profiles is bounded by $\bar{\nu}^*$ (Lemma 1.4.13). Thus, we get

$$\begin{aligned}L(\xi,\tilde{t}) = \sum_{s=1}^{g} \tilde{t}_s \;&\leq\; \sum_{s=1}^{g} t_s \;+\; \varepsilon^2 |I|_x / 2 \\ &=\; L(\xi,t) \;+\; \varepsilon^2 |I|_x / 2.\end{aligned} \tag{1.57}$$

□

Assignment to Snapshots: Unbalanced Short Jobs. Take a relative schedule $M(1), \ldots, M(g)$ and an optimal solution (ξ, t) of the LP with the objective value $L(\xi,t) \leq |I|_x$ (Proposition 1.4.19). Look at the LP: **(3)** there are mg constraints on $\xi = (\xi_{\vec{a}})$; **(4)** there is at least one assignment $\vec{a} \in A_{\bar{\varphi}}$ such that $\xi_{\vec{a}} > 0$. Hence, w.l.o.g. we can reassign the values of ξ such that that there are at most mg distinct assignments $\vec{a}$ for which $1 > \xi_{\vec{a}} > 0$ (they may correspond to several local profiles), and for all other assignments $\vec{a}$ either $\xi_{\vec{a}} = 1$ or $\xi_{\vec{a}} = 0$. (See Appendix B on page 197.) Notice that $\xi_{\vec{a}} = 1$ only for one $\vec{a} \in A_{\bar{\varphi}}$.

Now we assign the short operations of S_x to snapshots $M(1), \ldots, M(g)$. For each local profile $\bar{\varphi}$ we assign the operations of $S_x^{\bar{\varphi}}$ with respect to $(\xi_{\vec{a}})$, $\vec{a} =< s_1, \ldots, s_\mu > \in A_{\bar{\varphi}}$. If $\xi_{\vec{a}} = 1$, then we select all jobs j from $S_x^{\bar{\varphi}}$. If $1 > \xi_{\vec{a}} > 0$, we select jobs j from $S_x^{\bar{\varphi}}$ in a greedy manner until the total length of the selected jobs does not exceed $\xi_{\vec{a}} \cdot D(S_x^{\bar{\varphi}})$. Then, for each selected job j we assign operation o_{ij} to snapshot $M(s_i)$, $i = 1, \ldots, \mu$. By (1.49), each o_{ij} is assigned to to a free machine in snapshot $M(s_i)$. Here, the last selected job, which can cause the exceed of value $\xi_{\vec{a}} \cdot D(S_x^{\bar{\varphi}})$, is also assigned with respect to $\vec{a}$. Since there are at most mg distinct assignments $\vec{a}$ for which $1 > \xi_{\vec{a}} > 0$, there are at most mg such short jobs in S_x, and we call them *unbalanced*.

Deducing a Schedule: Sevastianov's Algorithm. Here we will use Sevastianov's algorithm [Sev86]. Assume that we are given an instance of the makespan version of the job shop problem. There are m machines and μ operations per job. Let $\Pi = \max_{\tau \in M} \sum_{\tau_{ij} = \tau} p_{ij}$ be the maximum machine load and $p_{\max} = \max_{ij} p_{ij}$ be the maximum operation processing time. Then, Sevastianov's algorithm outputs a schedule of length at most $\Pi + 2m\mu^3 p_{\max}$.

We deduce a schedule for the long operations of L_x as it is defined in snapshots $M(1), \ldots, M(g)$. In order to deduce a schedule for the short operations of S_x we proceed as follows.

First, for $s = 1, \ldots, g$, we define the set $\mathcal{O}(s)$ of short operations assigned to snapshot $M(s)$. The maximum load of $\mathcal{O}(s)$ can be bounded by the length t_s of snapshot $M(s)$ with some additive increase $\Delta(s) \geq 0$, called the *snapshot enlargement*. The overall snapshot enlargement $\sum_{s=1}^{g} \Delta(s)$ is bounded by the total length of the unbalanced jobs from S_x.

Next, we apply Sevastianov's algorithm to each set $\mathcal{O}(s)$. The length of the output schedule for $\mathcal{O}(s)$ is at most

$$t_s \; + \; \Delta(s) \; + \; 2m\mu^3 p_{\max}(s), \tag{1.58}$$

where $p_{\max}(s)$ is the length of the longest operation in $\mathcal{O}(s)$.

Finally, we unite schedules over all snapshots $M(s)$, $s = 1, \ldots, g$ into one feasible schedule for the jobs of S_x and L_x. The length of the deduced schedule is at most

$$\sum_{s=1}^{g} (t_s + \Delta(s) + 2m\mu^3 p_{\max}(s)) \leq L(x,t) + \Delta_x, \tag{1.59}$$

where

$$\Delta_x := \sum_{s=1}^{g} \Delta(s) + 2m\mu^3 \sum_{s=1}^{g} p_{\max}(s) \tag{1.60}$$

is the *schedule enlargement*.

Defining the values of k_x^* and q_2^*. Here, we define the values of k_x^* and q_2^* such that

$$\Delta_x \leq \varepsilon^2 |I|_x / 2. \tag{1.61}$$

There are two important values

$$\Delta_x^1 := \sum_{s=1}^{g} p_{\max}(s) \text{ and } \Delta_x^2 := \sum_{\ell=1}^{g} \Delta(\ell). \tag{1.62}$$

Form one side, we can bound the value of Δ_x^1 by the total length of the first longest g jobs in S_x. From another side, Δ_x^2 corresponds to mg unbalanced jobs in S_x, and we can bound its value by the total length of the first longest mg jobs in S_x, respectively.

Now we use (1.43), (1.45) and (1.47). Then, $g \leq 2\mu|L_x|$. Let D be the total length of the first $2\mu|L_x|$ jobs in S_x. These jobs are numbered by

$$|L_x|, |L_x|, \ldots, |L_x| + 2\mu|L_x| - 1.$$

Then, $2m \cdot D$ is an upper bound for $\Delta_x^1 + \Delta_x^2$, and $2m\mu^3 \cdot 2m \cdot D$ is an upper bound for $2m\mu^3(\Delta_x^1 + \Delta_x^2)$. From (1.60) and (1.62) we have that

$$\Delta_x \leq 4m^2\mu^3(\ell_{|L_x|} + \ell_{|L_x|+1} + \ldots + \ell_{|L_x|+2\mu|L_x|-1}). \tag{1.63}$$

Next, we use the following result.

Proposition 1.4.20 ([CM99, JP99b]). *Suppose $\ell_0 \geq \ell_1 \geq \ldots \geq \ell_{n'} > 0$ is a sequence of real numbers and $P \geq \sum_{j=0}^{n'} \ell_j$. Let β be a nonnegative integer, $\alpha > 0$, and assume that n' is sufficient large (i.e. all the indices of the ℓ_j's in the statement are smaller than n'; e.g. $n' > (\beta+1)^{\lceil \frac{1}{\alpha} \rceil}$ suffices). Then, there exists an integer $k = k(\beta,\alpha)$ such that $\ell_k + \ldots + \ell_{k+\beta k-1} \leq \alpha \cdot P$, and $k \leq (\beta+1)^{\lceil \frac{1}{\alpha} \rceil - 1}$.*

From (1.42) and (1.43) we can find an upper bound P on $\sum_{j=0}^{n'} \ell_j$. We simply define

$$P = 2^{me^*}|I|_x, \quad \alpha = \varepsilon^2/(8m^2\mu^3 2^{me^*}) \quad \text{and} \quad \beta = 2\mu. \tag{1.64}$$

Then, by Proposition 1.4.20 we can define

$$k_x^* := k(\beta,\alpha) \leq (2\mu+1)^{\left\lceil \frac{8m^2\mu^3 2^{me^*}}{\varepsilon^2} \right\rceil - 1} \tag{1.65}$$

such that

$$4m^2\mu^3(\ell_{k_x^*} + \ell_{k_x^*+1} + \ldots + \ell_{k_x^*+2\mu k_x^*-1}) \leq \varepsilon^2 |I|_x/2. \tag{1.66}$$

Now we use (1.44), (1.45) and (1.46), respectively. We define the value of q_2^* such that (1.61) holds. Consider case (1). We have $|L_x| = k_x^*$ and $|U_x| > k_x^*$. Then, $\Delta_x \leq \varepsilon^2|I|_x/2$ holds by (1.63) and (1.66). Consider case (2). We have $L_x = U_x$ and $S_x = J_x \setminus U_x$. Then, $|L_x| = |U_x| \leq k_x^*$ and for each shot job j in S_x it holds $\ell_j < \varepsilon^2|I|_x/q_2^*$. From (1.65) we have

$$|L_x| \leq k_x^* \leq (2\mu+1)^{\left\lceil \frac{8m^2\mu^3 2^{me^*}}{\varepsilon^2} \right\rceil - 1}. \tag{1.67}$$

From (1.63) we have

$$\begin{aligned} \Delta_x &\leq 4m^2\mu^3(\ell_{|L_x|} + \ell_{|L_x|+1} + \ldots + \ell_{|L_x|+2\mu|L_x|-1}) \\ &\leq 4m^2\mu^3 \left(2\mu|L_x|\frac{\varepsilon^2|I|_x}{q_2^*}\right). \end{aligned} \tag{1.68}$$

Hence, from (1.67) it follows that

$$\begin{aligned} \Delta_x &\leq (2m\mu)^4 k_x^* \left(\frac{\varepsilon^2|I|_x}{q_2^*}\right) \\ &\leq (2m\mu)^4 (2\mu+1)^{\left\lceil \frac{8m^2\mu^3 2^{me^*}}{\varepsilon^2} \right\rceil - 1} \left(\frac{\varepsilon^2|I|_x}{q_2^*}\right) \\ &\leq (2\mu+1)^{\frac{(2m\mu)^3 2^{me^*}}{\varepsilon^2} + (2m\mu)^4} \left(\frac{\varepsilon^2|I|_x}{q_2^*}\right) \\ &\leq (2\mu+1)^{\left\lceil \frac{(2m\mu)^5 2^{me^*}}{\varepsilon^2} \right\rceil} \left(\frac{\varepsilon^2|I|_x}{q_2^*}\right) \end{aligned}$$

For $2m\mu \leq 1/\varepsilon$ we define

$$q_2^* \;=\; (2\mu+1)^{\left\lceil \frac{2me^*}{\varepsilon^7} \right\rceil} \;\geq\; 2(2\mu+1)^{\left\lceil \frac{(2m\mu)^5 2me^*}{\varepsilon^2} \right\rceil}.$$

This gives $\Delta_x \leq \varepsilon^2 |I|_x / 2$.

Solving the Simplified Problem. One can see that we have deduced a schedule for the operations of sets $J_x^{\bar{\varphi}} = \cup N_x^{\bar{\varphi}} \cup Y_x^{\bar{\varphi}}$. From (1.59) and (1.61) the length of schedule is at most

$$L(\xi,t) \;+\; \varepsilon^2|I|_x/2 \;\leq\; |I|_x \;+\; \varepsilon^2|I|_x/2 \;\leq\; (1+\varepsilon)|I|_x \;=\; |I|_{x+1}. \qquad (1.69)$$

Furthermore, even if we replace the tiny jobs of $Y_x^{\bar{\varphi}}$ by the tiny jobs of $\underline{Y}_x^{\bar{\varphi}}$ in each set $J_x^{\bar{\varphi}}$, we will be able to deduce a schedule for the operations of sets $\underline{J}_x^{\bar{\varphi}}$. Indeed, all new jobs from $\underline{J}_x^{\bar{\varphi}}$ will fall into set $S_x^{\bar{\varphi}}$ of short jobs. (See the definition of set S_x and sets $S_x^{\bar{\varphi}}$.) Due to (ii) and (iii) in the definition of general problem, this increases the value of $D(S_x^{\bar{\varphi}})$ by at most

$$\varepsilon^2|I|_x/2\overline{\nu}^* q_2^* \;<\; \varepsilon^2|I|_x/2\overline{\nu}^*.$$

By Proposition 1.4.19, the objective function value $L(\xi,t)$ increases by at most $\varepsilon^2|I|_x/2$, i.e.

$$L(\xi,t) \;\leq\; |I|_x \;+\; \varepsilon^2|I|_x/2.$$

Thus, by following the same line of ideas, we can deduce a schedule which length is bounded by

$$L(\xi,t) \;+\; \varepsilon^2|I|_x/2 \;+\; \varepsilon^2|I|_x/2 \;<\; (1+\varepsilon)|I|_x \;=\; |I|_{x+1}. \qquad (1.70)$$

Solving the General Problem. Clearly, we can also follow all above procedures in the case when the set of crossing huge jobs $K_x \neq \emptyset$. However, we need to adjust the values of k_x^* and q_2^*. Furthermore, we need to adjust the processing times of the crossing operations, see Figure 1.21 b) on 59.

We simply add all jobs in K_x into set L_x of long jobs. As we discussed, there are at most $2m$ crossing operations. Hence, $|K_x| \leq 2m$ and the number of snapshots in any relative schedule can be bounded as follows

$$g \;\leq\; 2\mu|L_x| \;\leq\; 2\mu(|L_x \setminus K_x| + 2m) \leq 4\mu m|L_x \setminus K_x|.$$

Then, in (1.64) the value of β changes to $2\mu m$, and in (1.67) we have

$$|L_x \backslash K_x| \leq k_x^* := k(\beta,\alpha) \leq (\mu m+1)^{\left\lceil \frac{8m^2\mu^3 2^{me^*}}{\varepsilon^3} \right\rceil - 1}.$$

In (1.68) we also have

$$\Delta_x \leq 4m^2\mu^3(4\mu m|L_x \backslash K_x|)\left(\frac{\varepsilon^2|I|_x}{q_2^*}\right) \leq (2\mu m)^4 k_x^* \left(\frac{\varepsilon^2|I_x|}{q_2^*}\right).$$

Similarly, we can define

$$q_2^* = (4\mu m + 1)^{\left\lceil \frac{2^{me^*}}{\varepsilon^7} \right\rceil}.$$

Hence, we can claim (1.70).

In total, we can deduce a schedule for the operations of set K_x and sets $N_x^{\bar{\varphi}}$, $\widetilde{Y}_x^{\bar{\varphi}}$ which will fit inside interval I_{x+1}. This completes the proof of Lemma 1.4.18.

1.4.9 Weight-Shifting and Merging

After we have proved the modified Smith's rule for tiny jobs, the compacting step of our method is quite simple. We can follow the same line of ideas as for the single machine problem $1|r_j|\sum w_j C_j$. We simply use weight-shifting and merging described in Sections 1.3.7 and 1.3.8, respectively.

Weight-Shifting. Assume that at some release date R_x we have a lot of huge jobs (H_x) and tiny jobs (T_x). Which jobs can wait until the next interval? Take one profile π. The jobs of $H_x(\pi)$ having the same size must complete by decreasing weights w_j. By Lemmas 1.4.12,1.4.16,1.4.18 there is at most a constant number of such sizes. By Lemma 1.4.14, the jobs of $T_x(\pi)$ must complete by decreasing ratio w_j/ℓ_j. By Lemma 1.4.9, all jobs that start in I_x must complete within the next e^* intervals. We select only the jobs that can be potentially scheduled in I_x.

Lemma 1.4.21. *With $1+O(\varepsilon)$ loss and in $O(n\log n)$ time, we can enforce $D(T_x) \leq t^*|I|_x$ and $|H_x| \leq H^*$ at each release date R_x, where t^* and H^* are some constant.*

Merging. At each release date R_x we partition the ordered set of tiny jobs $T_x(\pi)$ into subsets of roughly equal size $\approx \varepsilon^2|I|_x/2q^*$ (but less than $\varepsilon^2|I|_x/q^*$). Then, we merge the jobs of each such subset into a new tiny job of profile π.

Lemma 1.4.22. *With $1+O(\varepsilon)$ loss and in $O(n\log n)$ time, we can enforce $|T_x| \leq T^*$ and $|H_x| \leq H^*$ at each release date R_x, where T^* and H^* are some constant.*

1.4.10 Blocks and Dynamic Programming

Here, we follow the ideas of Lemmas 1.3.8,1.3.14 and Lemma 1.4.7. As in the one machine case, we can consider only near-optimal schedules where every job to be completed within a constant number of intervals after its release date.

Lemma 1.4.23. *With $1+\varepsilon$ loss, we can assume that each job completes within d^* intervals after its release in a schedule, where d^* is some constant.*

Similarly to Section 1.3.9, we can define blocks, and then use the dynamic programming framework presented in Section 1.3.10. The combination of the structuring and compacting steps with a dynamic programming algorithm gives a PTAS with $O(n \log n)$ running time. This completes the proof of Theorem 1.4.1.

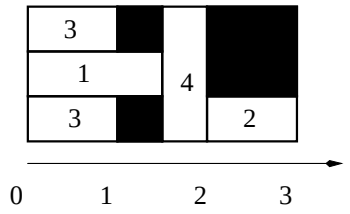
1.5 A PTAS FOR THE MULTIPROCESSOR TASK SCHEDULING PROBLEM

In this section we address the following multiprocessor task scheduling problem. We are given a set of n tasks $T = \{1,2,\ldots,n\}$ and a set $M = \{1,\ldots,m\}$ of m processors. Each task $j \in T$ has a positive weight w_j, a release date r_j, and an associated function $p_{\cdot j} : 2^M \longrightarrow \mathbb{Z}^+ \cup \{+\infty\}$ that gives the processing time $p_{\tau j}$ of task j in terms of the set of processors $\tau \subseteq M$ that are *allotted* to j. (Here 2^M denotes the set of all non-empty subsets of M.) Each processor can work on at most one task at a time, and given an *allotment* $\tau(j) \subseteq M$ for a task $j \in T$, the processors of $\tau(j)$ are required to execute task j in union during $p_{\tau(j)j}$ time units. Here we assume that m is fixed, and the goal is to find a task allotment and a non-preemptive feasible schedule under this allotment such that $\sum w_j C_j$ is minimized.

We will consider the following two basic versions. In the dedicated version, each task $j \in T$ has a processing time p_j and a prespecified processor set $\mathrm{fix}_j \subseteq M$ such that for function $p_{\cdot j}$ it holds $p_{\tau j} = p_j$ if $\tau = \mathrm{fix}_j$, and $p_{\tau j} = \infty$ otherwise. In the parallel version, each task $j \in T$ has a processing time p_j and a prespecified number $\mathrm{size}_j \in M$ such that for function $p_{\cdot j}$ it holds $p_{\tau j} = p_j$ if $|\tau| = \mathrm{size}_j$, and $p_{\tau j} = \infty$ otherwise. For an illustration see Figure 1.23 on the facing page and Figure 1.24 on the next page.

As the jobs shop problem considered in Section 1.4, the above stated problem of scheduling multiprocessor (dedicated and parallel) tasks is also a generalization of the single machine problem $1|r_j|\sum w_j C_j$. Here we also follow the main parts of our method: (1) Structuring, (2) Compacting, and, (3) Dynamic Programming. Indeed, in coping with multiprocessor tasks we combine all already known techniques from both the single machine case and the job shop case. In part (1), Sec-

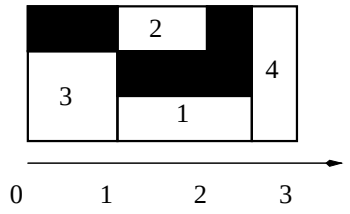
j	1	2	3	4
fix_j	$\{2\}$	$\{3\}$	$\{1,3\}$	$\{1,2,3\}$
w_j	10	10	5	5
p_j	1.5	1	1	0.5
r_j	1.5	1	0	1.5



$$\sum w_j C_j = 10 \times 1.5 + 10 \times 3 + 5 \times 1 + 5 \times 2 = 50$$

Figure 1.23: A schedule for 4 dedicated tasks

j	1	2	3	4
size_j	1	1	2	3
w_j	10	10	5	5
p_j	1.5	1	1	0.5
r_j	1.5	1	0	1.5



$$\sum w_j C_j = 10 \times 2.5 + 10 \times 2 + 5 \times 1 + 5 \times 3 = 65$$

Figure 1.24: A schedule for 4 parallel tasks

tions 1.5.1–1.5.8, and in parts (2) and (3), Sections 1.4.9 and 1.4.10, we just apply adopted transformations. However, in many things we add some novel non-trivial ideas which were not used before.

As the first step of part (1), in Section 1.5.1, we perform basic structuring of the problem. Similar to the single machine case, we can round processing times, release dates, weights and introduce intervals.

As the next step of part (1), in Sections 1.5.5 and 1.5.4, similar to the job shop case, we introduce *profiles* and classify tasks as *huge* and *tiny* ones.

As the final step of part (1), in Section 1.5.6–1.5.8, we formulate and prove Smith's rule for scheduling tiny tasks of the same profile. Following the main ideas of our method, we first use an LP formulation, LP rounding procedure for "assigning" tiny jobs to intervals, and then a special procedure for "packing" tasks inside single intervals. From one side, the LP formulation and its rounding procedure here are similar to the ones in the single machine case. From another side, the packing procedure here is similar to the one in the job shop case, we use a PTAS for the makespan version of the problem [ABKM97, CM99, JP99b].

In part (2) of our method, Section 1.4.9, we apply the well known weight-shifting and merging techniques. We enforce that there is at most a constant number of tasks released at each interval, and their total processing time is a small multiple of the size of the interval.

Surprisingly, the main difficulty occurs in part (3) of our method. There is one very important feature of the problem. Here, any task can require more than one processor. Thus, while tasks run in a schedule, more "earlier" tasks can intersect in a very irregular way blocking more "later" tasks.

In order to copy with this, we can enforce regular *gaps* in a schedule, see corresponding Sections1.5.2 and 1.5.3. A *gap* is an interval in time where all the processors are idle. This simple idea allows us to show that in a near-optimal schedule every task can be completed within a constant number of intervals after its release date. Similar to the single machine case, as it described Sections 1.3.9 and 1.3.10, we can introduce a block structure and apply the dynamic programming framework.

In total, we conclude with the following main result:

Theorem 1.5.1. *There are PTASs for* $Pm|\,\mathrm{fix}_j, r_j|\sum w_jC_j$, $Pm|\,\mathrm{size}_j, r_j|\sum w_jC_j$ *and* $Pm|\,\mathrm{set}_j|\sum w_jC_j$ *that compute for any fixed* m *and* $\varepsilon > 0$ *accuracy,* $(1+\varepsilon)$*- approximate schedules in* $O(n\log n)$ *time.*

Unfortunately, for the case of general multiprocessor tasks we can only define task *profiles* in the case when tasks have no release dates. (The same technique works

for the case when preemptions are allowed.) In this particular case, we can follow the same line of ideas and construct a PTAS. When we add release dates to general multiprocessor tasks, the definition of *profiles* becomes very complex. Here, we just give some preliminary results and point out some main ideas which can lead to a PTAS. However, we conclude with the following:

Conjecture 1.5.2. *There is a PTAS for* $Pm|\,\mathrm{set}_j, r_j|\sum w_j C_j$.

1.5.1 Basic Structuring

In the following we consider only dedicated and parallel versions of the problem. For both variants we will use $p_{\cdot j}$ and p_j to denote the function and the processing time associated with task j. We will also use $\tau(j)$ to denote the the allotment of task j. In the dedicated version, where it holds $p_{\tau j} = p_j$ if $\tau = \mathrm{fix}_j$, and $p_{\tau j} = \infty$ otherwise, we define $\tau(j)$ be equal to fix_j. In the parallel version, where it holds $p_{\tau j} = p_j$ if $|\tau| = \mathrm{size}_j$, and $p_{\tau j} = \infty$ otherwise, we define $|\tau(j)|$ be equal to size_j. To unify these two basic versions, we will use $p_j := \min_\tau p_{\tau j}$ to denote the *length* of task j. Notice that the length p_j is just the processing time of job j in both dedicated and parallel but the general version.

To simplify notation we will use that $1/\varepsilon$ is integral (in particular $\varepsilon \leq 1/2^m$). For a task j, we will use S_j and C_j to denote the start and completion time of j. For a task set X, we use $D(X) := \sum_{j\in X} p_j$ to denote the total length of X. As before, OPT denotes the objective value of the optimal schedule, I_x denotes interval $[R_x, R_{x+1})$ and $|I|_x$ denotes its length $R_{x+1} - R_x$, where $R_x = (1+\varepsilon)^x$ for integer x.

As in previous cases, we first use geometric rounding to create a well-structured set of task processing times, release dates and weights:

Lemma 1.5.3. *With* $1+\varepsilon$ *loss and in* $O(n)$ *time , we can enforce all* $p_{\tau j}$, $\tau \in 2^m$, *and* r_j *be integer powers of* $1+\varepsilon$.

Hence, all release dates r_j are of the form $R_x = (1+\varepsilon)^x$ for some integer x.

Lemma 1.5.4. *With* $1+\varepsilon$ *loss and in time* $O(n)$, *we can enforce all* w_j/p_j *be distinct.*

Next, we use stretching to structure schedules:

Lemma 1.5.5. *With* $1+\varepsilon$ *loss, we can assume that all* $S_j \geq \varepsilon p_{\tau(j)j}$ *in a schedule.*

Lemma 1.5.6. *With* $1+\varepsilon$ *loss and in* $O(n)$ *time, we can enforce all* $r_j \geq \varepsilon p_j$.

Lemma 1.5.7. *With* $1+\varepsilon$ *loss, we can assume that any task crosses at most* $s^* = \lceil \log_{1+\varepsilon}(1+\frac{1}{\varepsilon})\rceil$ *intervals in a schedule.*

1.5.2 The Schedule-Shifting Technique

Take a feasible schedule. For each task j, let $\tau(j)$ be the allotment, $z(j)$ and $y(j)$ be whose indices for which $S_j \in R_{y(j)}$ and $C_j \in R_{z(j)}$. (See Figure 1.25.)

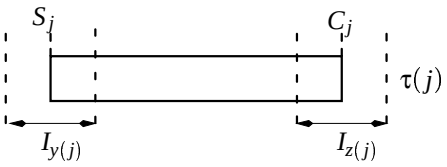


Figure 1.25: Indices

Then, we create a new schedule by setting $S'_j = S_j - R_{z(j)} + R_{z(j)+1}$ for all tasks j. The new schedule is feasible, and the distance between C_j and the end of interval $I_{z(j)}$ is preserved. This generates $\varepsilon(R_{z(j)} - R_{y(j)})$ additional idle time on the processors of $\tau(j)$ before the start of a task j. See Figure 1.26. In a schedule

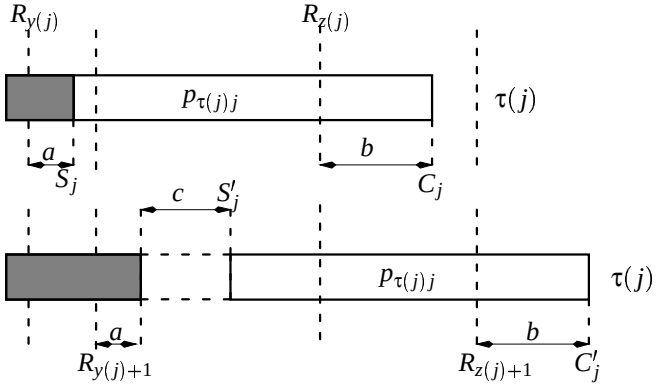


Figure 1.26: Shifting

$S_j = R_{y(j)} + a$ and $C_j = R_{z(j)} + b$ for a task j. When constructing the new schedule, a and b are preserved, i.e. $C'_j = R_{z(j)+1} + b$ and the processors in $\tau(j)$ may be busy

up to $R_{y(j)+1}+a$. Thus, the created idle time for job j is equal to

$$\begin{aligned} c &= S'_j - (R_{y(j)+1} + a) = C'_j - p_{\tau(j)j} - R_{y(j)+1} - a \\ &= R_{z(j)+1} + b - p_{\tau(j)j} - R_{y(j)+1} - a = \\ &= \varepsilon(R_{z(j)} - R_{y(j)}) + (R_{z(j)} + b) - p_{\tau(j)j} - (R_{y(j)} + a) \\ &= \varepsilon(R_{z(j)} - R_{y(j)}) + S_j - p_{\tau(j)j} - C_j \\ &= \varepsilon(R_{z(j)} - R_{y(j)}). \end{aligned}$$

1.5.3 Creating of Gaps

A *gap* is an interval in time where all the processors of M are idle. We can enforce regular gaps in a schedule.

Lemma 1.5.8. *With $1+8\varepsilon$ loss, we can assume that there is a gap within any sequence of $4s^*/\varepsilon$ consecutive intervals of a schedule.*

Proof. Take an optimal schedule of value OPT. For an illustration see Figure 1.27 on the next page. We first partition it into a sequence of *superblocks*, where each superblock consists of $2s^*/\varepsilon$ consecutive intervals. Here $s^* = \lceil \log_{1+\varepsilon}(1+\frac{1}{\varepsilon}) \rceil$, Lemma 1.5.7. Next, we partition each superblock into $1/\varepsilon$ *blocks*, where each block consists of $2s^*$ consecutive intervals. See Figure a) 1.27. There are $1/\varepsilon$ blocks in a superblock. Hence, there is one block out of these $1/\varepsilon$ blocks in which the average weighted completion time of the tasks is at most an ε factor of the total weighted completion time of the tasks that complete in the superblock.

Take such a block, say block B. Let b and $b' = b + 2s^* - 1$ be the indices of the first and last B's intervals. Let also $t = I_b/\varepsilon^2$ be the middle point in B. Note that

$$R_b + t = R_b + I_b/\varepsilon^2 = R_b(1 + 1/\varepsilon) = R_{b+s^*}.$$

We split the tasks that complete in B into two subsets T^- and T^+ of tasks j with completion times $C_j \le R_b + t$ and $C_j > R_b + t$, respectively. See Figure a) 1.27.

We shift the optimal schedule. In this new schedule the tasks of $T = T^- \cup C^+$ run in $[R_{b+1}, R_{b'+2})$, i.e. in intervals $I_{b+1}, \ldots, I_{b'+1}$, and each task j has at least $\varepsilon(R_{z(j)} - R_{y(j)})$ idle time on the allotted processors of $\tau(j)$. See Section 1.5.2, and Figure b) 1.27.

Then, we reschedule the tasks of T as follows:

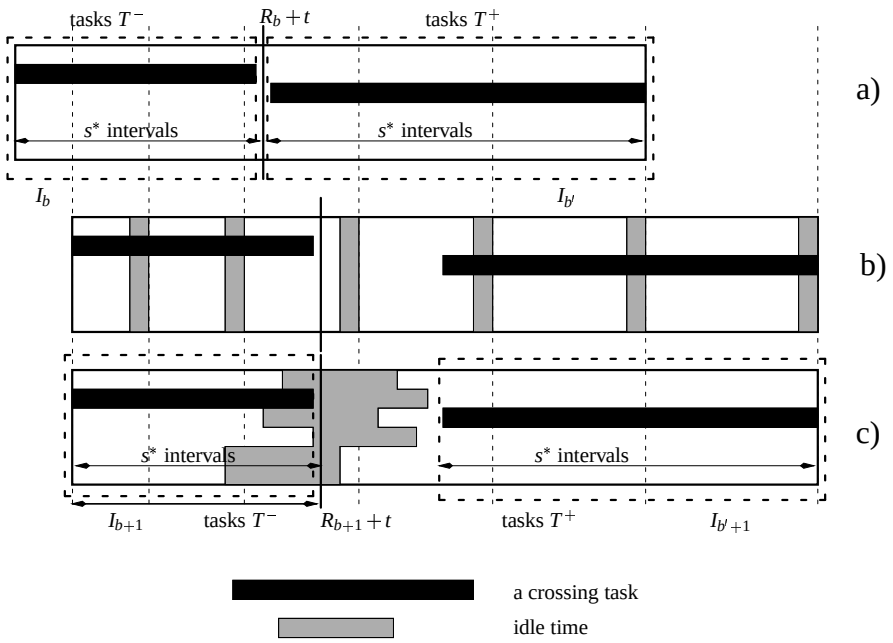


Figure 1.27: Creating a gap

Step 1. Reschedule the tasks from T^- by eliminating all created idle time from R_{b+1} up to $R_{b+1} + t_i$ (i.e. the tasks are shifted backwards),

Step 2. Reschedule the tasks form T^+ by eliminating all created idle time from $R_{b+1} + t_i$ up to $R_{b'+2}$ (i.e. the tasks are shifted forwards).

Thus, the idle time within all intervals $I_{b(i)+1}, \ldots, I_{b'(i)+1}$ is moved to the time point $R_{b(i)+1} + t$. See Figure c) 1.27.

Since in any schedule tasks start and complete consecutively, and $s^* = \lceil \log_{1+\varepsilon}(1+\frac{1}{\varepsilon}) \rceil$, the total idle time in intervals $I_{b(i)+1}, \ldots, I_{b'(i)+1}$ is at least

$$\begin{aligned}
\sum_{\ell=b}^{b'-1} \varepsilon I_\ell &= \varepsilon (R_{b+2s^*} - R_b) \\
&= \varepsilon R_b ((1+\varepsilon)^{2s^*} - 1) \\
&\geq I_b ((1+1/\varepsilon)^2 - 1) \\
&\geq I_b/\varepsilon^2.
\end{aligned}$$

Recall that each task crosses at most s^* intervals. Thus, no task from T^- crosses the time point $\bar{t} = R_{b+1} + t$. To observe that no task from T^+ crosses $\bar{t}$ we use the following fact. The distance between any task in T^- and any task in T^+ on any machine has to be at least I_b/ε^2. Hence, since all tasks from T^- complete before $\bar{t} = R_{b+1} + t = R_{b+1} + I_b/\varepsilon^2$, no task is processed on any machine at $\bar{t}$. Thus, $\bar{t}$ is a gap.

Consider the tasks of T^+. By the above procedure, only their completion times can increase dramatically. However, we can bound them as follows

$$\begin{aligned} C_j &\leq C_j + \varepsilon(R_{b+2s^*+1} - R_b) = C_j + \varepsilon R_b((1+\varepsilon)^{2s^*+1} - 1) \\ &= C_j + \varepsilon R_b((1+1/\varepsilon)^2(1+\varepsilon) - 1) \leq C_j + \varepsilon R_b(2(1+1/\varepsilon)^2 - 1) \\ &\leq C_j + R_b(5 + 2/\varepsilon) \leq C_j + 5R_b(1 + 1/\varepsilon) \\ &= C_j + 5t \leq 6C_j. \end{aligned}$$

In a similar way we reschedule tasks in each superblock of the schedule. The objective value of all blocks is just an ε factor of OPT. Thus, by shifting and rescheduling the objective value of the new schedule is at most

$$(1+\varepsilon)(1+6\varepsilon)OPT \leq (1+8\varepsilon)OPT.$$

Since there is a gap in each superblock, i.e. in a sequence of $2s^*/\varepsilon$ intervals, there is a gap within any sequence of $4s^*/\varepsilon$ consecutive intervals. □

1.5.4 Profiles

For a task j we can define $\pi_\tau = \infty$ if $p_{\tau j} = \infty$, and $\pi_\tau = 1$ if $p_{\tau j} = p_j$. Then, we define $|2^M|$-tuple $\pi = (\pi_\tau)_{\tau \in 2^M}$ be the *profile* of j.

Lemma 1.5.9. *The number of distinct profiles is bounded by a constant* ν^*.

Notice that two dedicated tasks j and k have the same profile iff $\text{fix}_j = fix_k$, and two parallel tasks j and k have the same profile iff $\text{size}_j = size_k$.

1.5.5 Huge and Tiny Tasks

We say that task j is *huge* in an interval I_x if $p_j \geq \varepsilon^2 |I|_x/q^*$, and *tiny* otherwise. We define the value of parameter q^* later in Lemma 1.5.16. We will write HT_x

and TT_x to denote sets of huge and tiny tasks released at R_x (HT for huge tasks and TT for tiny tasks).

By Lemma 1.5.6, we can also bound the number of huge task sizes as follows:

Lemma 1.5.10. *There are at most $z^* = O(\varepsilon, q^*)$ distinct sizes (p_j powers of $1+\varepsilon$) in HT_x.*

By using the schedule-stretching technique we can prove the following:

Lemma 1.5.11. *With $1+\varepsilon$ loss, we can assume that no tiny task crosses an interval in a schedule.*

1.5.6 Scheduling Tiny Tasks: Smith's Rule

We first need to introduce some notations. By Lemma 1.5.11 no tiny task crosses an interval in a schedule. Then, for a tiny job j we define two indices $x(j) \leq y(j)$ such that $R_{x(j)} = r_j$ and $S_j, C_j \in I_{y(j)}$. Less formally, job j is released at $R_{x(j)}$ and completely scheduled in interval $I_{y(j)}$.

Smith's Rule. Let j and k be two tiny tasks such that $x(k) \leq x(j)$ (here $r_k \leq r_j$) and $\frac{w_k}{p_k} > \frac{w_j}{p_j}$ (see Lemma 1.5.4). We say that tiny tasks *obey* Smith's rule if $y(k) \leq y(j)$ (here $S_k \leq S_j$) for all such pairs of tasks j and k. In other words, if tasks k, j are available in an interval, then task k with greater value w_k/p_k starts not later than task j with respect to intervals $I_{y(k)}$ and $I_{y(j)}$.

Now we can prove the following:

Lemma 1.5.12. *With $1+7\varepsilon$ loss, for each profile π we can assume that all tiny jobs of π obey Smith's rule in a schedule.*

Proof. We can briefly sketch the proof given in Sections 1.5.7,1.5.8 as follows. We first take an optimal schedule. Then, for each interval I_y we define sets $Y_y(\pi)$ of tiny jobs of profile π processed inside I_y. Then, by using values $D_y(\pi) = D(Y_y(\pi))$ we formulate an LP(π) which defines a fractional assignment of tiny jobs of profile π to intervals. We show that any optimal LP solution can be rounded to an integral assignment. This does not change the objective value but increases the value of each $D_y(\pi)$ by at most $\varepsilon^2 |I|_y/q^*$. By using such an integral assignment over all profiles π, we can find new indices $y(j)$ and new sets $\widetilde{Y}_y(\pi)$. Then, tiny jobs of profile π obey Smith's rule with respect to these $y(j)$. Finally, by using the time-stretching technique and a PTAS for the makespan version of the

problem [ABKM97, CM99, JP99b], we replace all sets $Y_y(\pi)$ by $\widetilde{Y}_y(\pi)$ in the optimal schedule. This gives us a schedule with the objective function value at most $(1+7\varepsilon)OPT$ in which all tiny jobs obey Smith's rule. □

1.5.7 Assigning to Intervals

Suppose we are given an optimal schedule. Take an interval I_y. Let $Y_y(\pi)$ be the set of tiny tasks of π scheduled inside interval I_y.

LP Formulation. Then, we formulate the fractional assignment problem for all tiny tasks j of profile π as the following LP(π):

$$\begin{aligned} \textbf{Minimize} \quad & F_\pi(\xi) = \textstyle\sum_j w_j \sum_{y \geq x(j)} \xi_{jy} R_y \\ \text{s.t. (1)} \quad & \Delta_y(\pi) = \textstyle\sum_j \xi_{jy}\, p_j \leq D_y(\pi) = D(Y_y(\pi)) \ \text{ for all } y, \\ \text{(2)} \quad & \textstyle\sum_{y \geq x(j)} \xi_{jy} = 1 \ \text{ for all } j, \\ \text{(3)} \quad & \xi_{jy} \geq 0 \ \text{ for all } y \geq x(j) \text{ and } j, \end{aligned} \tag{1.71}$$

where the variables have the following interpretation:

ξ_{jy}**:** the fraction of tiny task j assigned to an interval $I_y = [R_y, R_{y+1})$;

$D_y(\pi)$**:** the load in interval $I_y = [R_y, R_{y+1})$ on profile π,

$\Delta_y(\pi)$**:** the fractional load in interval $I_y = [R_y, R_{y+1})$ on profile π, and

$F_\pi(\xi)$**:** the average weighted fractional completion time of tiny tasks of profile π.

Accordingly, the constraints have the following meaning: **(1)** in each interval I_y the fractional load $\Delta_y(\pi)$ on profile *pi* is at most $D_y(\pi)$, **(2)-(3)** each tiny task j is assigned completely.

Informally, in the optimal schedule we allow tiny jobs of profile π be fractionally processed in several intervals. From one side, we assign each yth fraction of job j be completed in interval $I_y = [R_y, R_{y+1})$, and take R_y as its "completion time". From another side, preserving "loads" $D_y(\pi) = D(Y_y(\pi))$ in all intervals I_y, we keep the schedule of huge jobs without changes.

As in Lemmas 1.3.10 and 1.4.15, we can bound the average weighted fractional completion time as follows:

Lemma 1.5.13. *For any optimal solution ξ of the LP(π) it holds that*

$$F_\pi(\xi) \leq OPT'(\varphi), \tag{1.72}$$

where $OPT'(\varphi)$ is the average weighted completion time of the tiny tasks of profile π in the optimal schedule. Accordingly, it also holds that

$$\sum_\varphi F_\varphi(\xi) \leq OPT', \tag{1.73}$$

where OPT' is the average weighted completion time of all tiny tasks in the optimal schedule.

LP Rounding. Let $\xi = (\xi_{jy})$ be an optimal solution of the LP(π). If ξ is integral, then for each tiny task j there is exactly one $y(j)$ such that $\xi_{y(j)j} = 1$. In other words, an integral LP solution can be treated as an integral assignment of tiny tasks of profile π to intervals. As in Lemmas 1.3.9 and 1.4.16, we can prove the following result:

Lemma 1.5.14. *For each profile π, an optimal solution ξ of the LP(π) can be rounded into an integral one such that*

(1) *$F_\pi(\xi)$ does not increase;*

(2) *each $\Delta_y(\pi)$ increases by at most $\varepsilon^2|I|_y/q^*$;*

(3) *tiny tasks j of profile π obey Smith's rule with respect to intervals $I_{y(j)}$ given by rounded ξ.*

Now we combine integral assignments ξ for all profiles π. For each tiny task j we define new index $y(j) \geq x(j)$, respectively. Then, by using $y(j)$, for each interval I_y and profile π, we define sets $\widetilde{Y}_y(\pi)$ of tiny tasks of profile π that are assigned to I_y. As in Lemmas 1.3.12 and 1.4.17, we can prove the following:

Lemma 1.5.15. *For all tiny tasks j it holds that $y(j) \geq x(j)$ and*

$$\sum w_j R_{y(j)} \;\leq\; OPT'. \tag{1.74}$$

Furthermore, for each interval I_y and profile pi it holds

$$D(\widetilde{Y}_y(\pi)) \;\leq\; D(Y_y(\pi)) \;+\; \varepsilon^2|I|_y/q^*. \tag{1.75}$$

1.5.8 Packing in Single Intervals

Here, by using the results of Lemmas 1.5.14 and 1.5.15, we complete the proof of Lemma 1.5.12.

First, we take an optimal schedule of value OPT. For an illustration see Figure 1.28. Consider an interval I_x. Each task that runs in I_x can be either tiny or huge, and each huge task can be either crossing or non-crossing, see Figure 1.28 a). There are at most $2m$ huge crossing task. Then, we can define the set K_x of huge crossing tasks, and for each profile π we define sets $N_x(\pi)$ and $Y_x(\pi)$ of non-crossing huge tasks and tiny tasks, respectively.

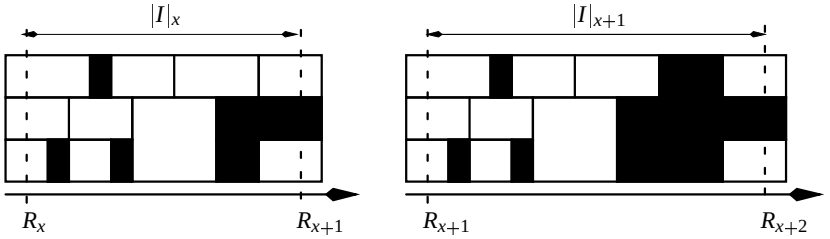


Figure 1.28: Tasks in interval I_{x+1}

Now we apply time-stretching to the optimal schedule. We add $\varepsilon|I|_x$ idle time on machines in each interval I_x, see Figure 1.28 b). This increases the objective function value by at most a factor of $1+\varepsilon$. Furthermore, the tasks of set K_x and sets $N_x(\pi)$, $Y_x(\pi)$ run in interval I_{x+1}.

Next, we use the results of Lemmas 1.5.14 and 1.5.15. For each tiny task j we find new $y(j) \geq x(j)$, and for each interval I_x we find new sets $\widetilde{Y}_x(\pi)$ of tiny tasks. We simply reschedule all tiny tasks in the schedule. Inside each interval I_{x+1} we replace the tiny tasks of sets $Y_x(\pi)$ by the tiny tasks of sets $\widetilde{Y}_x(\pi)$.

Then, each tiny task j completes at $C_j \in I_{y(j)+1}$. By Lemmas 1.5.14 and 1.5.15 we have that

$$\sum_j C_j \;\leq\; \sum_j (1+\varepsilon)\, R_{y(j)+1} \;\leq\; (1+\varepsilon)^2 OPT'. \tag{1.76}$$

In order to get a feasible schedule, we reschedule the tasks of set K_x and sets $N_x(\pi)$, $\widetilde{Y}_x(\pi)$ inside each single interval I_{x+1}. This also increases the objective function value by at most a factor of $1+\varepsilon$, and the objective function value of the final schedule is at most

$$(1+\varepsilon)^3 OPT \;\leq\; (1+7\varepsilon)OPT.$$

To complete the proof of Lemma 1.5.12, in the rest of this section we prove the following result:

Lemma 1.5.16. *Define* $q^* = \nu^*(2(m+1)!+1)^{\left\lceil \frac{2^{ms^*}}{\varepsilon^5} \right\rceil}$. *Then, the tasks of sets* $Y_x(\pi)$ *can be replaced by the tasks of sets* $\widetilde{Y}_x(\pi)$ *inside each single interval* I_{x+1}.

General Problem. In order to prove the above stated lemma we need to solve the following problem. We are given task sets K_x, $N_x(\pi)$, $Y_x(\pi)$, $\widetilde{Y}_x(\pi)$ (over all π). It is known:

(i) there exists a schedule for the tasks of set K_x and sets $N_x(\pi)$, $Y_x(\pi)$ inside interval I_x;

(ii) the length of sets $Y_x(\pi)$ and $\widetilde{Y}_x(\pi)$ differs by at most $\varepsilon^2|I|_x/q^*$;

(iii) for each task j in $Y_x(\pi)$ or $\widetilde{Y}_x(\pi)$ it holds $p_j \leq \varepsilon^2|I|_x/q^*$.

Can we define the value of parameter q^* such that there exists a schedule for the tasks of set K_x and sets $N_x(\pi)$, $\widetilde{Y}_x(\pi)$ inside interval I_{x+1}?

Simplified problem. To simplify the description, we first consider the case when the set of crossing huge tasks $K_x = \emptyset$. We also define

$$T_x(\pi) \;=\; N_x(\pi) \cup Y_x(\pi) \;\text{ and }\; T_x = \cup_\pi T_x(\pi).$$

Then, all tasks j in set $T_x(\pi)$ have the same profile $\pi = (\pi_t au)_{\tau \in 2^M}$, but differ in length p_j. Formally, each task j in $T_x(\pi)$ has processing time $\pi_{\tau j} = \pi_\tau \cdot p_j$ on the processors of $\tau \in 2^M$.

Due to (i), there exists a schedule for the tasks of sets $T_x(\pi)$ which length is at most $|I|_x$. By Lemma 1.5.7 any task crosses at most s^* intervals. Thus, the total length of the jobs in T_x is at most the total time available within s^* intervals following I_x. If $m \geq 2$ and $s^* \geq 2$ we can bound

$$D(T_x) \;\leq\; m(s^*|I|_{x+s^*}) \;\leq\; ms^*(1+\varepsilon)^{s^*}|I|_x \;\leq\; ms^*2^{s^*}|I|_x \;\leq\; 2^{ms^*}|I|_x. \quad (1.77)$$

In the following we will use the known PTAS for the makespan version of the problem [ABKM97, CM99, JP99b]. First, we deduce a schedule for the tasks of sets $T_x(\pi)$. By defining the value of parameter q^* we ensure that the schedule length is bounded by $|I|_{x+1} = (1+\varepsilon)|I|_x$. Then, we show that this bound remains valid even if we replace the tasks of $Y_x(\pi)$ by the tasks of $\widetilde{Y}_x(\pi)$ in $T_x(\pi)$. Finally, for the general case when the set of crossing huge jobs $K_x \neq \emptyset$, we change the value of q^*.

Long and Short Tasks. We take the tasks in T_x and number them by $0,1,\ldots,n'$ in order of non-increasing lengths

$$p_0 \;\geq\; p_1 \;\geq\; p_2 \;\geq\; \ldots \;\geq\; p_{n'}. \tag{1.78}$$

Here, $n'+1$ is the total number of tasks in T_x.

Next, we introduce an integer k_x^* and a constant $q^* \geq 0$. We will specify their exact values later. In addition, we introduce sets U_x, L_x and S_x as follows. We first define

$$U_x = \{\, j \,|\, j \in T_x \text{ and } p_j \geq \varepsilon^2 |I|_x / q^* \,\}. \tag{1.79}$$

Informally, we put all tasks j from set T_x with length $p_j \geq \varepsilon^2|I|_x/q^*$ into set U_x. Then, with respect to the number of tasks in U_x and the value of k_x^* we consider two cases:

(1) If $|U_x| \leq k_x^*$ then we put $L_x := U_x$ and $S_x := T_x \setminus U_x$;

(2) If $|U_x| > k_x^*$ then we put the first k_x^* tasks $0,1,\ldots,k_x^*-1$ into set L_x, and other tasks $k_x^*, k_x^*+1,\ldots,n'$ into set S_x.

For simplicity, we call the tasks of L_x *long* and the tasks of S_x *short*. Finally, for each profile π we define set

$$S_x(\pi) \;=\; S_x \cap T_x(\pi).$$

Notice that in both cases (1) and (2) it holds that

$$L_x \subseteq U_x \;\text{ and }\; S_x \;=\; J_x \setminus L_x. \tag{1.80}$$

Furthermore, the number of long tasks

$$|L_x| \;:=\; \min\{\, k_x^*, |U_x| \,\}, \tag{1.81}$$

i.e. either $U_x = L_x$ or the first k_x^* longest tasks from U_x are in L_x.

Snapshots and Relative Schedules. A *processor assignment* for the long tasks in L_x is a mapping $\tau : L_x \to 2^M$ which defines the allotment $\tau(j)$ for each long task $j \in L_x$. Two long tasks k and j are called *compatible*, if $\tau(k) \cap \tau(j) = \emptyset$. A *snapshot* is a set of compatible tasks. Given a processor assignment, a *relative schedule* is a sequence of snapshots $M(1),\ldots,M(g)$, such that

- each long task j from L_x occurs in a subsequence of consecutive snapshots $M(u_j),\ldots,M(v_j)$, $1 \le u_j \le v_j \le g$, and
- any two consecutive snapshots are different.

For an illustration see Figure 1.29. The number of snapshots

$$g \;\le\; 2|L_x|. \tag{1.82}$$

Roughly speaking, each relative schedule corresponds to an order of processing of the long tasks. To any given non-preemptive schedule of L_x, one can associate a relative schedule in a natural way by looking at every instant where a task of L_x starts or ends, and writing the set of tasks of being processed with their allotments right after that transition.

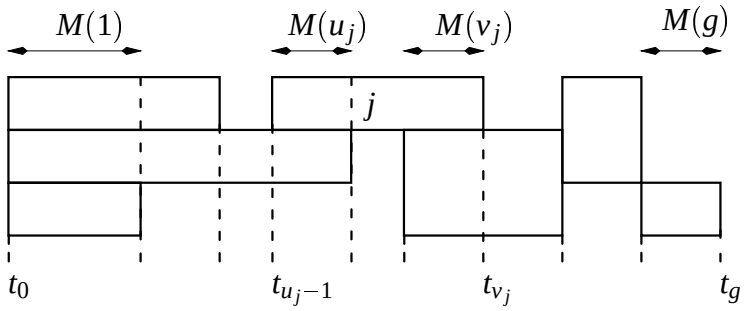


Figure 1.29: Long task j in a relative schedule

Configurations and Free Processors. Given a set $\mu \subseteq M$, a *μ-configuration* $C(\mu)$ is a partition of μ into non-empty sets. Let $N(\mu)$ be the total number of μ-configurations and let $C(\mu)_1,\ldots,C(\mu)_{N(\mu)}$ be all μ-configurations. Notice that

$$N(\mu) \;\le\; N(M) \;=\; B(m) \;\le\; m!, \tag{1.83}$$

where $B(m)$ is the mth Bell's number [Knu68].

For a fixed processor assignment of L_x, $\tau : L_x \to 2^M$, we consider a relative schedule, $M(1),\ldots,M(g)$. Then,

$$F(s) := M \setminus \bigcup_{j \in M(s)} \tau(j)$$

is the set of *free processors* in snapshot $M(s)$, $s = 1,\ldots,g$. For each such set $F(s)$ we have $F(s)$-configurations $C(F(s))_i$, $i = 1,\ldots,N(F(s))$.

LP Formulation. Assume that we have a processor assignment for the long tasks in L_x, $\tau : L_x \to 2^M$. Let $M(1),\ldots,M(g)$ be a relative schedule of L_x with respect to this assignment.

Assume that we have scheduled the long tasks of L_x as in $M(1),\ldots,M(g)$. This creates a structure where each snapshot $M(s)$ represent an interval in which free processors in $F(s)$ do not process long tasks. For an illustration see Figure 1.29 on the preceding page. Our next goal is to use these free processors for executing all short tasks in $S_x = \cup_\pi S_x(\pi)$.

Recall that all tasks j in $S_x(\pi)$ have the same profile $\pi = (\pi_\tau)_{\tau\in 2^M}$, but differ in length p_j. Thus, to execute all tasks in $S_x(\pi)$ on a processor set $\tau \in 2^M$ we need $p_\tau D(S_x(\pi)) = p_\tau \sum_{j\in S_x(\pi)} p_j$ total processing time.

Here, we can formulate the relaxed problem as the following LP:

$$\begin{aligned}
\textbf{Minimize}\quad & t_g \\
\textbf{s.t.}\ \textbf{(0)}\quad & t_0 = 0, \\
\textbf{(1)}\quad & t_s \;\geq\; t_{s-1}, \ \text{for } s = 1,\ldots,g, \\
\textbf{(2)}\quad & t_{v_j} - t_{u_j-1} \;\geq\; p_{\tau(j)j}, \ \text{for all } j \in L_x, \\
\textbf{(3)}\quad & \textstyle\sum_{i=1}^{N(F(s))} x_{i,s} \;=\; t_s - t_{s-1}, \ \text{for } s = 1,\ldots,g, \\
\textbf{(4)}\quad & \textstyle\sum_{s=1}^{g} \sum_{i:\tau\in C(F(s))_i} x_{i,s} \;\geq\; \sum_\pi \pi_\tau \cdot D(S_x(\pi)) \cdot y_{\tau,\pi}, \ \text{for all } \tau \in 2^M, \\
\textbf{(5)}\quad & x_{i,s} \;\geq\; 0, \ \text{for } s = 1,\ldots,g,\ i = 1,\ldots,N(F(s)), \\
\textbf{(6)}\quad & \textstyle\sum_\tau y_{\tau,\pi} \;=\; 1, \ \text{for all } \pi, \\
\textbf{(7)}\quad & y_{\tau,\pi} \;\geq\; 0, \ \text{for all } \pi,
\end{aligned}$$

where the variables have the following interpretation:

t_s**:** a time point at which snapshot $M(s)$ ends and $M(s+1)$ starts; here, $t_0 = 0$ and t_g is the starting time and the finishing time of the schedule, see Figure 1.29 on the facing page;

$x_{i,s}$**:** the length of the ith $F(s)$-configuration $C(F(s))_i$ in snapshot $M(s)$; here, the short tasks of S_x can be executed on processor sets $\tau \in C(F(s))_i$ during an interval of length $x_{i,s}$ inside $M(s)$, see Figure 1.31 on page 92;

$y_{\tau,\pi}$**:** the τth fraction of tasks in $S_x(\pi)$ allotted to a processor set $\tau \in 2^M$; Here, for each profile $\pi = (\pi)_{\tau \in 2^M}$, $\pi_\tau \cdot D(S_x(\pi))$ is the total processing time of the tasks in $S_x(\pi)$ allotted to a processor set $\tau \in 2^M$.

The constants have the following meaning. Constraints **(0)-(2)** define a schedule for the large tasks in L_x with respect to snapshots $M(1), M(2), \ldots, M(g)$. Constraints **(3)** define a structure for all $F(s)$-configurations $C(F(s))_i$ inside each snapshot $M(s)$. Constraints **(4)**, for each processor set $\tau \in 2^M$, define a balance between all $x_{i,s}$ configurations $C(F(s))_i$ which have the processors of τ and all $y_{\tau,\pi}$th fractions of $S_x(\pi)$ allotted to the processors of τ. Constraints **(6)-(7)** define that each set $S_x(\pi)$ is allotted completely to sets $\tau \in 2^M$.

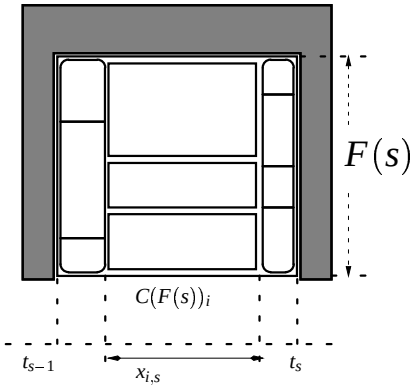


Figure 1.30: Inside a snapshot

Proposition 1.5.17. *There exists a processor assignment and a relative schedule of the long tasks in L_x such that $t_g \leq |I|_x$. Furthermore, if each $D(S_x(\pi))$ increases by at most $\varepsilon^2|I|_x/2\nu^*$, then t_g increases by at most $\varepsilon^2|I|_x/2$, i.e. $t_g \leq |I|_x + \varepsilon^2|I|_x/2$.*

Proof. Recall that there exists a schedule for the tasks of T_x in interval I_x. We just "copy" the processor assignment and the relative schedule for the tasks in L_x. Due to the LP formulation, there exists a solution (t,x,y) of the LP such that $t_g \leq |I|_x$.

Let us fix such these processor assignment and relative schedule. Take (t,x,y). Then, from **(3)** of the LP we have

$$t_g = \sum_{s=1}^{g}(t_s - t_{s-1}) = \sum_{s=1}^{g}\sum_{i=1}^{N(F(s))} x_{i,s}. \tag{1.84}$$

Now we increase each $D(S(\pi)_x)$ by $\varepsilon^2|I|_x/2\nu^*$. How much do we increase t_g solving the LP?

Fix the values of y. Then, we find a minimal increase $\Delta_{i,s}$ in each $x_{i,s} \neq 0$ such that **(0)-(5)** of the LP are satisfied. (We also find the corresponding increase in each $t_s \neq 0$.) Let $\tilde{x} = x + \Delta$ and $\tilde{t}$ be the new found values of x and t, respectively. Then, from **(3)** of the LP we have

$$\tilde{t}_g = \sum_{s=1}^{g} \sum_{i=1}^{N(F(s))} \tilde{x}_{i,s}. \tag{1.85}$$

We increase the length of each $D(S_x(\pi)$ by $\varepsilon^2|I|_x/2\nu^*$. Thus, in **(4)** of the LP we have new total processing times

$$\begin{aligned} \tilde{D}_\tau(\pi) :=\ & \pi_\tau \cdot (D(S_x(\pi)\ +\ \varepsilon^2|I|_x/2h^*\nu^*) \\ =\ & \pi_\tau \cdot D(S_x(\pi))\ +\ \pi_\tau\varepsilon^2|I|_x/2\nu^*. \end{aligned} \tag{1.86}$$

By the definition of profile $\pi = (\pi_\tau)_{\tau \in 2^M}$ it holds that all $\pi_\tau \in \{1, \infty\}$. Thus,

$$\pi_\tau\varepsilon^2|I|_x/\nu^*\ \leq\ \varepsilon^2|I|_x/2\nu^*. \tag{1.87}$$

Look at **(3), (4)** of the LP. Here $\tilde{x} = x + \Delta$ and each $\Delta_{i,s}$ in a minimal increase in $x_{i,s} \neq 0$ under fixed y. Hence, for each τ in 2^M there are two cases: (a) **(4)** holds on τ as an exact equality, and (b) **(4)** holds on τ as a strong inequality.

Consider a processor set $\bar{\tau}$ in case (a). Then, for $\tilde{x}$ we have

$$\begin{aligned} \sum_{s=1}^{g} \left(\sum_{i:\bar{\tau} \in C(F(s))_i} \tilde{x}_{i,s} \right)\ =\ & \sum_{\pi} \tilde{D}_{\bar{\tau}}(\pi) \cdot y_{\bar{\tau},\pi} \\ \leq\ & \sum_{\pi} (\pi_{\bar{\tau}} \cdot D(S_x(\pi)\ +\ \varepsilon^2|I|_x/2\nu^*). \end{aligned} \tag{1.88}$$

Remember that for (t,x,y) in **(4)** of the LP it holds

$$\sum_{s=1}^{g} \left(\sum_{i:\bar{\tau} \in C(F(s))_i} x_{i,s} \right)\ =\ \sum_{\pi} \pi_{\bar{\tau}} D(S_x(\pi)) \cdot y_{\bar{\tau},\pi}.$$

Thus, from $\tilde{x}_{i,s} = x_{i,s} + \Delta_{i,s}$ we have

$$\sum_{s=1}^{g} \left(\sum_{i:\bar{\tau} \in C(F(s))_i} \Delta_{i,s} \right)\ \leq\ \varepsilon^2|I|_x/2\nu^* \sum_{\pi} y_{\bar{\tau},\pi}. \tag{1.89}$$

Recall that the values of $\Delta_{i,s}$ are minimal. Hence, each $\Delta_{i,s} \neq 0$ occurs at least once in (1.89) while we run through such processor subset $\bar{\tau}$ in case (a). Thus, combining we have

$$\begin{aligned}
\tilde{t}_g - t_g &= \sum_{s=1}^{g} \left(\sum_{i=1}^{N(F(s))} \Delta_{i,s} \right) \\
&\leq \sum_{\bar{\tau}} \sum_{s=1}^{g} \left(\sum_{i:\bar{\tau} \in C(F(s))_i} \Delta_{i,s} \right) \\
&\leq \sum_{\bar{\tau}} \left(\varepsilon^2 |I|_x / 2\nu^* \sum_{\pi} y_{\bar{\tau},\pi} \right) \\
&\leq \varepsilon^2 |I|_x / 2\nu^* \left(\sum_{\pi} \sum_{\bar{\tau}} y_{\bar{\tau},\pi} \right).
\end{aligned} \tag{1.90}$$

From **(6)** of the LP we have

$$\sum_{\bar{\tau}} y_{\bar{\tau},\pi} \leq \sum_{\tau \in 2^M} y_{\tau,\pi} = 1,$$

and by Lemma 1.5.9 we have

$$\sum_{\pi} \sum_{\bar{\tau}} y_{\bar{\tau},\pi} \leq \nu^*.$$

Thus, we get

$$\tilde{t}_g - t_g \leq \varepsilon^2 |I|_x / 2. \tag{1.91}$$

□

Allotment to Processors: Unbalanced Short Tasks. Take a processor assignment for L_x and a relative schedule with snapshots $M(1), M(2), \ldots, M(g)$ as in Proposition 1.5.17. Let (t, x, y) be an optimal solution of the LP with the objective value $t_g \leq |I|_x$.

Next, we fix the values of t and x. Look at y in **(4)** and **(6)** of the LP. There are 2^m common constraints on all $y_{\tau,\pi}$. (See Appendix B on page 197.) Hence, we can reassign the values of y such that there are at most 2^m distinct (τ, π) for which

$1 > y_{\tau,\pi} > 0$ (one τ can correspond to several profiles π), and for all the other either $y_{\tau,\pi} = 1$ or $y_{\tau,\pi} = 0$ ($y_{\tau,\pi} = 1$ only for one $\tau \in 2^M$).

First, we allot the tasks in L_x as it is defined in the processor assignment of L_x. Next, we allot the short tasks of S_x to processors sets $\tau \in 2^M$ as follows. For each profile π we allot the tasks of $S(\pi)_x$ with respect to the values of $(y_{\tau,\pi})$, $\tau \in 2^M$. If $y_{\tau,\pi} = 1$, then we allot all tasks j from $S(\pi)_x$ to the processors of τ. If $1 > y_{\tau,\pi} > 0$, we select jobs j from $S(\pi)_x$ in a greedy manner until their total length is less than

$$D(S(\pi)_x) \cdot y_{\tau,\pi},$$

and then we allot all selected tasks to the processors of τ. Here, we also allot the last selected task j, which can cause the exceed of value $D(S(\pi)_x) \cdot y_{\tau,\pi}$, marking it as *unbalanced*. Since there are at most 2^m distinct (τ,π) for which $1 > y_{\tau,\pi} > 0$, there are at most 2^m such unbalanced tasks in S_x.

Deducing a Schedule. Now we want to deduce a schedule for the tasks in L_x and S_x. We use snapshots $M(1), M(2), \ldots, M(g)$, solution (t,x,y), and allotments for L_x and L_x as it is defined above.

First, we schedule the tasks in L_x with respect to snapshots $M(1), M(2), \ldots, M(g)$ and the values of t. Due to the LP formulation, it gives a feasible schedule.

Next, we schedule the short tasks of S_x on the free processors in $F(s)$ inside snapshots $M(s)$, $s = 1, \ldots, g$, as follows. For $s = 1, \ldots, g$ we select configurations $C(F(s))_i$ with $x_{i,s} > 0$, $i = 1, \ldots, N(F(s))$. Then, for each $\tau \in C(F(s))_i$ we select the tasks in S_x allotted to τ (except the unbalanced ones) in a greedy manner until their total length does not exceed $x_{i,s} > 0$. Then, we schedule these selected tasks on the processors of τ inside an interval of length $x_{i,s} > 0$ inside snapshot $M(s)$. (For an illustration see Figure 1.30 on page 88.) If it happens that for some $\tau \in C(F(s))_i$ the selected tasks do not fit inside interval $x_{i,s} > 0$, we increase its length by a small value $\Delta_{i,s}$. At the end of the procedure, we assign the unbalanced tasks in a similar way.

To remain correct, we always adjust the values of t_s such that $t_s - t_{s-1}$ ($s = 1, \ldots, g$) increases by

$$\Delta(s) \;=\; \sum_{i=1}^{N(F(s))} \Delta_{i,s},$$

called the *snapshot enlargement*. (See Figure 1.31 on the next page.) Then, the length of the deduced schedule is at most

$$\sum_{s=1}^{g} (t_s - t_{s-1}) \;=\; t_g \;+\; \sum_{s=1}^{g} \Delta(s) \;=\; t_g \;+\; \Delta_x, \tag{1.92}$$

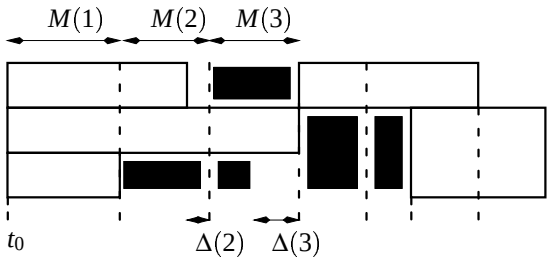


Figure 1.31: Deducing a schedule

where

$$\Delta_x \;=\; \sum_{s=1}^{g} \Delta(s) \tag{1.93}$$

is the the total *schedule enlargement*.

Defining the values of k_x^* and q_2^*. Here, our goal is to define the values of k_x^* and q^* such that

$$\Delta_x \;\leq\; \varepsilon^2 |I|_x / 2. \tag{1.94}$$

Remember the allotment procedure. We add $\Delta_{i,s}$ to $x_{i,s}$ only in two cases. In the first case we accommodate a short task which cannot fit into an interval of length $x_{i,s} > 0$. In the second case we accommodate an unbalanced short task.

There are at most 2^m unbalanced tasks. From (1.82) and (1.83), there are at most $2|L_x|B(m)$ intervals. In total, we add $\Delta_{i,s}$ to $x_{i,s}$ for accommodating at most

$$2|L_x|B(m) + 2^m \leq 2^{m+1}(|L_x|B(m))$$

tasks from S_x. Now we use (1.78) and (1.80). By (1.93), we can bound the total schedule enlargement as follows

$$\Delta_x \;\leq\; 2^{m+1}(p_{|L_x|} \;+\; p_{|L_x|\,+\,1} + \cdots p_{|L_x|\,+\,|L_x|B(m)-1}), \tag{1.95}$$

i.e. 2^{m+1} times the total length of the $|L_x|B(m)$ longest tasks in S_x.

Next we use the following result.

Proposition 1.5.18 ([CM99, JP99b]). *Suppose $p_0 \geq p_1 \geq \ldots \geq p_{n'} > 0$ is a sequence of real numbers and $P \geq \sum_{j=0}^{n'} p_j$. Let β be a nonnegative integer, $\alpha > 0$, and assume that n' is sufficient large (i.e. all the indices of the p_j's in the statement are smaller than n'; e.g. $n' > (\beta+1)^{\lceil \frac{1}{\alpha} \rceil}$ suffices). Then, there exists an integer $k = k(\beta,\alpha)$ such that $p_k + \ldots + p_{k+\beta k-1} \leq \alpha \cdot P$, and $k \leq (\beta+1)^{\lceil \frac{1}{\alpha} \rceil - 1}$.*

From (1.77) and (1.78), we can find an upper bound P on $\sum_{j=0}^{n'} p_j$. We simply define

$$P = 2^{ms^*}|I|_x, \quad \alpha = \varepsilon^2/(22^{m+1}2^{ms^*}) \quad \text{and} \quad \beta = B(m). \tag{1.96}$$

Then, by Proposition 1.5.18 we can define

$$k_x^* := k(\beta,\alpha) \leq (B(m)+1)^{\left\lceil \frac{2^{ms^*+m+2}}{\varepsilon^2} \right\lceil - 1} \tag{1.97}$$

such that

$$2^{m+1}(p_{k_x^*} + p_{k_x^*+1} + \cdots p_{k_x^*+k_x^*B(m)-1}) \leq \varepsilon^2 |I|_x/2. \tag{1.98}$$

In the following we define the value of q^*. Consider case (1). We have $|L_x| = k_x^*$ and $|U_x| > k_x^*$. Then, $\Delta_x \leq \varepsilon^2|I|_x/2$ holds by (1.95) and (1.98). Consider case (2). We have $L_x = U_x$, $S_x = J_x \setminus U_x$. Then, $|L_x| = |U_x| \leq k_x^*$ for each task j in set S_x it holds $p_j < \varepsilon^2|I|_x/q^*$. Furthermore, by (1.98) we have

$$|L_x| \leq k_x^* \leq (B(m)+1)^{\left\lceil \frac{2^{ms^*+m+2}}{\varepsilon^2} \right\rceil - 1}. \tag{1.99}$$

Then, by (1.95) we have

$$\begin{aligned}
\Delta_x &\leq 2^{m+1}(p_{|L_x|} + p_{|L_x|+1} + \cdots p_{|L_x|+|L_x|B(m)-1}) \\
&\leq 2^{m+1}|L_x|B(m)\left(\frac{\varepsilon^2|I|_x}{q^*}\right) \\
&< |L_x|(B(m)+1)^{2^{m+1}}\left(\frac{\varepsilon^2|I|_x}{q^*}\right).
\end{aligned} \tag{1.100}$$

For $2^m \leq 1/\varepsilon$ and $m \geq 2$ we define

$$\begin{aligned}
q^* &= \nu^* \cdot (2m!+1)^{\left\lceil \frac{2^{ms^*}}{\varepsilon^5} \right\rceil} \\
&\geq (2m!+1)^{\left\lceil \frac{2^{ms^*+3m}}{\varepsilon^2} \right\rceil} \\
&\geq (2B(m)+1)^{\left\lceil \frac{2^{ms^*+m+2}}{\varepsilon^2} \right\rceil + 2^{m+1}-1} \\
&\gg 2\nu^*.
\end{aligned} \tag{1.101}$$

Then, from (1.99) and (1.100) we have $\Delta_x \leq \varepsilon^2 |I|_x/2$.

Solving the Simplified Problem. One can see that we have deduced a schedule for the tasks of sets $T_x(\pi) = \cup N_x(\pi) \cup Y_x(\pi)$. From (1.92) and (1.94) the length of the deduced schedule is at most

$$t_g + \Delta_x \leq t_g + \varepsilon^2 |I|_x/2 \leq |I|_x + \varepsilon^2 |I|_x/2 \leq (1+\varepsilon)|I|_x = |I|_{x+1}. \quad (1.102)$$

Furthermore, even if we replace the tiny tasks of $Y_x(\pi)$ by the tiny tasks of $Y_x(\pi)$ in each set $T_x(\pi)$, we will be able to deduce a schedule for the tasks of sets $T_x(\pi)$. Indeed, all new tasks from $T_x(\pi)$ will fall into set $S_x(\pi)$ of short tasks. (See the definition of set S_x and sets $S_x(\pi)$.) Due to (ii) and (iii) in the definition of general problem, and (1.101), this increases the value of $D(S_x^{\varphi})$ by at most

$$\varepsilon^2 |I|_x/q^* < \varepsilon^2 |I|_x/2\nu^*.$$

By Proposition 1.5.17, the objective function value t_g increases by at most $\varepsilon^2|I|_x/2$, i.e.

$$t_g \leq |I|_x + \varepsilon^2 |I|_x/2.$$

Thus, by following the same line of ideas, we can deduce a schedule which length is bounded by

$$t_g + \varepsilon^2 |I|_x/2 + \varepsilon^2 |I|_x/2 < (1+\varepsilon)|I|_x = |I|_{x+1}. \quad (1.103)$$

Solving the General Problem. Clearly, we can also follow all above procedures in the case when the set of crossing huge tasks $K_x \neq \emptyset$. However, we need to adjust the values of k_x^* and q^*. Furthermore, we need to adjust the processing times of the crossing tasks, see Figure 1.28 b) on 83.

We simply add all tasks in K_x into set L_x of long tasks. As we discussed, there are at most $2m$ crossing tasks. Hence, $|K_x| \leq 2m$ and the number of snapshots in any relative schedule can be bounded as follows

$$g \leq 2|L_x| \leq 2(|L_x \setminus K_x| + 2m) \leq 4m|L_x \setminus K_x|.$$

Then, in (1.96) β changes to $4mB(m)$ and we have

$$|L_x \setminus K_x| \leq k_x^* = k(\beta, \alpha) \leq (B(m)+1)^{\left\lceil \frac{2ms^*+m+2}{\varepsilon^2} \right\rceil - 1}.$$

In (1.100) we also get

$$\Delta_x \leq 2^{m+1}(2mB(m)|L_x \setminus K_x|)\left(\varepsilon^2 |I|_x/q^*\right)$$

$$\leq |L_x \setminus K_x|(2mB(m)+1)^{2^{m+1}+1}\left(\varepsilon^2 |I|_x/q^*\right).$$

Similarly, we can define

$$\begin{aligned} q^* &= \nu^*(2(m+1)!+1)^{\left\lceil \frac{2ms^*}{\varepsilon^5} \right\rceil} \\ &> \nu^*(2mB(m)+1)^{\left\lceil \frac{2ms^*+m+2}{\varepsilon^2} \right\rceil + 2^{m+1}-1} \\ &\gg 2\nu^*. \end{aligned}$$

Hence, we can claim (1.103).

In total, we can deduce a schedule for the tasks of set K_x and sets $N_x(\pi)$, $\widetilde{Y}_x(\pi)$ which will fit inside interval I_{x+1}. This completes the proof of Lemma 1.5.16.

1.5.9 Weight-Shifting and Merging

After we have proved the modified Smith's rule for tiny jobs, the compacting step of our algorithm is quite simple. We can follow the same line of ideas as in one machine case $1|r_j|\sum w_jC_j$. We simply use weight-shifting and merging described in Sections 1.3.7 and 1.3.8, respectively.

Weight-Shifting. Assume that there are a lot of tasks at some R_x. Which tasks can wait until the next interval? Consider huge tasks HT_x and tiny tasks TT_x. Take one profile π. The tasks of $HT_x(\pi)$ having the same size must complete by decreasing weight. By Lemma 1.5.10 there is at most a constant number of such sizes. The tasks of $TT_x(\pi)$ must complete by decreasing ratio w_j/p_j. We select only the tasks that can be potentially scheduled in I_x.

Lemma 1.5.19. *With $1+O(\varepsilon)$ loss and in $O(n\log n)$ time, we can enforce that $D(TT_x) \le t^*|I|_x$ and $|HT_x| \le H^*$ at each release date R_x, where t^* and H^* are some constant.*

Merging. At each release date we partition the ordered set of tiny jobs $TT_x(\pi)$ into sets of roughly equal length $\approx \varepsilon^2|I|_x/2q^*$ (but less than $\varepsilon^2|I|_x/q^*$). Each such set creates a new tiny job with the weight equal to the total weight of the set.

Lemma 1.5.20. *With $1+O(\varepsilon)$ loss and in $O(n\log n)$ time, we can enforce that $|TT_x| \le T^*$ and $HT_x| \le H^*$ at each release date R_x, where T^* and H^* are some constant.*

1.5.10 Blocks and Dynamic Programming

Here, we first use the result of Lemma 1.5.8 and then we follow the known ideas of Lemmas 1.3.8,1.3.14. We can consider only near-optimal schedules where every job to be completed within a constant number of intervals after its release date.

Lemma 1.5.21. *With $1+O(\varepsilon)$ loss, we can assume that each job completes within d^* intervals after its release, where d^* is some constant.*

Similarly to Section 1.3.9, we can define blocks, and then use the dynamic programming framework presented in Section 1.3.10. The combination of the structuring and compacting steps with a dynamic programming algorithm gives a PTAS with $O(n\log n)$ running time. This completes the proof of Theorem 1.5.1.

1.5.11 Extension to General Multiprocessor Tasks

Let us consider the problem of scheduling general multiprocessor tasks. We are given a set of n tasks $T=\{1,2,\ldots,n\}$ and a set $M=\{1,\ldots,m\}$ of m processors. Each task $j\in T$ has a positive weight w_j, a release date r_j, and an associated function $p_{\cdot j}: 2^M \longrightarrow \mathbb{Z}^+\cup\{+\infty\}$ that gives the processing time $p_{\tau j}$ of task j in terms of the set of processors $\tau\subseteq M$ that are *allotted* to j. (Here 2^M denotes the set of all non-empty subsets of M.) Each processor can work on at most one task at a time, and given an *allotment* $\tau(j)\subseteq M$ for a task $j\in T$, the processors of $\tau(j)$ are required to execute task j in union during $p_{\tau(j)j}$ time units. Here we assume that m is fixed, there are no restrictions on $p_{\cdot j}$, and the goal is to find a task allotment and a non-preemptive feasible schedule under this allotment such that $\sum w_jC_j$ is minimized.

Scheduling without Release dates. Here we assume that there are no release dates, i.e. all $r_j=0$. For each task $j\in T$ we also define the *length* $p_j=\min_{\tau\in 2^M} p_{\tau j}$.

We first follow the same line of ideas as in Sections 1.5.1. We create a well-structured set of task processing times and weights.

Lemma 1.5.22. *With $1+\varepsilon$ loss and in $O(n)$ time , we can enforce all $p_{\tau j}$, $\tau\in 2^m$, be integer powers of $1+\varepsilon$.*

Lemma 1.5.23. *With $1+\varepsilon$ loss and in time $O(n)$, we can enforce all w_j/p_j be distinct.*

Lemma 1.5.24. *With $1+\varepsilon$ loss, we can assume that all $S_j\geq \varepsilon p_{\tau(j)j}$ in a schedule.*

Lemma 1.5.25. *With $1+\varepsilon$ loss, we can assume that any task crosses at most $s^* = \lceil \log_{1+\varepsilon}(1+\frac{1}{\varepsilon}) \rceil$ intervals in a schedule.*

There are no changes in Section 1.5.3. However, we will need the following result.

Lemma 1.5.26. *With $1+O(\varepsilon)$ loss, in any schedule we can assume all allotments $\tau(j)$ be such that $p_{\tau(j)j} \leq h^* \cdot p_j$, where $p_j = \min_\tau p_{\tau j}$ is the length of task j and h^* is some common constant.*

Proof. Take a feasible schedule of value OPT. By Lemma 1.5.8, there are at most $4s^*/\varepsilon$ intervals between two consecutive gaps. If there is a gap in I_x, the total time available in $4s^*/\varepsilon$ next intervals can be estimated as follows

$$mI_x(1+\varepsilon)^{4s^*/\varepsilon} \leq h^*(\varepsilon^2 I_x/2^m),$$

where h^* is some constant. If there are some tasks j with $p_{\tau(j)j} > h^* \cdot p_j$ in these intervals, then running them on the fastest processor subsets requires at most $\varepsilon^2 I_x$ idle time. (There are at most 2^m subsets.) By using time-stretching we create εI_x idle time in each gap, and reschedule the tasks. The objective value of the final schedule is at most $(1+6\varepsilon)(1+\varepsilon)OPT \leq (1+14\varepsilon)OPT$. □

Profiles. For a task j we can define $p_{\tau j} = \infty$ whenever $p_{\tau j} > h^* \cdot p_j$. Now we define the *profile* of j to be an $|2^M|$-tuple $\pi = (\pi_\tau)_{\tau \in 2^M}$ such that $p_{\tau j} = p_j \cdot (1+\varepsilon)^{\pi_\tau}$. We adopt the conversion that $\pi_\tau = \infty$ if $p_{\tau j} = \infty$.

Lemma 1.5.27. *The number of distinct profiles is bounded by a constant ν^*.*

Release Index. By Lemma 1.5.24, we can assume that any task j starts later εp_j. We introduce an index $x(j)$ for each task j that corresponds to the interval $I_{x(j)}$ earlier which processing of j can not be started. By Lemma 1.5.24, we put $R_{x(j)} \leq \varepsilon p_j \leq R_{x(j)+1}$, and call $x(j)$ the *release index* of j.

Huge and Tiny Tasks. We say that task j is *huge* in an interval I_x if $p_j \geq \varepsilon^2 |I|_x/q^*$, and *tiny* otherwise. Then, HT_x and TT_x are sets of huge and tiny tasks j with $x(j) = x$.

A PTAS. Now we have profiles π and release indices $x(j)$. Furthermore, we have a bound h^* on the difference between length p_j and all processing times $p_{\tau j}$. By adjusting the value of parameter q^* and using a PTAS for the makespan version of the problem [JP99a], we can prove Smith's rule. Then, we can proceed with the same procedures in compacting and dynamic programming. As for the parallel and dedicated case, this gives a PTAS with $O(n \log n)$ running time.

Scheduling with Release Dates. As we saw, rounding of general multiprocessor tasks without release dates is quite simple. However, as soon as release dates are introduced, the problem becomes more complex. It can happen that some tasks cross a release date making some processors busy for next several intervals. Then, a new released task should either wait until the fastest processors are free or try to run on some other ones which, it can happen, are much slower. Thus, the bound in Lemma 1.5.26 and the definition of profiles do not work.

From another side, by Lemma 1.5.8 there is a gap within any sequence of $4s/\varepsilon$ consecutive intervals. Let $x(j)$ and $y(j)$ be whose indices for which $r_j \in R_{x(j)}$ and $S_j \in R_{y(j)}$. Then, we can reformulate the result of Lemma 1.5.26 as follows

Lemma 1.5.28. *With $1+\varepsilon$ loss, in any schedule we can that for each task j that either the processing time $p_{\tau(j)j} \leq h^* \cdot p_j$ or $y(j) \leq x(j) + 4s^*/\varepsilon$.*

We note that the special case $p_{\tau(j)j} > h^* \cdot p_j$ occurs when, at time $r_j = R_{x(j)}$, the processors of $\tau(j)$ are busy processing a crossing task, whose processing time spans all of the interval $I_{x(j)}$ and beyond. Thus, if we know the schedule of the crossing tasks, we can, for each task j, replace p_j by

$$\min\{p_{\tau j} | \text{processors of } \tau \text{ are not busy at time } r_j \text{ with a crossing task spanning all of } I_x\}.$$

Since we do not know ahead of time the schedule of the crossing tasks, the dynamic program will have to perform these updates dynamically.

Furthermore, there is a need for a more suitable definition of huge and tiny task. Here, we have to add the task processor allotment and the length of an interval where the task is scheduled. In this case we have to "guess" a lot of information before we can deduce a schedule. Hence, we can conclude that there is a need for a special a number of new ideas which would replace Smith's rule, the weight-shifting technique, and the merging technique.

1.6 CONCLUDING REMARKS

With this work we continue a series of recent papers on approximation algorithms for scheduling problems with the average weighted completion time objective; see, in particular, [ABC+99, CPS+96, Che98, HSSW97, Sch96, Shm98, Sku98]. On one hand, one can use several rounding techniques which transform optimal solutions to LP relaxations to near-optimal solutions. This idea leads to many good practical approximation algorithms. On the other hand, one can use the input transformation technique and the weight-shifting technique. This idea leads

to polynomial time approximation schemes (PTASs): algorithms that compute $(1+\varepsilon)$-solutions in polynomial time, but exponential in $1/\varepsilon$.

Thereby, the results obtained show that a combination of both these ideas is a quite powerful method for designing PTASs. They also prove the strength of PTASs for the makespan version of the problem. In fact, if we consider the problem of scheduling on a single machine, there is no need for some special techniques. However, if we consider the job shop problem or the multiprocessor scheduling problem, we cannot avoid using PTASs for the makespan versions. One of the interesting aspects here is that we do not really use it as a subroutine, but rather as a part of our proof technique which includes LP formulations, rounding of optimal LP solutions to integer assignments, and scheduling jobs (tasks) inside single intervals. This fits nicely with the observation that in the single machine problem Smith's rule for tiny jobs can be simply obtained by rounding, see Sections 1.3.4 and 1.3.5,1.3.6.

We improve and generalize a number of results presented earlier in [CPS$^+$96, CLL98, ABC$^+$99, ABKM00, ABF$^+$00, FJP01b]. However, many interesting questions remain. Though we have obtained PTASs for the problems with a fixed number of machines m, the running time of these algorithms is impractical. Basically, we depend on the running time of PTASs for the makespan versions and the running time of the enumeration procedure in dynamic programming. Although desirable, we do not believe that a running time of the form $f(m)\cdot \text{poly}(n^{1/\varepsilon})$, for a $(1+\varepsilon)$-approximation, is achievable. Regarding the job shop scheduling problem, we considered the case when the number of operations per job μ is a fixed constant. Indeed, one of the most interesting questions is extending of our results for an arbitrary (not fixed) μ. However, to our best knowledge, there is no PTAS known for the corresponding makespan variant of the problem, and we rather expect the discovery of some inapproximability results. For future results, it would be interesting if new more powerful technique be developed and our conjecture be proven.

CHAPTER 2

DISTANCE CONSTRAINED LABELING OF DISK GRAPHS

2.1 INTRODUCTION

The *frequency assignment* problem (FAP) addresses questions pertinent to the point-to-point communication in *radio/mobile telephony* networks. It asks for an assignment of frequencies to transmitters so as to avoid interference in a situation when signals of similar frequencies are used in the same location. It also asks that the assignment should use as less frequencies as possible.

Another important aspect of the FAP concerns the fact that the number of transmitters in modern communication networks increases. This means that the corresponding transmitter systems should be flexible to possible changes, e.g. installing or disinstalling of a group of transmitters. Hence, an assignment of frequencies to new transmitters should not lead to major changes in the already existing transmitter system, and should not decrease the quality of communication. Thus, taking into account all these features, one needs to consider the FAP as an on-line problem.

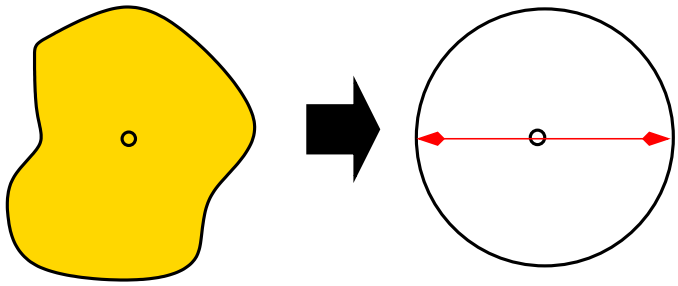


Figure 2.1: Transmitter - Disk

The most common model for an instance of the FAP is the *interference graph*. Each vertex of the interference graph represents a transmitter. If simultaneous

broadcasting by two transmitters may cause an interference, then they are connected by an edge in the interference graph. Often interference graphs are assumed to have a special structure, e.g. planar graphs, or grids [vdHLS98].

Alternatively, one can associate the coverage area of a transmitter with a disk of a particular diameter. Then, if two disks intersect, the vertices are connected by an edge in the interference graph . In this case, the underlying interference graph is a *disk graph*, that is the intersection graph of disks in the plane.

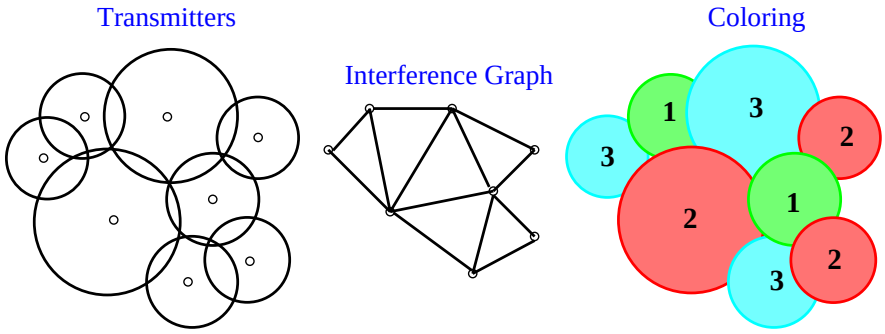


Figure 2.2: FAP – Coloring

Regarding the assumption that a pair of "close" transmitters should be assigned different frequencies, the FAP is equivalent to the problem of *coloring* the interference graph. (See Figure 2.2.)

Coloring. A (vertex) k-coloring of a graph $G = (V,E)$ is a function $c : V \to \{1,\ldots,k\}$ such that $c(u) \neq c(v)$ whenever u is adjacent to v. If a k-coloring of G exists, then G is called k-colorable. The *chromatic number* of G is defined as

$$\chi(G) \;=\; \min\{k \;:\; G \text{ is } k\text{-colorable}\}.$$

However, it was observed in [Hal80] that the signal propagation may affect the interference even in distant regions (but with decreasing intensity). Hence, not only "close" transmitters should get different frequencies, but also frequencies used at some distance should be appropriately separated. In this case, the FAP can be modeled as the problem of *distance-constrained labeling* of the interference graph [Lee98]. For an illustration see Figure 2.3 on the next page.

Distance Constrained Labeling. Let $p_1 \geq p_2 \geq \ldots \geq p_k$ be a non-increasing sequence of positive integers, called *distance constraints*. An $L_{(p_1,\ldots,p_k)}$*-labeling*, or

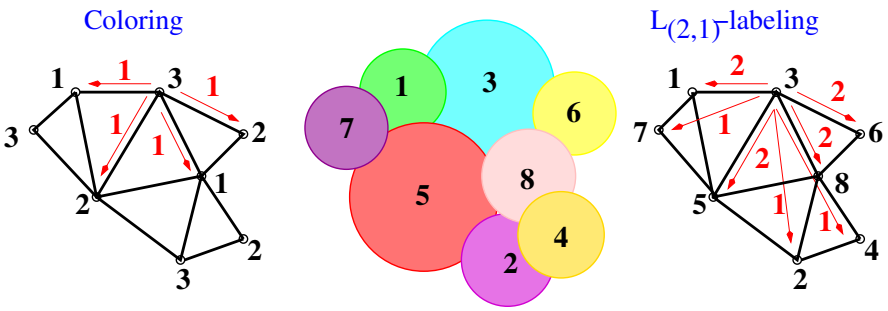


Figure 2.3: FAP – $L_{(2,1)}$-labeling

a *distance constrained labeling*, of a graph G is a function $c : V(G) \to \{1,\ldots,L\}$ such that $|c(u)-c(v)| \geq p_i$ whenever the distance between u and v in G is at least i, for $i = 1,\ldots,k$. If a $L_{(p_1,\ldots,p_k)}$-labeling of G exists, then G is called $L_{(p_1,\ldots,p_k)}$-labeled. The $(p_1,\ldots,p_k)$-labeling number of G is defined as

$$\chi_{(p_1,\ldots,p_k)}(G) \;=\; \min\{L \;:\; G \text{ is } L_{(p_1,\ldots,p_k)}\text{-labeled}\}.$$

First, we can observe the following simple properties. If $k = 1$ and $p_1 = 1$, then

$$\chi_{(1)}(G) = \chi(G), \tag{2.1}$$

where $\chi(G)$ is the chromatic number of G. If $p_1 = p_2 = \cdots = p_k = 1$, then

$$\chi_{(1,\ldots,1)}(G) = \chi(G^k), \tag{2.2}$$

where G^k is the k-th power of G, i.e. a graph which arises from G by adding the edges which connect all the vertices at the graph distance at most k. Furthermore, as it was shown in [GY92, FK02], for any integer t it holds

$$\chi_{(tp_1,\ldots,tp_k)}(G) = t\cdot(\chi_{(p_1,\ldots,p_k)}(G) \;-\; 1) \;+\; 1. \tag{2.3}$$

Hence, we can assume w.l.o.g. that all integers $p_1,\ldots,p_k$ have no common divisor. Combining (2.2) and (2.3), we can bound

$$\begin{aligned}\chi_{(p_1,\ldots,p_k)}(G) \;&\leq\; \chi_{(p_1,\ldots,p_1)}(G)\\ &=\; 1 \;+\; p_1(\cdot\chi_{(1,\ldots,1)}(G) \;-\; 1)\\ &=\; 1 \;+\; p_1(\chi_{(1)}(G^k) - \;1).\end{aligned} \tag{2.4}$$

Accordingly, for $k = 2$ and $(p_1, p_2) = (2,1)$ we have

$$\begin{aligned} \chi_{(2,1)}(G) &\leq \chi_{(2,2)}(G) \\ &= 2(\chi_{(1,1)}(G) - 1) + 1 \\ &\leq 2\chi_{(1)}(G^2) \\ &= 2\chi(G^2). \end{aligned} \tag{2.5}$$

McCormick [McC83] showed that for any fixed $k \geq 2$ finding the value of $\chi(G^k)$ is an NP-hard problem. Furthermore, even if one restricts to a planar graph G, computing $\chi(G^2)$ is still an NP-hard problem. There is the long-standing Wegner's conjecture [Weg77]: For any planar graph G with the maximum degree $\Delta(G) \geq 8$, the chromatic number of the 2nd power graph G^2 is at least $\lceil \frac{3}{2}\Delta \rceil + 1$. There are a number of recent results coming closer and closer to the conjectured bound. The current best result $\chi(G^2) \leq \frac{5}{3}\Delta + 78$ is due to Molloy & Salavatipour [MS].

The most intensively studied case of distance-constrained labeling is $k = 2$ and $(p_1, p_2) = (2,1)$. The existence of an $L_{(2,1)}$-labeling was explored for different graph classes in [BKTvL00, CK96, GM96, GY92, vdHLS98]. The exact value of $\chi_{(2,1)}$ can be derived for *cycles*, and there are polynomial-time algorithms which compute the value of $\chi_{(2,1)}$ for *trees* and *co-graphs* [CK96]. For any fixed $L \geq 4$, the problem of recognizing graphs G such that $\chi_{(2,1)}(G) \leq L$ is NP-complete [FKK99]. For a planar graph G, the problem of deciding whether $\chi_{(2,1)}(G) \leq 9$ was shown to be NP-complete in [BKTvL00]. Molloy & Salavatipour [MS] presented an approximation algorithm which produces an $L_{(p_1,p_2)}$-labeling of a planar graph G with the largest label at most $\frac{5}{3}(2p_2 - 1)\Delta(G) + 12p_1 + 144p_2 - 78$.

It is expected that for every k-tuple of distance constraints $(p_1, \ldots, p_k)$ and a graph G, there exists a bound L_0 such that for every $L \geq L_0$ the decision problem $\chi_{(p_1,\ldots,p_k)}(G) \leq L$ is NP-complete. So far, this conjecture has been proven for $k = 2$ and (p_1, p_2), where $p_1 \geq 2p_2$ [Fia00].

Disk Graphs. The intersection graph G of a set D of disks in the plane is called a *disk graph*, and D is called the *disk representation* of G. The *diameter ratio* of D is denoted by $\sigma(D)$. If all disks of D have unit diameter and $\sigma(D) = 1$, then G is called a *unit disk graph*. If $1 < \sigma(D) \leq \sigma$ for some constant σ, then G is called a σ-*disk graph*.

Interestingly, every planar graph is a *coin graph*, i.e. the intersection graph of interior-disjoint disks [Koe36]. Hence, the class of disk graphs is more general than the class of planar graphs. The recognition problem of a (unit, σ-) disk graph is NP-hard [BK96, BK98, HK01]. Hence, an algorithm that works on the set of graph's disks as the input is substantially weaker than one which works only on the sets of graph's nodes and edges. From this point of view, the requirement of the disk representation of a disk graph is very strong. From another side, it is not hard to find the disk representation of a disk graph when dealing with real-world applications, e.g. in constructing the interference graph for a radio and/or mobile telephony network.

There are a number of results on coloring of disk graphs. For a unit disk graph, the 3-coloring is NP-complete even when the disk representation is given [CCJ90]. There are a 3-approximation algorithm [BK98, Pee91] and a 5-competitive algorithm [Mal97, Pee91]. Both algorithms assume the knowledge of the disk representation of a disk graph, but they can be also easily adjusted to the general case [EF01]. Regarding disk graphs, there is a 5-approximation algorithm which uses the disk representation of a disk graph [Mal97]. On the other hand, there is no online coloring algorithm with a constant competitive ratio for planar graphs [GL88]. Hence, there is no such an on-line algorithm for general disk graphs as well.

Our Results. Here we consider the problem of distance-constrained labeling of σ-disk graphs. We present several off-line and on-line algorithms for the case of general distance constraints $(p_1,\ldots,p_k)$ and for the case when $k=2$ and $(p_1,p_2)=(2,1)$.

First, for a fixed k-tuple of distance constraints $(p_1,\ldots,p_k)$ we give an on-line $L_{(p_1,\ldots,p_k)}$-labeling algorithm which requires the disk representation of σ-disk graphs, i.e. it works on a set of disks D with $\sigma(D)\leq\sigma$ as the input. The algorithm is based on the so-called *hexagonal tiling*, *circular labeling*, and *first-fit* techniques. We derive an upper bound on its competitive ratio and show for each fixed k-tuple $(p_1,\ldots,p_k)$ of distance constraints and each fixed diameter ratio σ the on-line algorithm is constant competitive. As an example, we demonstrate the algorithm for the case $k=2$ and $(p_1,p_2)=(2,1)$. We show that for σ-disk graphs with at least one edge and $\sigma\leq\sqrt{7}/2$ the competitive ratio of the algorithm is bounded by 16.67. The ratio also tends to 12.5 as the clique number of a graph grows to infinity.

Next, we derive lower bounds for on-line coloring and labeling. We consider the case when the disk representation of disk graphs is known. We start with simple lower bounds for unit disk graphs. We show that no on-line coloring algorithm

can be better than 2-competitive, and no on-line $L_{(2,1)}$-labeling algorithm can be better than 5-competitive. Then, we switch to disk graphs. We prove that in the case when either the disk representation of disk graphs is not given or the diameter ratio is not bounded, no on-line labeling algorithm with a constant competitive ratio exists. In addition, we give a lower bound on any general $L_{(p_1,\ldots,p_k)}$-labeling algorithms for σ-disk graphs. By using this result we show that our on-line labeling algorithm is asymptotically optimal for the class of unit disk graphs.

Finally, we deal with the off-line setting. We explore the case $k = 2$ and $(p_1, p_2) = (2,1)$. We present two approximation algorithms for unit disk graphs. The first algorithm is based on the so-called *cutting* technique, which uses the disk representation of a unit disk graph. The second algorithms is *robust*, what is, it does not require the disk representation, and it either outputs a feasible labeling or shows that the input is not a unit disk graph. The approximation ratio of the *cutting* algorithm is bounded by 12, whereas the approximation ratio of the *robust* algorithm is bounded by $10,67$. The bound also tends to 9 and to 10 as the clique number of a unit disk graph grows to infinity, respectively. Finally, we present a simple offline $L_{(p_1,\ldots,p_k)}$-labeling algorithm for σ-disk graphs which does not require the disk representation. For each fixed σ and k the approximation ratio of the algorithm is constant $O(k^2\sigma^2)$.

The following table summarizes the known and new online and approximation algorithms for coloring and labeling problems on unit disk graphs (UDG), on σ-disk graphs (σ-DG), and on disk graphs (DG).

	Offline		Online	
	+	−	+	−
Coloring				
UDG	3 [Pee91]	3 [Pee91]	5 [Mal97, Pee91]	5 [Mal97, Pee91]
σ-DG	5 [Mal97]	5 [Mal97]	[∗]	YES [EF01]
DG	5 [Mal97]	5 [Mal97]	NO [EF01]	NO [GL88]
$L_{(2,1)}$-labeling				
UDG	12 [∗]	10.6 [∗]	16.67 [∗]	NO [∗]
$L_{(p_1,\ldots,p_k)}$-labeling				
UDG	YES [∗]	YES [∗]	YES [∗]	NO [∗]
σ-DG	YES [∗]	YES [∗]	YES [∗]	NO [∗]
DG	?	?	NO [∗]	NO [∗]

Here, "+/-" shows either the disk representation of graphs is given or not; "YES" means a constant competitive algorithm; "NO" means that no constant competitive algorithms can exist; "?" shows an open problem; "[∗]" means a result presented in this chapter; "number" corresponds to the approximation ratio or the competitive ratio of the respective algorithm.

Last Notes. Let A be an algorithm which works on disk graphs. Let $D = \{D_1, D_2, \ldots, D_n\}$ be a set of disks. Let G be the disk graph of D. If A works on D as the input, we say that A *requires* the disk representation of G, and if A works on $V(G)$ and $E(G)$ as the input, we say that A *does not require* the disk representation of G. We say that an algorithm A is an *offline* $L_{(p_1,\ldots,p_k)}$*-labeling algorithm* if it runs in polynomial time and outputs a proper labeling of the vertices of G. If the maximum label used is at most $\rho \cdot \chi_{(p_1,\ldots,p_k)}(G)$, then A is called an ρ-*approximation* algorithm. The value ρ is called the *approximation ratio* of A. We say that an algorithm A is an *online* $L_{(p_1,\ldots,p_k)}$*-labeling algorithm* if it properly labels the vertices of G in an externally determined order $D_1 \prec \cdots \prec D_n$, and at each time t it irrevocably assigns a label to D_t seeing only the edges which connect vertices $D_1, \ldots, D_t$. If the maximum label used is at most $\rho \cdot \chi_{(p_1,\ldots,p_k)}(G)$ for any order $\prec$ on D, then A is called an ρ-*competitive* algorithm. The value ρ is called the *competitive ratio* of A.

The rest of this chapter is organized as follows. In Section 2.2 we give some preliminary results. In Section 2.3 we introduce a *circular labeling*. In Section 2.4 we present a general online algorithm and derive an upper bound on its competitive ratio. In Section 2.5 we present lower bounds for online coloring and labeling. In Section 2.6 we present two offline $L_{(2,1)}$-labeling algorithms. In Section 2.7 we derive a general offline labeling algorithm. Finally, in Section 2.8 we give some concluding remarks.

2.2 PRELIMINARIES

In this section we give some preliminary results which will be used throughout this chapter. First, we introduce hexagonal cells on the plane and cell cliques in a disk graph. Then, we introduce the plane-mesh distance, and derive some simple results.

Let $\mathcal{E}$ be the Euclidean plane. Let x, y be the coordinates in $\mathcal{E}$. For a σ-disk graph G we will use $D = \{D_1, \ldots, D_n\}$ to denote the disk representation of G. Then, for each D_i, $i = 1, \ldots, n$, we will use $d_i \in \mathbb{R}_+$ and (x_i, y_i) to denote the diameter and center of D_i. We will also write $\sigma(D)$ to denote the diameter ratio $\max d_i / \min d_i \leq \sigma$. For each vertex $v \in V(G)$, we will use D_v to denote the disk of v. Thus, an edge $e = \{u, v\} \in E(G)$ iff $D_v \cap D_u$.

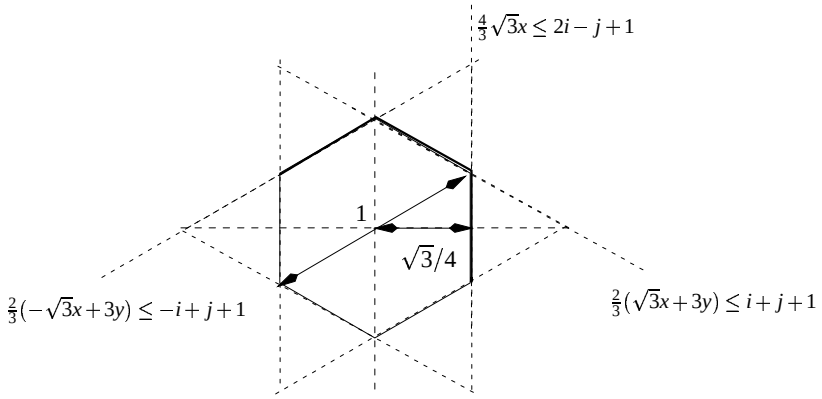


Figure 2.4: A simplex C_{ij}

2.2.1 Cells

We will use the following partition of plane $\mathcal{E}$ into hexagons. For $i, j \in \mathbb{Z}$ we define a unit hexagon C_{ij} is the set of all points $(x,y) \in \mathcal{E}$ such that:

$$\begin{aligned} 2i-j-1 < &\quad \tfrac{4}{3}\sqrt{3}x &&\le 2i-j+1 \\ i+j-1 < &\quad \tfrac{2}{3}(\sqrt{3}x+3y) &&\le i+j+1 \\ -i+2j-1 < &\quad \tfrac{2}{3}(-\sqrt{3}x+3y) &&\le -i+2j+1. \end{aligned}$$

Here, C_{ij} contains exactly two adjacent corners of the bounding simplex, and the distance between every two points inside C_{ij} is at most one. For an illustration see Figure 2.4. Furthermore, each point of plane $\mathcal{E}$ belongs to exactly one cell $C_{i,j}$. For an illustration see Figure 2.5 on page 110. For simplicity, any C_{ij} will be called a *cell*, and $\mathcal{C}$ will denote the set of all cells C_{ij}, for $i, j \in \mathbb{Z}$.

2.2.2 Cell Cliques

For a disk graph G given by a disk set D, and a cell C_{ij} let

$$D(i,j) := \{D_k \mid D_k \in D \text{ and } (x_k,y_k) \in C_{ij}\}$$

be the set of disks with centers in C_{ij}, and let

$$V(i,j) := \{v \in V(G) \mid D_v \in D(i,j)\}$$

be the set of vertices with disks in C_{ij}. Then, we can prove the following simple result:

Lemma 2.2.1. *Each set $V(i,j)$ induces a clique in G. Hence, $|D(i,j)| = |V(i,j)|$ is at most the clique number $\omega(G)$.*

Proof. The distance between every two points inside cell C_{ij} is at most one. Hence, any pair of disks in $D(i,j)$ intersect. This means that for any two $u,v \in V(i,j)$ holds $\{u,v\} \in E(G)$. Hence, $V(i,j)$ induces a clique in G. □

2.2.3 Plane and Mesh Distance

Let $\mathrm{dist}_{\mathcal{E}}(p,p')$ denote the standard plane distance between two points $p,p' \in \mathcal{E}$. Then, the *plane distance* between two cells C and C' is defined as

$$\mathrm{dist}_{\mathcal{E}}(C,C') \;=\; \inf\{\mathrm{dist}_{\mathcal{E}}(p,p') : p \in C, p' \in C'\}.$$

With every cell $C_{ij} \in \mathcal{C}$ we associate a vertex (i,j) in an infinite triangular mesh M. We connect any two vertices by an edge if the corresponding cells are neighbors. For an illustration see Figure 2.5 on the following page.

We will write $\mathrm{dist}_M(C_{ij},C_{st})$ to denote the *mesh distance* between two cells C_{ij} and C_{st}, which is measured as the number of edges in the shortest path connecting (i,j) and (s,t) in mesh M.

Lemma 2.2.2. *For $m \geq 2$ and $i,j \in \mathbb{Z}$, each of cells $C_{i+t,j}$, $C_{i,j+t}$, $C_{i+t,j+t}$, where $t \in \{m+1,-m-1\}$, have mesh distance $m+1$ and plane distance $\frac{m\sqrt{3}}{2}$ from C_{ij}. Furthermore, any cell at mesh distance $m+1$ from C_{ij} has plane distance at least $\lfloor \frac{m}{2} \rfloor + \frac{1}{2}\lceil \frac{m}{2} \rceil$.*

Proof. Every cell has size $\sqrt{3}/2$, see Figure 2.4 on the preceding page. For simplicity, we consider the case when $i=0$ and $j=0$ and $t=m+1$. For an illustration see Figure 2.6 on the following page. Clearly, $C_{m+1,0}$, $C_{0,m+1}$ and $C_{m+1,m+1}$ are at mesh distance $m+1$, see Figure 2.6 b). Furthermore, there are m cells on the shortest line from $C_{0,0}$, see Figure 2.6 a). Hence, the plane distance is $\frac{m\sqrt{3}}{2}$.

Consider the cells which are mesh distance $m+1$. For an illustration see Figure 2.7 on page 111. From one side, the "corner" cells $C_{m+1,0}$ and $C_{m+1,m+1}$ are at the maximum plane distance from $C_{0,0}$, see Figure 2.7 a). From another side, the "middle" cells, $C_{m+1,m/2}$ if m is even and $C_{\lfloor \frac{m+1}{2} \rfloor, m+1}$, $C_{m+1-\lfloor \frac{m+1}{2} \rfloor}, m+1$ if m is odd, are at the minimum plane distance from $C_{0,0}$. Then, the minimum plane distance can be bounded as $\lfloor \frac{m}{2} \rfloor$ times cell's diameter 1 and $\lceil \frac{m}{2} \rceil$ times the cell's side $\frac{1}{2}$. This is equal to $\lfloor \frac{m}{2} \rfloor + \frac{1}{2}\lceil \frac{m}{2} \rceil$. □

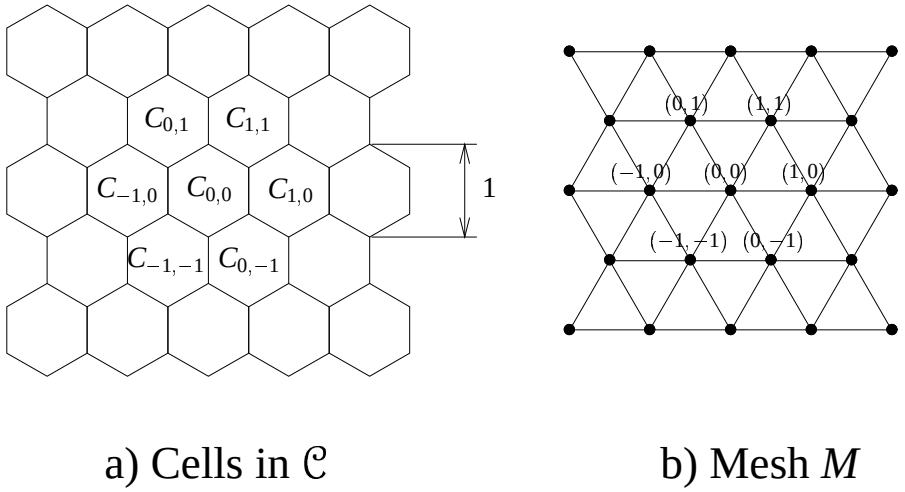


Figure 2.5: Cells-Mesh

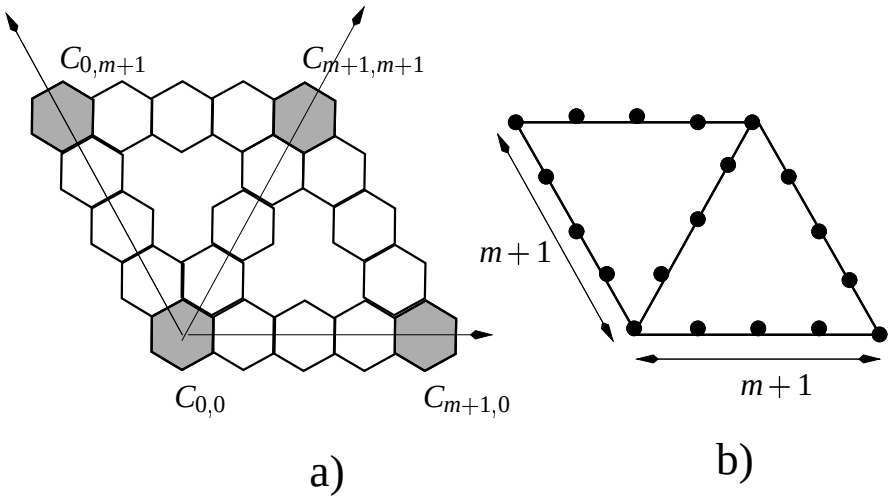


Figure 2.6: Cells for $i = 0$ and $j = 0$ and $t = m+1$

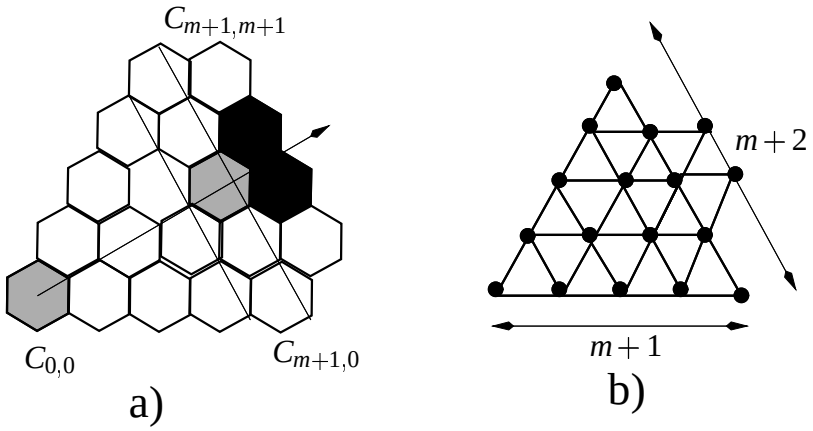


Figure 2.7: Middle cells

Corollary 2.2.3. *For $m \geq 2$ and $i, j \in \mathbb{Z}$, cells $C_{i,j}, C_{i+m+1,j}, C_{i,j+m+1}, C_{i+m+1,j+m+1}$ have pairwise mesh distance $m+1$ and plane distance $\frac{m\sqrt{3}}{2}$.*

Corollary 2.2.4. *Let $a = \left\lceil \frac{2k\sigma}{\sqrt{3}} \right\rceil$, where $k \geq 2$ and $\sigma \geq 1$. Then, cells $C_{i,j}$, $C_{i+t,j}$, $C_{i,j+t}$, $C_{i+t,j+t}$, where $t \in \{a+1, -a-1\}$, have pairwise mesh distance $a+1$ and pairwise plane distance greater than $k\sigma$.*

2.2.4 Patterns

Let $k \geq 2$ and $\sigma \geq 1$. As in Corollary 2.2.4, we define $a = \left\lceil \frac{2k\sigma}{\sqrt{3}} \right\rceil$. Then, the set of a^2 cells C_{st} with coordinates $s,t \in \{0,..,a\}$ is called a pattern.

We say that a cell $C_{ij} \in \mathcal{C}$ belongs to the (s,t)*th class* if

$$i - 1 = s \bmod a$$

and

$$j - 1 = t \bmod a.$$

In total, there are a^2 classes.

Informally, by sifting the pattern around the plane, we "copy" its cells. For an illustration see Figure 2.8 on the following page. Then, a cell C_{ij} belongs to a the (s,t)th class if it is a "copy" of the (s,t)th cell in the pattern.

Now we can prove the following simple result:

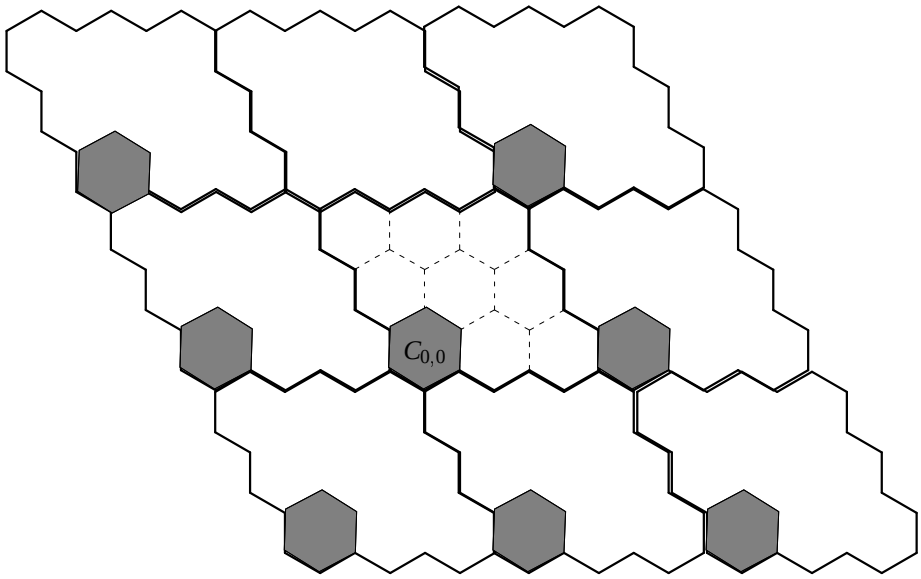


Figure 2.8: Shifting the pattern: The copies of $C_{0,0}$

Lemma 2.2.5. *Any two cells in one class have plane distance greater than* $k\sigma$.

Proof. The proof follows the definition of classes and Corollary 2.2.4. □

2.3 CIRCULAR LABELING

Here we introduce and prove the existence of a special *circular labeling* for the cells in $\mathcal{C}$. This will will be used later in Section 2.4.

Let $\sigma \geq 1$ be the diameter ratio, $(p_1, \ldots, p_k)$ be a k-tuple of distance constraints, where $p_1 \geq p_2 \geq \ldots, \geq p_k$, and $\mathcal{C}$ be the set of cells C_{ij}, where $i, j \in \mathbb{Z}$. We say that a mapping $\varphi : \mathcal{C} \to \{1, 2, \ldots, \ell\}$ is an *ℓ-circular labeling* of $\mathcal{C}$ with respect to $(p_1, \ldots, p_k)$ and σ if for any two cells C' and C'' in $\mathcal{C}$ at plane distance $\mathrm{dist}_{\mathcal{C}}(C, C') \leq i \cdot \sigma$ it holds

$$\min\{|\varphi(C) - \varphi(C')|, \ell - |\varphi(C) - \varphi(C')|\} \geq p_i,$$

for all $i \in \{1, \ldots, k\}$.

For an illustration see Figure 2.9 on the next page. Informally, we take a circle with nodes $1, 2, \ldots, \ell$. Then, every cell C is assigned to a node $\varphi(C) \in \{1, 2, \ldots, \ell\}$. The "circular distance" between any two cells C and C' is equal to the number

edges between nodes $\varphi(C)$ and $\varphi(C')$. This can be defined as

$$\min\{|\varphi(C)-\varphi(C')|, \ell - |\varphi(C)-\varphi(C')|\}.$$

Then, we require any two cells C and C' at plane distance at least $i\cdot\sigma$ be at "circular distance" at least p_i, for all $i \in \{1,\ldots,k\}$.

The existence of such a circular labeling is guaranteed by the following:

Theorem 2.3.1. *For every k-tuple $(p_1,\ldots,p_k)$ and $\sigma \geq 1$, an ℓ^*-circular labeling of $\mathcal{C}$ can be found in $O(\ell^*\sigma^4k^4)$ time, where*

$$\ell^* := 1 + 6\left(4(2p_1 - 1) + \sum_{m=2}^{a}(m+1)\cdot(2p_{\lceil\frac{3m-4}{4\sigma}\rceil} - 1)\right).$$

Proof. Given k and $\sigma \geq 1$, we define $a = \lceil\frac{2k\sigma}{\sqrt{3}}\rceil$, and define a pattern with all cells $C_{s,t}$, where $s,t \in \{0,..,a\}$.

We select the cells in the pattern one by one while labeling with an initial sequence of labels $1,2,3,\ldots$ in a *first-fit* manner. For a selected cell C_{st} from the pattern we first find the least feasible label $\varphi_{s,t}$, and then we define $\varphi(C) = \varphi_{s,t}$ for any cell C in the (s,t)th class. By Lemma 2.2.5, any two cells in one class have plane distance greater than $k\sigma$. Hence, at the end of the procedure we find a feasible circular labeling of $\mathcal{C}$.

In the following we show that ℓ^* is a upper bound on the largest $\varphi_{s,t}$ label used in the pattern, and the labeling procedure takes at most $O(\ell^*\sigma^4k^4)$ steps. This will complete the proof of the theorem.

Consider a cell C in the pattern. For an illustration see Figure 2.10 on the following page. By Corollary 2.2.4, every cell which is at mesh distance at least $a+1$ is at

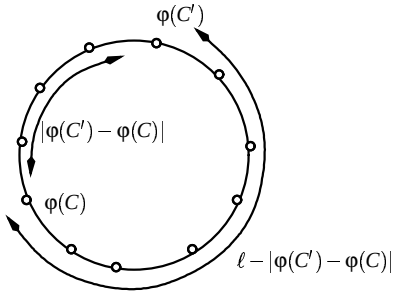


Figure 2.9: A circle with ℓ nodes, and cells C and C'

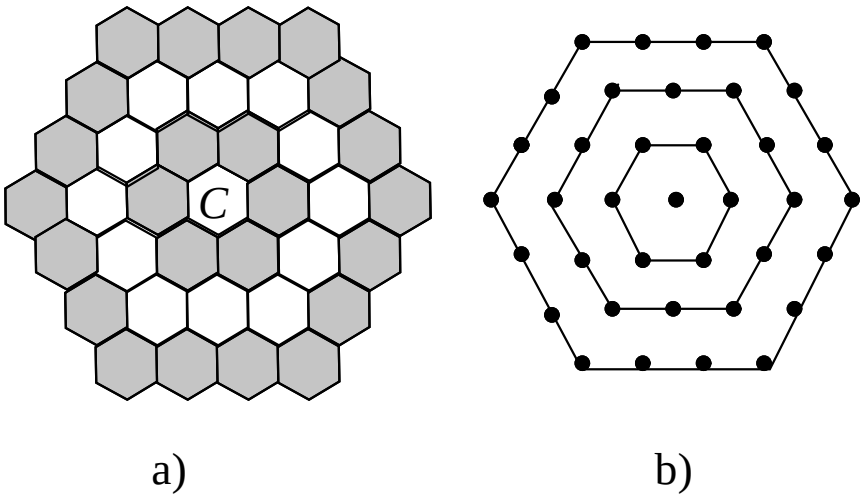


Figure 2.10: Labeling of C

plane distance greater than $k\sigma$. Hence, in order to find a feasible label for C we need to check all already labeled cells at mesh distance at most a.

There are 6 cells at mesh distance 1 from C, see Figure a) and b) 2.10. Each of these 6 cells has plane distance at most $1 \cdot \sigma$ from C. In the worst case, all 6 cells are labeled, and any two of the labels differ by $2p_1 - 1$. Hence, in order to select a feasible label for C we will "skip" at most $6(2p_1 - 1)$ "forbidden" numbers. Similarly, for 12 cells at mesh distance 2 from C, we will "skip" at most $12(2p_1 - 1)$ "forbidden" numbers.

For $m \geq 2$, there are $6(m+1)$ cells at mesh distance $m+1$ from C. By Lemma 2.2.2, the plane distance from C is at most $m\sqrt{3}/2$ but at least

$$\left\lfloor \frac{m}{2} \right\rfloor + \frac{1}{2} \left\lceil \frac{m}{2} \right\rceil .$$

By the definition of a circular labeling, we need to find the least integer $i \leq k$ such that

$$\left\lfloor \frac{m}{2} \right\rfloor + \frac{1}{2} \left\lceil \frac{m}{2} \right\rceil \leq i \cdot \sigma.$$

We can bound it as follows

$$\begin{aligned} i &\geq \frac{1}{\sigma}\left(\left\lfloor\frac{m}{2}\right\rfloor + \frac{1}{2}\left\lceil\frac{m}{2}\right\rceil\right) \\ &\geq \frac{1}{\sigma}\left(\frac{m}{2} - 1 + \frac{m}{4}\right) \\ &= \frac{(3m-4)}{4\sigma}. \end{aligned}$$

Then, in the worst case, all $6(m+1)$ cells are labeled, and any two of the labels differ by

$$2p_{\lceil\frac{3m-4}{4\sigma}\rceil} - 1.$$

As before, in the worst case we will "skip" at most

$$6(m+1)(2p_{\lceil\frac{3m-4}{4\sigma}\rceil} - 1)$$

"forbidden" numbers.

In total, summing up for mesh distance 1, 2 and over all $3 \leq m+1 \leq a$ at most

$$6\left(4(2p_1 - 1) + \sum_{m=2}^{a}(m+1)\cdot(2p_{\lceil\frac{3m-4}{4\sigma}\rceil} - 1)\right) = \ell^* - 1$$

numbers are "forbidden" be selected as a label for cell C in the pattern.

There are $a^2 = O(k^2\sigma^2)$ cells in the pattern. For each cell C in the pattern we have to check all cells at mesh distance at most a, and each cell for at most ℓ^* numbers. Thus, the labeling procedure procedure finds an ℓ^*-circular labeling of $\mathcal{C}$ in at most $O(\ell^* a^4) = O(\ell^* k^4\sigma^4)$ steps. □

2.3.1 *A Circular* 25*-Labeling for* $(p_1, p_2) = (2,1)$

Consider $k = 2$ and $(p_1, p_2) = (2,1)$. We take a pattern with 25 cells, and label the cells of $\mathcal{C}$ as it is depicted in Figure 2.11 on the next page. One can see that any two cells with the same label are at plane distance at least $2\sqrt{3}$. Furthermore, any two cells with ℓ and $\ell+1$ labels, $\ell = 1,\ldots,24$, are at plane distance at least $\frac{\sqrt{7}}{2}$. If we define $\sigma = \frac{\sqrt{7}}{2}$, then $2\sigma < 2\sqrt{3}$. Hence, the depicted labeling is a 25-circular labeling with respect to $(p_1, p_2) = (2,1)$ and $\sigma = \frac{\sqrt{7}}{2}$.

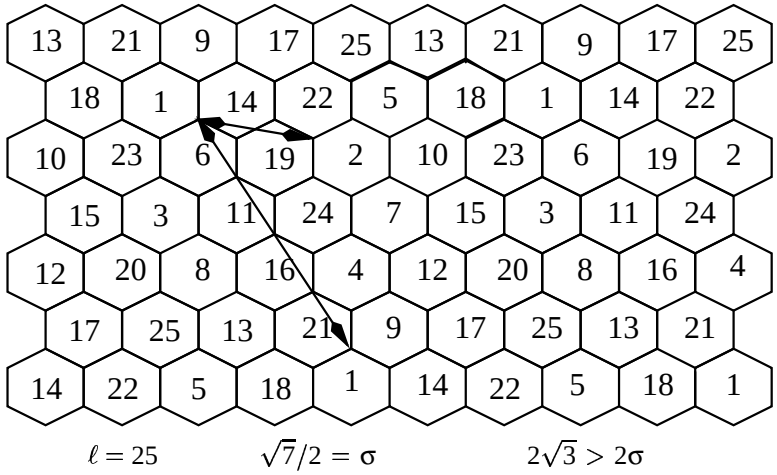


Figure 2.11: A 25-circular labeling with $(p_1, p_2) = (2,1), \sigma = \frac{\sqrt{7}}{2}$

2.4 GENERAL ONLINE LABELING OF σ-DISK GRAPHS

Let G be a σ-disk graphs given by a set $D = \{D_1, \ldots, D_n\}$ of n disks in $\mathcal{E}$. In the following we assume, w.l.o.g., that the coordinates of plane $\mathcal{E}$ are scaled such that minimum diameter $\min d_t = 1$ and the diameter ratio $\sigma(D) \leq \sigma$.

For a fixed k-tuple $(p_1, \ldots, p_k)$ of distance constraints, where $p_1 \geq p_2 \geq \ldots \geq p_k$, and a fixed $\sigma \geq 1$, we describe the following online labeling algorithm:

ONLINE DISK LABELING (ODL):
Input: An ordered sequence of disks $D_1 \prec \cdots \prec D_n$.
Output: An $L_{(p_1,\ldots,p_k)}$-labeling c of G.
1. Find a circular ℓ^*-labeling $\varphi : \mathcal{C} \to \{1, \ldots, \ell^*\}$.
2. For all cells $C_{i,j} \in \mathcal{C}$ define $D(i,j) := \emptyset$.
3. Select the disks one by one in the given order.
4. For the disk D_t perform
   4a. Find C_{ij} such that $(x_t, y_t) \in C_{i,j}$.
   4a. Define $t \in V(G)$. 4b. Define $c(t) := \varphi(C_{i,j}) + \ell^* \cdot |D(i,j)|$.
   4c. Put D_k into $D(i,j)$.

Informally, for each new disk the algorithm assigns a label which consists two parts: (1) the label of the cell which will contain this disk; (2) ℓ^* times the number of the disks which are already in the cell. The last part insures that all disk labels are properly separated.

We can prove the following result:

Lemma 2.4.1. *The maximum label used by* ODL *is most* $\ell^* \cdot \max_{i,j} |D(i,j)|$.

Proof. The first disk in $D(i,j)$ will get a label equal to

$$\varphi(C_{i,j}) \leq \ell^*.$$

The last disk in $D(i,j)$ will get a label equal to

$$\varphi(C_{i,j}) \ + \ \ell^* \cdot (|D(i,j)|) \leq \ell^* \cdot \max_{i,j} |D(i,j)|.$$

Since, ODL handles all $D(i,j)$ separately, the maximum label ised is bounded by

$$\ell^* \cdot \max_{i,j} |D(i,j)|.$$

□

Furthermore, we can prove the following result:

Lemma 2.4.2. *Let G be a disk graph given by a disk set D. Then, for any k-tuple* $(p_1, \ldots, p_k)$ *of distance constraints it holds*

$$\chi_{(p_1,\ldots,p_k)}(G) \ \geq \ 1 \ + \ p_1(\omega(G) \ - \ 1) \ \geq \ 1 + \ p_1(\max_{i,j}\{|D(i,j)|\} \ - \ 1).$$

Proof. Let K be a clique in G. Assume that one vertex in K has the least label 1, and other $|K|-1$ vertices have larger labels. By the definition of a $L_{(p_1,\ldots,p_k)}$-labeling, the labels of any two vertices in K should differ by at least p_1. Thus, the minimum label for K is at least

$$1 \ + \ p_1(|K| \ - \ 1).$$

By Lemma 2.2.1 for any set $D(i,j)$ of disks, the vertices of $V(i,j)$ form a clique in G and $|D(i,j)| = |V(i,j)|$ is at most the clique number $\omega(G)$. Thus, the $(p_1, \ldots, p_k)$-labeling number for G is at least $1 \ + \ p_1(\omega(G) \ - \ 1)$. □

Combining the above results, we can prove the following main theorem:

Theorem 2.4.3. *For every* $(p_1, \ldots, p_k)$ *and* $\sigma \geq 1$*, the algorithm* ODL *is an online* $L_{p_1,\ldots,p_k}$*-labeling algorithm for* σ*-disk graphs G with given disk representation. The competitive ratio* ρ *of* ODL *is bounded by*

$$\frac{\omega(G) \cdot \ell^*}{1 \ + \ (\omega(G) \ - \ 1) \cdot p_1} \ \leq \ \ell^*. \tag{2.6}$$

Proof. Let G be a σ-disk graph given by a disk set D. Notice that the value of $|D(i,j)|$ does not depend on an order in which the disks of D presented to ODL. Hence, ODL an online $L_{(p_1,\dots,p_k)}$-labeling algorithm for G. Furthermore, by Lemma 2.4.1 and Lemma 2.4.2, we can bound its competitive ratio ρ as it is defined in (2.6). □

Corollary 2.4.4. *The algorithm* ODL *is* $\frac{2\ell^*}{1+p_1}$*-competitive for the class of* σ*-disk graphs with at least one edge and given disk representation. Furthermore, the bound on its competitive ratio* ρ *tends to* ℓ^*/p_1 *as the clique number of* σ*-disk graphs grows to infinity.*

Proof. If a disk graph G has at least one edge, then $\omega(G) \geq 2$. From (2.6), for $w(G) = 2,3,4,\dots$ we have

$$\frac{2\ell^*}{1+p_1} \geq \frac{3\ell^*}{1+2p_1} \geq \frac{4\ell^*}{1+3p_1} \geq \cdots \geq \frac{\ell^*}{p_1}.$$

□

Corollary 2.4.5. *For* $(p_1,p_2) = (2,1)$ *and* $\sigma = \frac{\sqrt{7}}{2}$*, there is an online* $L_{(2,1)}$*-labeling algorithm which competitive ratio* ρ *is bounded by* 25 *for a class of* σ*-disk graphs of given disk representation, by* $\frac{50}{3} \approx 16.67$ *for a class of* σ*-disk graphs of with at least one edge and given disk representation, and the bound on* ρ *tends to* 12.5 *as the clique number of* σ*-disk graphs grows to infinity.*

Proof. We use the algorithm ODL combined with a 25-circular labeling depicted in Figure 2.11 on page 116. □

2.5 LOWER BOUNDS: ONLINE COLORING AND LABELING

Here we present some lower bounds for online coloring and labeling of disk graphs.

2.5.1 Coloring of Unit Disk Graphs

We start with a simple lower bound for online coloring of unit disk graphs.

Lemma 2.5.1. *For any positive* ε*, there is no* $(2-\varepsilon)$*-competitive coloring algorithm for the class of unit disk graphs, even if the disk representation of unit disk graphs is given.*

Proof. Let A be an algorithm with competitive ratio $2-\varepsilon$, for some $\varepsilon > 0$. Consider a unit disk graph G_{bad} depicted in Figure a) 2.12 on the next page. Let the vertices of G_{bad} be ordered as shown in Figure b) 2.12 on the following page.

From one side, vertices 1–6 form an independent set. The algorithm A has to color them by the same color. If it is not the case, then A is not $(2-\varepsilon)$-competitive. From another side, vertices 1–12 form a bipartite graph. To color them properly, the algorithm A needs exactly two more colors. Then, vertices 13, 14 and 15 require three extra colors. These vertices form a triangle, so they cannot share the same color, and each of them is adjacent to three vertices among 1–12 that are colored by three distinct colors.

In other words, A is forced to use at least six colors for online coloring of G_{bad}. However, the graph is 3-colorable. Hence, A is not an $(2-\varepsilon)$-competitive algorithm, if the disk representation of unit disk graphs is given. □

2.5.2 Labeling of Unit Disk Graphs

Now we present a simple lower bound for online $L_{(p_1,p_2)}$-labeling of unit disk graphs.

Lemma 2.5.2. *For any distance constraints (p_1,p_2) and $\varepsilon > 0$, there is no $(4p_2+1-\varepsilon)$-competitive $L_{(p_1,p_2)}$-labeling algorithm for the class of unit disk graphs, even if the disk representation of unit disk graphs is given.*

Proof. Consider a unit disk graph G_{bad} given by five "outer" unit disks 1, 2, 3, 4, 5 depicted in Figure 2.13 on the next page. No two of these five disks intersect. Hence, in the offline case, one needs exactly one label for G_{bad}. Hence, we have that $\chi_{(2,1)}(G_{\text{bad}}) = 1$.

Let A be an online $L_{(p_1,p_2)}$-labeling for unit disk graphs of given disk representation. For any disk graph and any order of the disks, A always outputs a feasible $L_{(p_1,p_2)}$-labeling.

It is not a matter in which order we present the disks of G_{bad}, any two labels assigned by A differ by at least p_2. If it is not the case, then adding the "central" unit disk 6 leads to a non-feasible for a unit disk graph given by all disks 1, 2, 3, 4, 5, 6, that is a contradiction.

Thus, the maximum label assigned by A to the disks of G_{bad} is at least

$$1 + p_2 + p_2 + p_2 + p_2 = 1 + 4p_2.$$

However, $\chi_{(2,1)}(G_{\text{bad}}) = 1$. Hence, the competitive ratio of A is at least $4p_2+1$, even if the disk representation of unit disk graphs is given. □

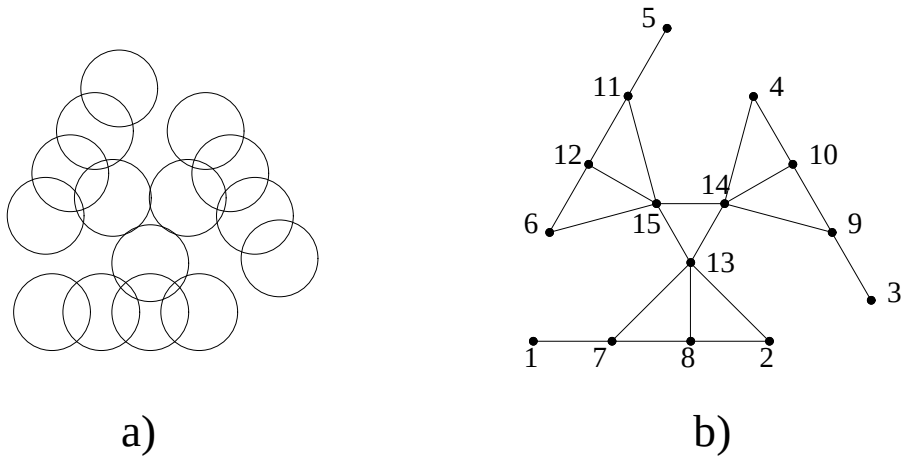


Figure 2.12: Graph G_{bad} for coloring

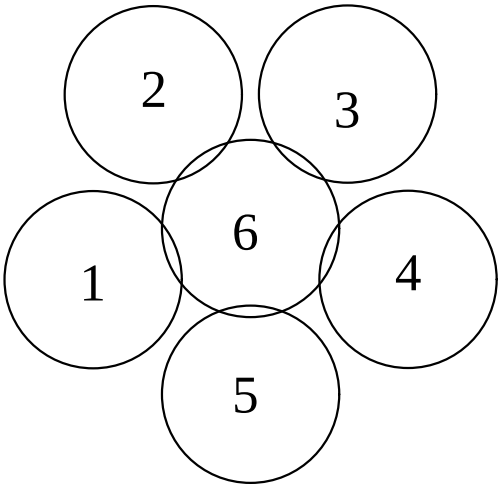


Figure 2.13: Graph G_{bad} for $L_{2,1}$-labeling

2.5.3 General Labeling of Disk Graphs

Let $k = 2$ and (p_1, p_2) be a 2-tuple of distance constraints. We can prove the following simple result:

Lemma 2.5.3. *If the disk representation of disk graphs is not given, no online $L_{(p_1,p_2)}$-labeling algorithm can be constant competitive.*

Proof. Let D be a set of n mutually disjoint disks. Let G a disk graph given by D. Then, there are no edges in G, and $\chi_{(p_1,p_2)}(G) = 1$.

Let A be an online $L_{(p_1,p_2)}$-labeling algorithm which is not given the disk representation of disk graphs. For any disk graph and any order of the vertices, A always outputs a feasible $L_{(p_1,p_2)}$-labeling.

We present the vertices v in $V(G)$ in an arbitrary order. If A assigns the same label to any two vertices in $V(G)$, then we add a new disk to D extending graph G such that these two vertices in are connected by a path of length 2. In this case, A outputs a non-feasible labeling for the "extended" graph, that is a contradiction. Hence, A will use distinct labels for all $|D|$ vertices in $V(G)$.

Thus, the maximum label used by A for G is at least $|D| = n$. However, $\chi_{(p_1,p_2)}(G) = 1$. Hence, the competitive ratio of A is bounded by n from below. □

This result can be generalized for any k-tuple $(p_1, p_2, \ldots, p_k)$ of distance constrains. Hence, in the following we only consider online algorithms which are given the disk representation of disk graphs. The next result shows the importance of an upper bound on the diameter ratio:

Lemma 2.5.4. *If an upper bound on the diameter ratio is not given, no online $L_{(p_1,p_2)}$-labeling algorithm can be constant competitive.*

Proof. Let D be a set of n mutually disjoint unit disks. For an illustration see Figure 2.14 on the following page. Let G a disk graph given by D. Then, there are no edges in G, and $\chi_{(p_1,p_2)}(G) = 1$.

Let A be an online $L_{(p_1,p_2)}$-labeling algorithm which is not given an upper bound on the diameter ratio. For any disk graph and any order of the disks, A always outputs a feasible $L_{(p_1,p_2)}$-labeling.

We present the disks of D in an arbitrary order to A. If A assigns the same label to any two disks of D, then we add a new disk to D of larger diameter which intersent any disks in D, see Figure 2.14. In this case, A outputs a non-feasible labeling for a disk graph given by the "extended" set of disks, that is a contradiction. Hence, A will use distinct labels for all disks in D.

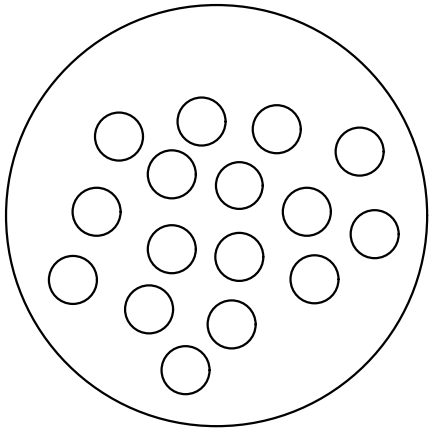


Figure 2.14: A set D of disks

The maximum label used by A for G is at least $|D| = n$. However, $\chi_{(p_1,p_2)}(G) = 1$. Hence, the competitive ratio of A is bounded by n from below. □

This result can be also generalized for any k-tuple $(p_1, p_2, \ldots, p_k)$ of distance constraints. Hence, in the following we only consider online algorithms which deal with σ-disk graphs. Now we are ready to present a lower bound on the competitive ratio of any online $L_{(p_1,\ldots,p_k)}$-labeling algorithm for σ-disk graphs:

Theorem 2.5.5. *For any fixed k-tuple $(p_1, \ldots, p_k)$ of distance constraints ($k \geq 2$), any fixed $\sigma \geq 1$ and any $\varepsilon > 0$, there is no $(\bar{\rho} - \varepsilon)$-competitive online $L_{(p_1,\ldots,p_k)}$-labeling algorithm for the class of σ-disk graphs of given disk representation, where*

$$\bar{\rho} \;=\; 1 \;+\; \frac{\sigma^2}{9} \max_{i=2,\ldots,k} \{i^2 p_i\}.$$

Proof. Take any $t \in (1, \sqrt{2})$ and define $a_k = \lfloor \frac{(k-1)\sigma+1}{t\sqrt{2}} + 1 \rfloor$. Next, define a set $D = \{D_{1,1}, D_{1,2}, \ldots, D_{a_k,a_k}\}$ of a_k^2 unit disks, where each disk $D_{j,l}$ is defined by its center in $(j \cdot t, l \cdot t)$, and all j, l are integers from $\{1, 2, \ldots, a_k\}$. All disks are mutually disjoint and the centers of any two closest disks at plane distance t. For an illustration see Figure 2.15 on the next page.

Consider a unit disk graph G given by D. Clearly, G consists of a_k^2 independent vertices (disks). In the offline case, we only need one label for G, i.e.,

$$\chi_{(p_1,\ldots,p_k)} \;=\; 1.$$

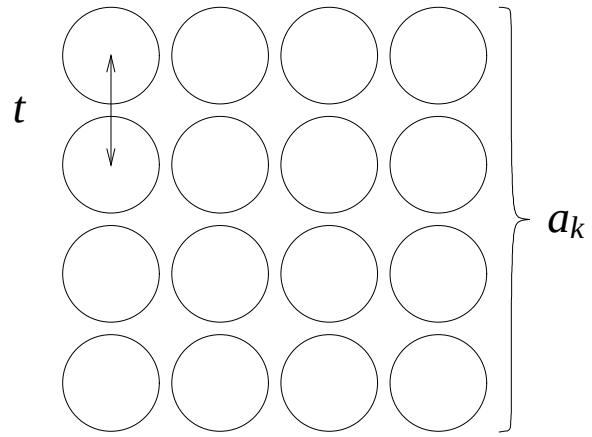


Figure 2.15: The set D of a_k^2 unit disks

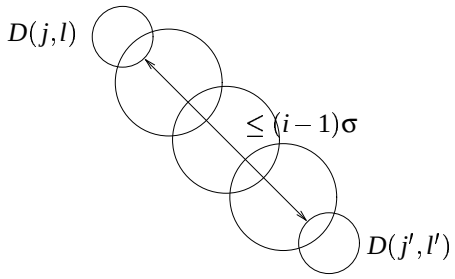


Figure 2.16: Disks $D_{j,l}$ and $D_{j',l'}$

Now consider two disks $D_{j,l}$ and $D_{j',l'}$ in D with coordinates j,l and j',l', respectively. Let $a_i = \lfloor \frac{(i-1)\sigma+1}{t\sqrt{2}} + 1 \rfloor$ for $i = 2,\ldots,k$. Let i be the minimum such that $|j-j'| \leq a_i$ and $|l-l'| \leq a_i$. Then, $D_{j,l}$ and $D_{j',l'}$ are at plane distance at most $(i-1)\cdot\sigma$. We construct a set $D(j,l,j',l')$ of $(i-1)$ disks of diameter σ which will connect $D_{j,l}$ and $D_{j',l'}$ by a path of length at most i. For an illustration see Figure 2.16 on the preceding page. In other words, in a σ-disk graph $G(j,l,j',l')$ given by $D \cup D(j,l,j',l')$ the vertices of disks $D_{j,l}$ and $D_{j',l'}$ are at graph distance i.

Let A be an online $L_{(p_1,\ldots,p_k)}$-labeling algorithm for the class of σ-disk graphs of given disk representation. For any σ-disk graph and any order of the disks, A always outputs a feasible $L_{(p_1,\ldots,p_k)}$-labeling.

We present the disks of D in an arbitrary order to A. For some i from $\{2,\ldots,k\}$, let $D_{j,l}$ and $D_{j',l'}$ be any two disks in D such that $|j-j'| \leq a_i$ and $|l-l'| \leq a_i$. If A assigns the labels to $D_{j,l}$ and $D_{j',l'}$ which differ by at most $p_i - 1$, then we add the disks of $D(j,l,j',l')$ to D. In this case, A outputs a non-feasible labeling for σ-disk graph $G(j,l,j',l')$ given by $D \cup D(j,l,j',l')$, that is a contradiction.

In total, for each $i = 2,\ldots,k$, and for any two disks from set $D_i = \{D_{j,l} \mid 1 \leq j,l \leq a_i\}$ of a_i^2 disks, A assigns the labels which differ by at least p_i. As in Lemma 2.5.2, for each $i = 2,\ldots,k$ the maximum label used by A is at least

$$1 \;+\; p_i \cdot (a^2 \;-\; 1) \;=\; 1 + \left(\left\lfloor \frac{(i-1)\sigma+1}{t\sqrt{2}} + 1 \right\rfloor^2 - 1 \right).$$

In total, the maximum label used by A for a σ-disk graph G given by D is at least

$$1 \;+\; \max_{i=2,\ldots,k} \left\{ \left(\left\lfloor \frac{(i-1)\sigma+1}{t\sqrt{2}} + 1 \right\rfloor^2 - 1 \right) p_i \right\},$$

and for $t = \frac{3}{2\sqrt{2}}$

$$\bar{\rho} \;=\; 1 \;+\; \frac{\sigma^2}{9} \max_{i=2,\ldots,k} \{ i^2 \cdot p_i \}.$$

From another side, $\chi_{(p_1,\ldots,p_k)}(G) = 1$. Hence, A cannot be better than $\bar{\rho}$-competitive. □

From Theorem 2.4.3 and Theorem 2.5.5 we have the following result:

Corollary 2.5.6. *For any k-tuple $(p_1,\ldots,p_k)$ of distance constraints ($k \geq 2$), and any $\sigma \geq 1$, the competitive ratio of the algorithm* ODL *is at most $O(\log k)$ times larger than the competitive ratio of any online $L_{(p_1,\ldots,p_k)}$-labeling algorithm for the class of σ-disk graphs with at least one edge and given disk representation. Therefore, the algorithm* ODL *is asymptotically optimal.*

Proof. Take a set D of unit disks as described in the proof of Theorem 2.5.5. Add a pair of new intersecting disks. These two disks intersect no disk in D.

Let G be a σ-disk graph given by D and the new disks. There is only one edge in G. We can use label 1 for all disks in D, and use labels 1 and p_1+1 for the new disks. Hence, we can show that

$$\chi_{(p_1,\ldots,p_k)} \;=\; p_1 \;+\; 1.$$

Then, following the proof of Theorem 2.5.5 we can show that a lower bound on the competitive ratio of any online algorithm is at least

$$\frac{1+\frac{\sigma^2}{9}\max_{i=2,\ldots,k}\{i^2p_i\}}{1+p_1} \;\geq\; c\cdot\frac{\sigma^2\max_{i=2,\ldots,k}\{i^2p_i\}}{1+p_1}, \tag{2.7}$$

where c is some suitable constant which neither depends on σ nor $(p_1,\ldots,p_k)$.

From another side, by using Theorem 2.3.1 and Theorem 2.4.3, we can show that an upper bound on the competitive ratio of our algorithm ODL is at most

$$\begin{aligned}\frac{2\ell^*}{1\;+\;p_1} &= 2\cdot\frac{1\;+\;6(4(2p_1\;-\;1)\;+\;\sum_{m=2}^{a}(m\;+\;1)\cdot(2p_{\lceil\frac{3m\;-\;4}{4\sigma}\rceil}\;-\;1))}{1\;+\;p_1}\\ &\leq\; c'\cdot\frac{\sigma^2\sum_{i=2}^{k}ip_i}{1+p_1}+O(1),\end{aligned} \tag{2.8}$$

where c' is some suitable constant which also neither depends on σ nor $(p_1,\ldots,p_k)$.

Let $s\geq 2$ be such that $p_i\leq(\frac{s^2}{i^2})\cdot p_s$ for all $i=2,\ldots,k$. Here $s\in\{2,\ldots,k\}$ delivers the maximum to $i^2\cdot p_i$. Then,

$$\sum_{i=2}^{k} i\cdot p_i \leq \sum_{i=2}^{k}\left(\frac{s^2}{i}\right)\cdot p_s \;=\; s^2\cdot p_s\left(\sum_{i=2}^{k}\frac{1}{i}\right) \;\leq\; \max_{i=2,\ldots,k}\{i^2p_i\}\cdot O(\log k). \tag{2.9}$$

Indeed, we can combine (2.7), (2.8) and (2.9). This will show that the competitive ratio of our algorithm OLD is at most $O(\log k)$ times the competitive ratio of any online $L_{(p_1,\ldots,p_k)}$-labeling algorithm. □

2.6 OFFLINE LABELING OF UNIT DISK GRAPHS

Here we explore the offline version of the distance-constrained labeling problem in the case when $k=2$ and distance constrains $(p_1,p_2)=(2,1)$. We deal with unit disk graphs. First, we consider the case when the disk representation of unit disk

graphs is given, and present a simple approximation algorithm which is based on the so-called *cutting* technique. Then, we present a robust algorithm, i.e., it does not require the disk representation and either outputs a feasible labeling, or shows that the input graph is not a unit disk graph.

2.6.1 Cutting Technique and Strip Graphs

The main idea of our cutting technique is rather simple: We "cut" the plane into strips of small width. Then, we take a unit disk graph and split it into several "strip" unit disk graphs which are induced by the strips. Finally, we label each strip disk graph, and combine all these together into one labeling for the original unit disk graph.

A unit disk graph G is called a $\frac{1}{\sqrt{2}}$-*strip unit disk graph* if there is a mapping $f\colon V(G) \to \mathbb{R} \times [0, \frac{1}{\sqrt{2}}]$ such that $(u,v) \in E(G)$ iff $\mathrm{dist}_{\mathcal{E}}(f(u), f(v)) \leq 1$. Informally, G is given by a set D of unit disks such that each disk from D has its center in a *strip* of width $\frac{1}{\sqrt{2}}$. For an illustration see Figure 2.17.

We will use the following simple properties which were mentioned in the introduction. Let G be a graph. Let G^2 be the 2nd power of G, i.e. a graph which arises from G by adding the edges which connect all vertices at graph distance 2. Then, a coloring of G^2 is an $L_{(1,1)}$-labeling of G and vise versa, i.e.

$$\chi_{(1,1)}(G) \;=\; \chi(G^2).$$

Furthermore, by multiplying all labels in an $L_{(1,1)}$-labeling for G by 2 we can obtain an $L_{(2,2)}$-labeling for G, i.e.

$$\chi_{(2,1)}(G) \;\leq\; \chi_{(2,2)}(G) \;\leq\; 2 \cdot \chi_{(1,1)}(G).$$

For an illustration see Figure 2.18.

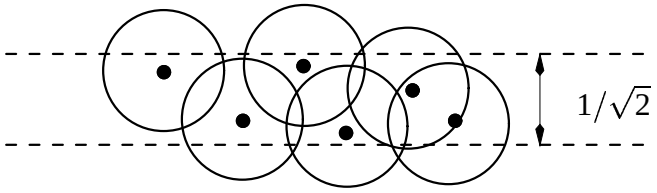


Figure 2.17: A $\frac{1}{\sqrt{2}}$-strip unit disk graph

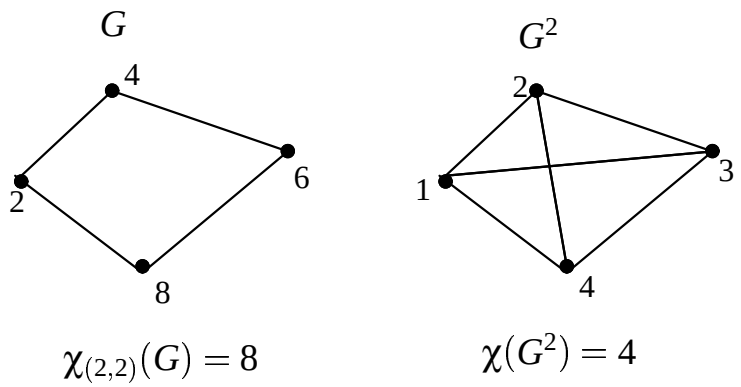


Figure 2.18: An $L_{(2,2)}$-labeling of G and a coloring of G^2

2.6.2 Coloring and Labeling of Strip Graphs

We start with the following result:

Lemma 2.6.1. *Let G be a $\frac{1}{\sqrt{2}}$-strip unit disk graph and let v be a vertex such that the unit disk corresponding to v has the least x-coordinate. Then, for G^2, the cardinality of the vertex set*

$$N_{G^2}(v) \;=\; \{u \in V(G) - \{v\} : \mathrm{dist}_G(u,v) \leq 2\}$$

is at most $3\omega(G) - 1$.

Proof. There is a *strip* of width $\frac{1}{\sqrt{2}}$, and each vertex v in G corresponds to a unit disk D_v with the center in this strip. Let v be a vertex in G which unit disk D_v has the smallest x-coordinate. For an illustration see Figure 2.19 on the following page.

Consider all vertices u in $V(G)$ which are at graph distance at most 2 from v, i.e. $\mathrm{dist}_G(u,v) \leq 2$. Then, for each such u, the x-coordinate of disk D_u and disk D_v differ by at most 2, see Figure 2.19 a) and b).

Consider all disks in a square of side $\frac{1}{\sqrt{2}}$, see Figure 2.19 b). Clearly, all of them intersect in pairs. This forms a clique in G. Hence, we can bound the maximum number of the disks in a square by $\omega(G)$.

Consider all disks D_u in a rectangle R of width $\frac{1}{\sqrt{2}}$ and length 2, see Figure 2.19 c). It can be covered by three squares of width $\frac{1}{\sqrt{2}}$. Hence the maximum number of disks in R is at most $3\omega(G)$.

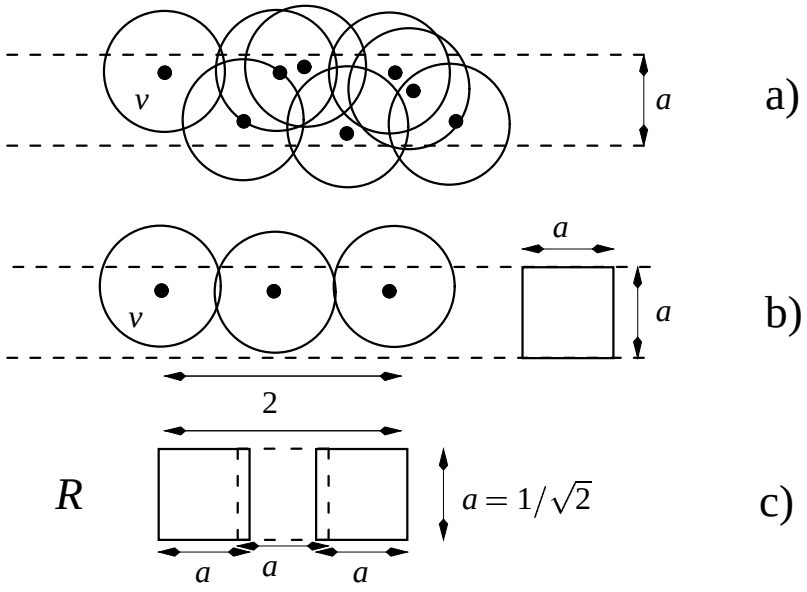


Figure 2.19: A $\frac{1}{\sqrt{2}}$-strip unit disk graph

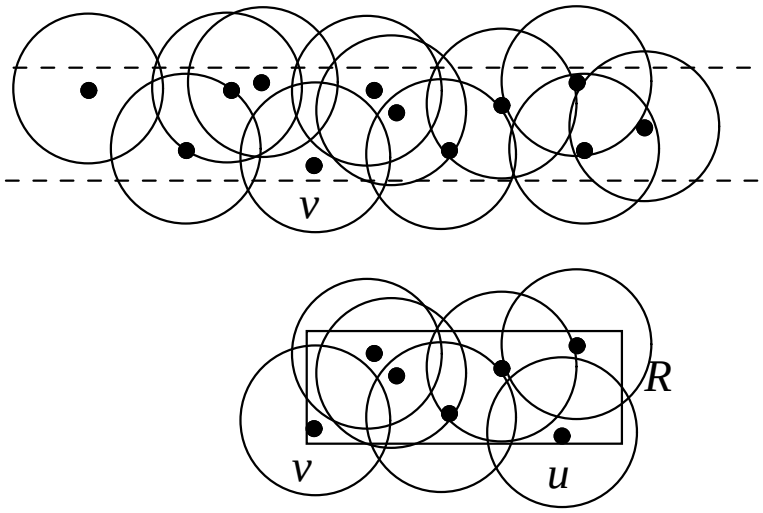


Figure 2.20: A vertex $v \in V(G)$ and a vertex $u \in N_{G^2}(v)$

Consider vertices u from $N_{G^2}(v)$. Each u is at graph distance at most 2 from v in G. Hence, each disk D_u is in a rectangle R having the center of disk D_v on its left side. For an illustration see Figure 2.20. Excepting disk D_v the number of such disks D_u in R is at most $3\omega(G)-1$. Hence, we can bound $|N_{G^2}(v)|$ by $3\omega(G)-1$. □

Let G be $\frac{1}{\sqrt{2}}$-strip unit disk graph. Let D_v be the disk of $v \in V(G)$. We order vertices v in $V(G)$ such that the x-coordinate of disks D_v does not increase. If $|V(G)| = n$, then such an *decreasing* order $\prec$ for the vertices of $V(G)$ can be found in $O(n \log n)$ time.

Informally, given a vertex v and all vertices u in $V(G)$ such that $v \prec u$, disk D_v has the least x-coordinate within all disks D_u. For an illustration see Figure 2.20. Then, by using Lemma 2.6.1, for each vertex v we can bound the number of such vertices u in $N_{G^2}(v)$ by $3\omega(G)-1$.

This helps in the following coloring algorithm:

FIRST FIT COLORING (FFC):
Input: A $\frac{1}{\sqrt{2}}$-strip unit disk graph G,
Output: A coloring of G^2.
Select vertices v from $G(V)$ in a *decreasing* order $\prec$ while coloring with an initial sequence of colors $1, 2, \ldots$. Assign the vertex v the least color that has not already been assigned to any vertex u adjacent to v in G^2.

Lemma 2.6.2. *The maximum color used by the algorithm* FFC *is bounded by* $3\omega(G)$.

Proof. For the first vertex in the order the algorithm FFC uses color 1. Then, for each next vertex v the algorithm FFC assigns the least color which is not used for vertices u in $N_{G^2}(v)$. As we know, the number of colored vertices u in $N_{G^2}(v)$ is bounded by $3\omega(G)-1$. Hence, FFC only uses colors from $\{1, 2, \ldots, 3\omega(G)\}$. □

Now we can give the following simple labeling algorithm:

STRIP LABELING (SL):
Input: A $\frac{1}{\sqrt{2}}$-strip unit disk graph G,
Output: An $L_{(2,1)}$-labeling of G.
1. Find an $L_{(1,1)}$-labeling for G.
2. Multiply all labels by 2.

Lemma 2.6.3. *The maximum label used by the algorithm* SL *is bounded by* $6\omega(G)$. *Furthermore, all labels used are even.*

Proof. By Lemma 2.6.2 we can color G^2 with at most $3\omega(G)$ colors. This gives a feasible $L_{(1,1)}$-labeling for G. Then, we multiply all labels by 2. This gives a feasible $L_{(2,2)}$-labeling for G which is also a feasible $L_{(2,1)}$-labeling for G. Thus, all labels used are even, and the maximum label used is at most $2 \cdot (3\omega(G)) = 6\omega(G)$. □

2.6.3 Cutting of Unit Disk Graphs

Now we are ready to describe an approximation algorithm for labeling of unit disk graphs. W.l.o.g. we assume that a unit disk graph G is connected and has at least one edge, i.e. $\omega(G) \geq 2$.

Given a unit disk graph G, we partition the plane into $k = O(|V(G)|)$ strips S_0, S_1, ..., S_k of width $\frac{1}{\sqrt{2}}$. Strip S_0 contains a disk with the most y-coordinate and and S_k contains a disk the least y-coordinate. All other strips are numbered from top to bottom, respectively. For an illustration see Figure 2.21 on the next page. This partition induces a partition of G into $\frac{1}{\sqrt{2}}$-strip unit disk graphs $G_0, \ldots, G_k$. In the case of disks with centers in two strips ties are broken arbitrarily.

Our main idea is as follows. Consider consecutive strips S_0, S_1, S_2 and S_3, S_4, S_5. The width of each strip is $\frac{1}{\sqrt{2}}$, and the width of two consecutive strips $\sqrt{2}$ is larger than the diameter of a unit disk. Thus, two disks in S_0, S_1, S_2 or S_3, S_4, S_5 can intersect. However, no disk in S_0 (S_1,S_2) can intersect with a disk in S_3 (S_4, S_5), see Figure 2.21.

We are interested in an $L_{(2,1)}$-labeling. Hence, any two vertices in $\cup_{i=1}^{3} G_i$ or in $\cup_{i=3}^{5} G_i$ may require their labels be different by 2, and any vertex in G_0 (G_1,G_2) and any vertex in G_3 (G_4, G_5) may require their labels be different by 1. By using the algorithm SL we find an $L_{(2,1)}$-labeling for each G_i, $i = 0, \ldots, 5$. By Lemma 2.6.3, we can bound the maximum label used as $\max_i \omega(G_i) \leq \omega(G)$. Furthermore, all labels are even.

To obtain a feasible $L_{(2,1)}$-labeling for $\cup_{i=1}^{3} G_i$, we let the labels of G_0 be the same (increase by 0), and increase the labels of G_1 and G_2 by $6\omega(G)$ and $12\omega(G)$, respectively. This defines all labels be even, and any two labels be different by at least 2. To obtain a feasible $L_{(2,1)}$-labeling for $\cup_{i=3}^{5} G_i$, we decrease the labels of G_3 by 1 (increase by -1), and increase the labels of G_4 and G_5 by $6\omega(G) - 1$ and $12\omega(G) - 1$, respectively. (Remember $\omega(G) \geq 2$.) This defines all labels be odd, and any two labels of $\cup_{i=3}^{5} G_i$ be different by at least 2. Finally, we simply

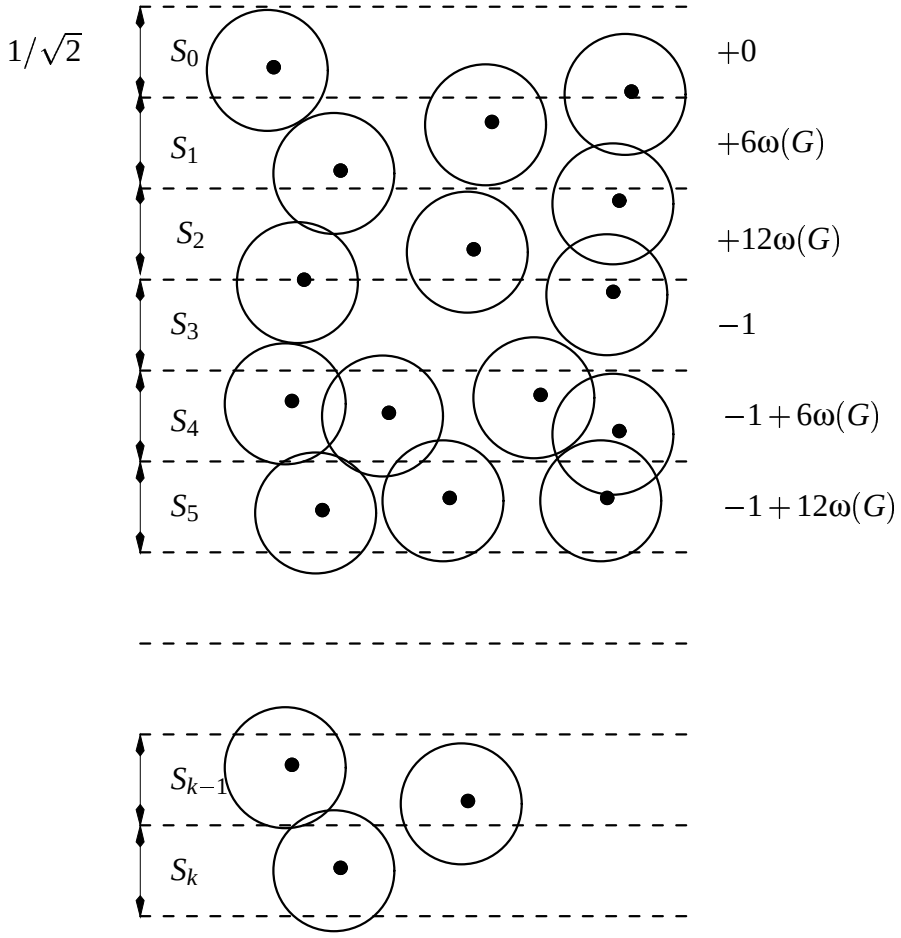


Figure 2.21: Strips $S_0, S_1 \ldots, S_k$

combine both parts. Since the labels of $\cup_{i=1}^{3} G_i$ are even and the labels of $\cup_{i=3}^{5} G_i$ are odd, it holds that any vertex in G_0 (G_1,G_2) and any vertex in G_3 (G_4, G_5) differ by 1. Hence, we have found a feasible $L_{(2,1)}$-labeling for $\cup_{i=0}^{5} G_i$.

By generalizing this idea we present the final algorithm:

CUTTING DISTANCE LABELING (CDL):
Input: A unit disk graph G,
Output: An $L_{(2,1)}$-labeling for G.
1. Partition the plane into $k = O(V(G))$ strips $S_0, \ldots, S_k$ of width $\frac{1}{\sqrt{2}}$.
2. For each $i \in \{0, \ldots, k\}$ find an $L_{(2,1)}$-labeling of G_i.
3. Change the labels of graph G_i by adding integer $\#_{(i \bmod 6)}$, where

$$(\#_0, \ldots, \#_5) = (0,\ 6\omega(G),\ 12\omega(G),\ -1,\ 6\omega(G)-1,\ 12\omega(G)-1).$$

Theorem 2.6.4. *The maximum label used by the algorithm* CDL *is at most* $18\omega(G)$.

Proof. By Lemma 2.6.3, the maximum label used on every G_i ($i = 1, \ldots, k$) is at most $6\omega(G)$. Hence, the maximal label assigned by the algorithm CDL is at most $12\omega(G) + 6\omega(G)$. □

Corollary 2.6.5. *The approximation ratio of the algorithm* CDL *is bounded by* 12*, and the bound tends to* 9 *as the clique number* $\omega(G)$ *of unit disk graphs grows to infinity.*

Proof. W.l.o.g. we can assume that $\omega(G) \geq 2$. Then, in order to label a clique of size $\omega(G)$ we must use the maximum label at least $1 + p_1(\omega(G) - 1)$, where $p_1 = 2$. Thus, by Theorem 2.6.4, the approximation ratio of CDL is bounded by

$$\frac{18\omega(G)}{2\omega(G) - 1}.$$

For $\omega(G) = 2$, the bound is equal to 12. If $\omega(G)$ grows to infinity, then the bound tends to 9. □

As the last note, it is not hard to observe that $\frac{1}{\sqrt{2}}$-strips were used in the description of the algorithm to simplify the explanation. To avoid irrational numbers, $\frac{1}{\sqrt{2}}$-strips in the algorithm can be replaced by c-strips, where c is any rational number between $\frac{2}{3}$ and $\frac{1}{\sqrt{2}}$.

2.6.4 Robust Algorithms

Here we present an approximation labeling algorithm which does not need the disk representation of a unit disk graph as a part of the input. (Recall that it is NP-hard to recognize unit disk graphs.)

An algorithm which solves an optimization problem on a class $\mathcal{C}$ of inputs is called *robust* if it satisfies the following conditions [RS01]:

1. Whenever the input is in $\mathcal{C}$, the algorithm finds the correct solution.
2. If the input is not in $\mathcal{C}$, then the algorithm either finds the correct solution, or reports that the input is not in $\mathcal{C}$.

Based on the ideas of [CCJ90], a robust algorithm computing the maximal clique of a unit disk graph is given in [RS01]. Every unit disk graph has an edge ordering $e_1 \prec_e \cdots \prec_e e_m$ such that for every edge e_i the neighbors of its endpoints induce a cobipartite subgraph C_i (i.e., the complement of a bipartite graph) of a graph induced by $\{e_1, \ldots, e_i\}$. If such an ordering $\prec_e$ exists, then each clique is contained in the cobipartite graph C_i for some edge e_i. The robust algorithm first constructs (if any exists) an edge ordering $\prec_e$ in time $O(m^2n)$, and then the algorithm finds a maximal clique in each graph C_i. This is equivalent to finding the maximum independent set in a bipartite graph which can be done in $O(m\sqrt{n})$ time by using the matching technique [HK73]. Therefore, the running time of the entire algorithm is $O(m^2n)$.

Let G be a unit disk graph and let G^2 be the 2nd power of G, i.e. a graph which arises from G by adding the edges which connect all vertices at graph distance 2. Then, we can prove the following simple result:

Lemma 2.6.6. *Every unit disk graph G has a vertex v such that the set*

$$N_G(v) = \{u \neq v \colon \{u, v\} \in E(G)\}$$

contains at most $3\omega(G) - 3$ *vertices and the set*

$$N_{G^2}(v) - N_G(v)$$

contains at most $11\omega(G)$ *vertices.*

Proof. Let G be a unit disk graph. Let D_v be the unit disk of $v \in V(G)$. Then, we can select a vertex v such that D_v has the least y coordinate. For an illustration see Figure 2.22 on the following page.

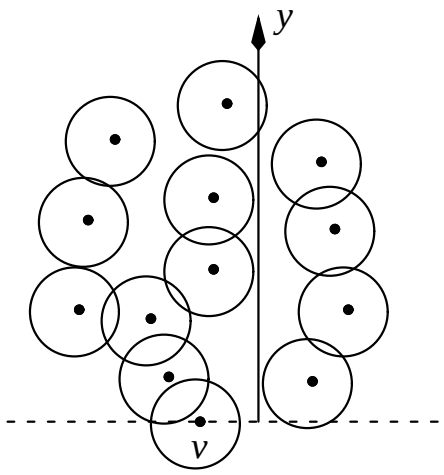


Figure 2.22: A vertex v with the least y-coordinate

Now consider the sector partition around v depicted in Figure 2.23. There are 14 sectors S_i, $i = 1,\ldots,14$. Consider a vertex u in $V(G)$. We say D_u is in S_i ($i = 1,\ldots,14$) if its center in S_i. To break ties, any disk on a border of two sectors is in the sector with smaller index.

Then, we have the following property. If $u \in N_G(v)$, i.e. D_u intersects D_v, then D_u in one of sectors S_i, $i = 1,2,3$. If $u \in N_{G^2}(v) - N_G(v)$, i.e. there is a disk which intersects D_v and D_u, then D_u in one of sectors S_i, $i = 4,\ldots,14$.

The sectors are constructed such that any two unit disks in one sector intersect. Thus, for each sector S_i, $i = 1,\ldots,14$, vertices u from $V(G)$ with disks D_u in S_i form a clique. Hence, for each sector S_i, $i = 1,2,3$, we can bound the number of the disks by $\omega(G) - 1$ (excepting our D_v), and for each sector S_i, $i = 4,\ldots,14$,

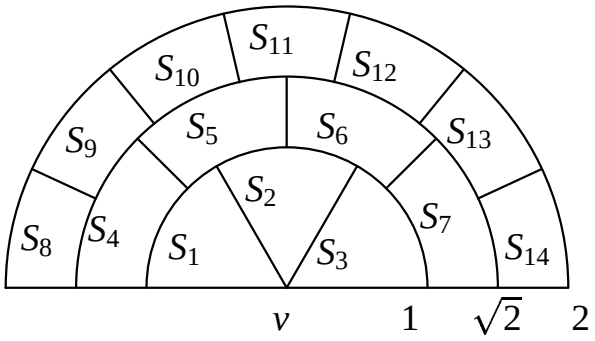


Figure 2.23: The sector partition around a vertex v

we can bound the number of disks by $\omega(G)$. In total, we can bound $|N_G(v)|$ by $3(\omega(G)-1)$, and $N_{G^2}(v)-N_G(v)$ by $(14-3)\omega(G)$. □

We say that a vertex ordering $v_1 \prec \cdots \prec v_n$ of G is *good* if for every $2 \leq i \leq n$: (i) $|N_G(v_i) \cap \{v_1,\ldots,v_{i-1}\}| \leq 3\omega(G)-3$; (ii) $|(N_{G^2}(v_i)-N_G(v_i)) \cap \{v_1,\ldots,v_{i-1}\}| \leq 11\omega(G)$.

Notice, that by Lemma 2.6.6 every unit disk graph has a good vertex ordering. Also, for a graph G one can either find a good vertex ordering, or conclude that there is no good ordering for G. Furthermore, if G has n vertices, this can be done in $O(n^3)$ time.

Now we are ready to present a robust $L_{(2,1)}$-labeling approximation algorithm for unit disk graphs. The algorithm described below, called RDL, does not use the disk representation of a unit disk graph G. It either concludes that G is not a unit disk graph, or it finds an $L_{(2,1)}$-labeling of G.

ROBUST DISTANCE LABELING (RDL):
Input: A graph G given as an adjacency list.
Output: An $L_{(2,1)}$-labeling c of $V(G)$, or the conclusion that G is not a unit disk graph.
1. Run the robust algorithm to compute $\omega(G)$. This algorithm either computes $\omega(G)$ or concludes that G is not a unit disk graph. 2. Find a good vertex ordering $v_1 \prec \cdots \prec v_n$. If there is no such an ordering, then conclude that G is not unit disk graph.
3. Label vertices sequentially in the order $\prec$ as follows:
3a. Assume that vertices $v_1,\ldots,v_{i-1}$ are already labeled.
3b. Let $\lambda \geq 1$ be the smallest integer which is not used as a label of vertices in $N_{G^2}(v_i) \cap \{v_1,\ldots,v_{i-1}\}$ nor is a member of the set

$$\bigcup_{j<i:\, v_j \in N_G(v_i)} \{c(v_j)-1, c(v_j), c(v_j)+1\}.$$

3c. Label v_i by $c(v_i)=\lambda$.

Theorem 2.6.7. *For any graph G, the algorithm* RDL *either produces an $L_{(2,1)}$-labeling for G with the maximum label at most $20\omega(G)-8$, or concludes that G is not a unit disk graph.*

Proof. Suppose that the algorithm RDL outputs that G is not a unit disk graph. If it occurs after the first step, then G has no edge ordering $\prec_e$ and therefore is not

a unit disk graph. If the algorithm halts at the second step, then its conclusion is verified by Lemma 2.6.6.

Suppose that RDL outputs a labeling. Let us first show that the maximum label used by the algorithm is not larger than $20\omega(G)-8$. We proceed by induction. The vertex v_1 is labeled by 1, since both sets declared in 3b are empty. Suppose that we have labeled vertices $v_1,\ldots,v_{i-1}$. We need to assign a label to v_i. If a neighbor of v_i has a label x then labels $x-1,x$ and $x+1$ are "forbidden" for v_i. If a vertex at distance two from v_i has a label x then x is "forbidden" for v_i. By (i), v_i has at most $3\omega(G)-3$ labeled vertices in $N_G(v_i)$. By (ii), there are at most $11\omega(G)$ labeled vertices in $N_{G^2}(v_i)-N_G(v_i)$. Hence, the total number of "forbidden" labels for v_i is at most

$$3\cdot(3\omega(G)\ -\ 3)\ +\ 11\omega(G)\ =\ 20\omega(G)\ -\ 9.$$

Since there are $20\omega(G)-8$ labels, it holds $c(v_i)\leq 20\omega(G)-8$. □

Corollary 2.6.8. *The approximation ratio of the algorithm* RDL *is bounded by* $\frac{32}{3}\approx 10.67$*, and the bound tends to* 10 *as the clique number* $\omega(G)$ *of unit disk graphs grows to infinity.*

Proof. W.l.o.g. we can assume that $\omega(G)\geq 2$. Then, in order to label a clique of size $\omega(G)$, the maximum label used is at least $1+p_1(\omega(G)-1)$, where $p_1=2$. Thus, by Theorem 2.6.7, the performance ratio of RDL is bounded by

$$\frac{20\omega(G)-8}{2\omega(G)-1}.$$

For $\omega(G)=2$, the bound is equal to $\frac{32}{3}\approx 10.67$. If $\omega(G)$ grows to infinity, then the bound tends to 10. □

We observe that the knowledge of geometrical disk representation is crucial for the construction of online algorithms with constant competitive ratio on unit disk graphs. For the coloring problem similar robust algorithm on unit disk graphs can be turned into online coloring algorithm (with worse competitive ratio). However, this is not the case for the labeling problem. The main reason why RDL cannot be turned into a "First-Fit" algorithm is that at the moment when we have to select a suitable label for a vertex v_i we need the corresponding information about all vertices from set $\{v_1,\ldots,v_{i-1}\}$ which are at distance two from v_i in G. Unfortunately, this information cannot be fully derived from G restricted to $\{v_1,\ldots,v_i\}$.

2.7 GENERAL OFFLINE LABELING OF σ-DISK GRAPHS

Here we discuss an offline labeling algorithm for σ-disk graphs. We assume that the disk representation of σ-disk graphs is not given. We will need the following simple result:

Lemma 2.7.1. *For each vertex v in a σ-disk graph G, the set*

$$N_G^{(k)}(v) = \{u \neq v \colon \mathrm{dist}_G(u,v) \leq k\}$$

consists of at most $(8k)^2\sigma^2\omega(G)$ vertices.

Proof. Let D_v be the disk for $v \in V(G)$. Assume w.l.o.g. that the smallest disk diameter is equal to 1, and the largest disk diameter is equal to σ.

Take a vertex $v \in V(G)$ and consider $u \in N_G^{(k)}(v)$. The centers of D_v and D_u are at plane distance at most $k\sigma$ from each other. For illustration see Figure 2.24 on the following page.

Consider a square S of width $4k\sigma$. We put the center of S at the center of D_v. Then, all disks D_u, $u \in N_G^{(k)}(v)$, fall into S. Next, we partition S into $(4)^2(2)^2k^2\sigma^2$ small squares of width $1/2$. For an illustration see Figure 2.25 on the next page. Any two disks that fall into a small square intersect. Hence, the set of vertices $u \in N_G^{(k)}(v)$ which have disks D_u in one small square form a clique. Thus, the number of vertices in any such set is bounded by the maximum clique number $\omega(G)$. In total, we can bound $|N_G^{(k)}(v)|$ by $(8)^2k^2\sigma^2\omega(G)$. □

Consider the following algorithm:

FIRST FIT LABELING (FFL):
Input: A σ-graph G in an adjacency list, and a k-tuple $(p_1, p_2, \ldots, p_k)$ of distance constraints.
Output: An $L_{(p_1,\ldots,p_k)}$-labeling for G.
For each $v \in V(G)$ find $N_G^{(k)}(v)$. Select vertices v from $G(V)$ in an arbitrary order while labeling with an initial sequence of labels $1, 2, \ldots$. Assign the vertex v the least feasible label which respects $(p_1, \ldots, p_k)$.

Now we can prove the following main result.

Theorem 2.7.2. *The algorithm* FFL *is an $O(k^2\sigma^2)$-approximate $L_{(p_1,\ldots,p_k)}$-labeling algorithm for the class of σ-disk graphs.*

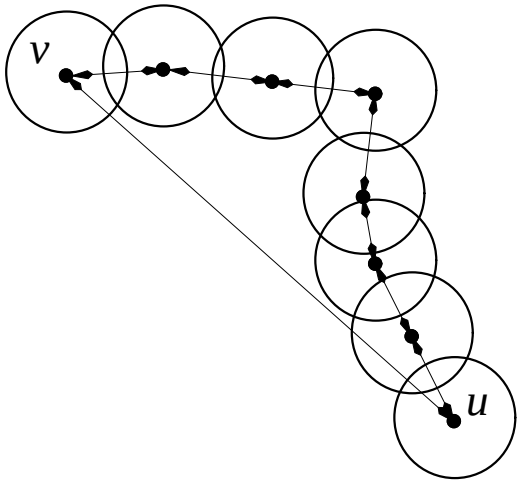


Figure 2.24: Vertices v and v'

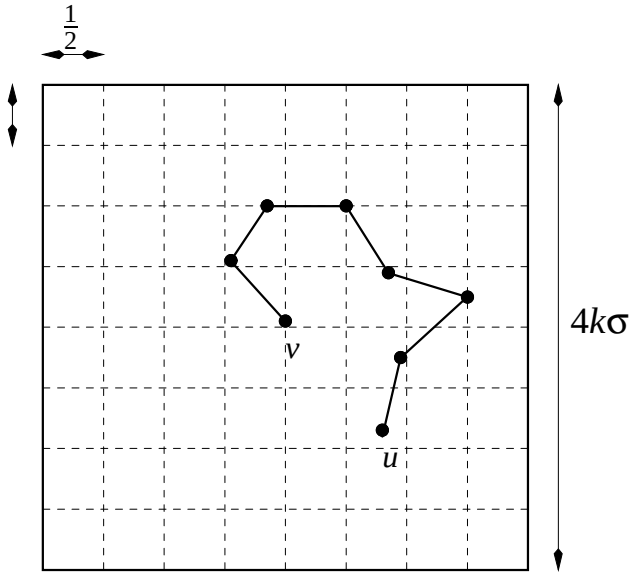


Figure 2.25: A square S at a vertex v

Proof. Let G be a σ-disk graph. Assume w.l.o.g. that the clique number $\omega(G) \geq 2$. By Lemma 2.7.1, for any vertex v the number of vertices in $N_G^{(k)}(v)$ is bounded by $(8k)^2\sigma^2\omega(G)$. Even if any two labels for $u \in N_G^{(k)}(v)$ differ by $2p_1$, that is more than $p_1 \geq p_2 \geq \ldots \geq p_k$, the the label assigned to v by FFL is most

$$1 + 2p_1((8k)^2\sigma^2\omega(G)).$$

From another side, in labeling a clique of size $\omega(G)$ the maximum label is at least

$$1 + p_1(\omega(G) - 1).$$

Since $\omega(G) \geq 2$, the approximation ratio of FFL is bounded by

$$\frac{1 + 2p_1((8k)^2\sigma^2\omega(G))}{1 + p_1(\omega(G) - 1)} = O(k^2\sigma^2).$$

□

Finally, we can notice that for each vertex v in $V(G)$ we find $N_G^{(k)}(G)$. Hence, FFL is not an online algorithm. Furthermore, the approximation guarantee holds only for σ-disk graphs. Due to the NP-hardness of the σ-disk graph recognition problem, this assumption is essential. However, it remains an open question whether our algorithm can be turned into a robust one.

2.8 CONCLUDING REMARKS

The distance constrained labeling problem, which is a natural generalization of the coloring problem, has only recently received increased attention. There were only few known results on the approximability of the problem and on the online version of the problem. In this chapter we considered the distance constrained labeling problem for the class of disk graphs. This model is related to the frequency assignment problem in radio and mobile telephony networks. We presented a number of approximation and online algorithms for different variants of disk graphs and distance constraints, obtaining the first results in this direction. We derive several new techniques, e.g. hexagonal tiling, circular labeling, plane cutting and neighborhood sectoring. These techniques are quite general and can be used in designing of online and offline algorithms for many other variants of the labeling problem. In fact, our results for $L_{(2,1)}$-labeling can be simply extended for $L_{(p,1)}$-labeling. Furthermore, all our techniques are very simple and do not require larger computational resources. The realization of our online labeling algorithm can be found in [Maš01].

Indeed, many interesting questions still remain. There is a number of known results regarding the complexity of several variants of the problems, but the complexity of the general labeling labeling problem, previously studied in [Fia00], and the complexity of $L_{(p_1,p_2)}$-labeling for planar graphs is still open. Regarding disk graphs, it is still unclear how powerful is the information about the geometrical disk representation. Though this information is important for online coloring and labeling algorithms, even in very simple cases, we can find offline robust algorithms which can avoid using of the disk representation. However, we think that one should first address the question of approximability of the maximum clique in the disk graphs. Regarding distance constrains, there are a number of results on $L_{(2,1)}$-labeling for different graph classes. However, there are no known results on $L_{(3,2,1)}$-labeling, that can be a natural generalization. We believe that our techniques can be quite useful here. Finally, one of the main questions concerns real-world applications, which have not been widely discussed so far. Anyway, we believe that in the near future there will be a great demand for algorithms solving both offline and online versions of the labeling problem.

CHAPTER 3

SCHEDULING OF DEDICATED MULTIPROCESSOR TASKS

Optical networks employing wavelength division multiplexing (WDM) are now a viable technology for implementing a next-generation network infrastructure that will support a diverse set of existing, emerging, and future applications [GLM+00]. WDM technology initially was deployed in point-to-point links in *wide area* distances [WASG96]. However, the work on WDM *local area* networks has also been currently under way, see e.g. [Hel00, KFH+00]. These networks are also known as *single-hop* WDM networks [Muk92].

The main future of broadcast WDM local area networks is the so-called *one-to-many transmission* or *multicasting* ability [APE97]. For an illustration see Figure 3.1 on the following page. That is, a transmission by a node in such a network on a given channel (wavelength) is received by *all* nodes listening on that channel at that point in time [SS00].

Since the number of channels may be less than the number of nodes and two or more nodes may want to send data packets to the same destination node, coordination among nodes that wish to communicate with each other is required. Many *transmission protocols* for coordinating *multicast* data transmissions have been proposed in the literature (see [TR00] for a survey). The approaches used range from sending a different copy of each data packet to each one of the corresponding destinations (*unicast service, or multicast service with full fanout splitting*), to transmitting a single copy of the data packet to all the destinations at once (*multicast service with no fanout splitting*).

Due to the relation to the later approach, here we address the *dedicated* variant of the multiprocessor tasks scheduling problem. We are given a set of n tasks $T = \{1,\dots,n\}$ and a set of m processors $M = \{1,2,\dots,m\}$. Each task $j \in T$ has a processing time p_j, a release date r_j and a prespecified set of processors $\mathrm{fix}_j \subseteq M$. Preemptions are not allowed. Each processor can work on at most one task at a time, each task must be processed simultaneously by all processors of fix_j. The objective is to minimize the *makespan* $C_{\max} = \max_j C_j$, where C_j denotes the completion time of task j.

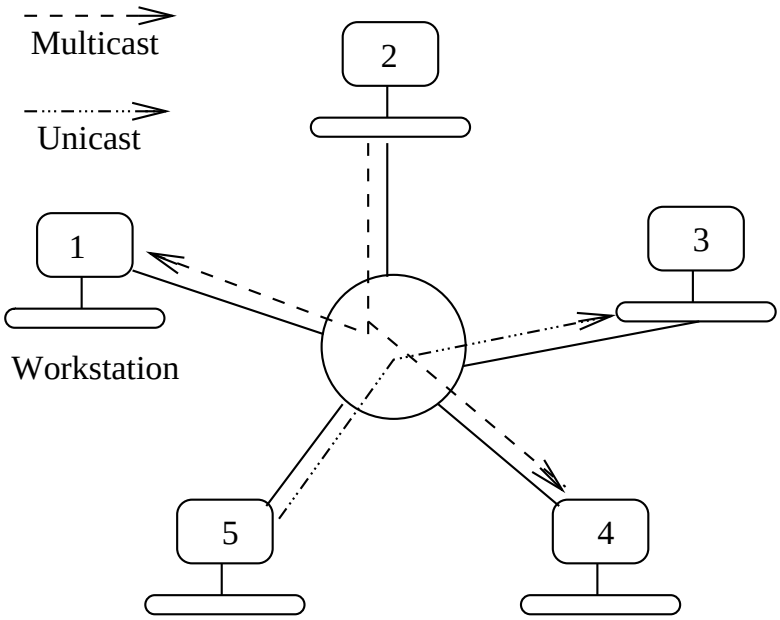


Figure 3.1: A passive-star-based local optical WDM network

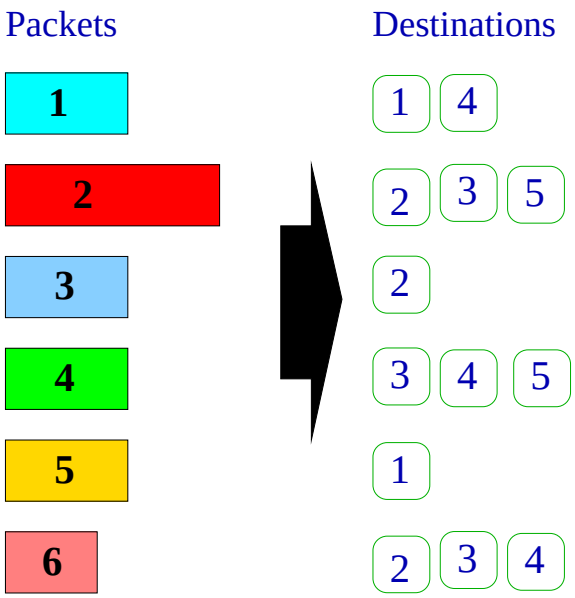


Figure 3.2: Multi-destination data packets

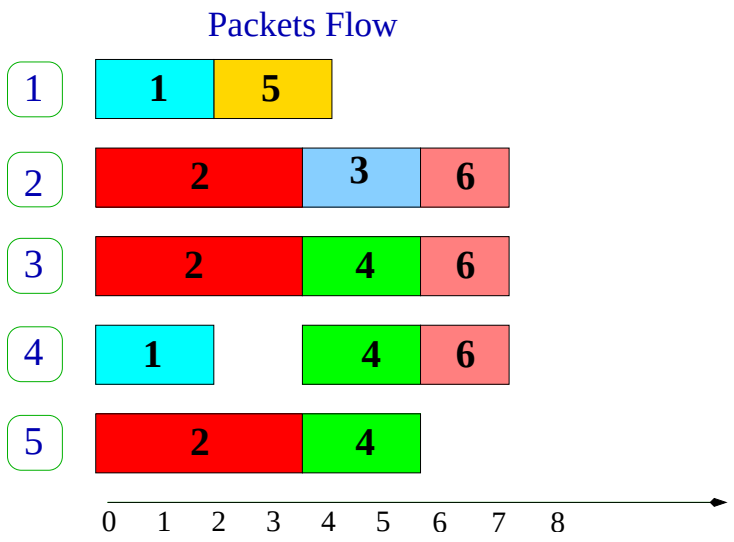


Figure 3.3: A schedule

In this framework, *processors* correspond to *nodes*, and *dedicated tasks* to *multi-destination* data packets. For an illustration see Figure 3.2 on the preceding page and Figure 3.3. The goal, minimizing the schedule makespan of dedicated tasks, corresponds to the overall *transmission* time, i.e. the time needed to send all data packets out.

We call fix_j the *type* and $|\mathrm{fix}_j|$ the *size* of task $j \in T$, and use $\Delta_{\max}$ to denote $\max_j |\mathrm{fix}_j|$ and $r_{\max}$ to denote $\max_j r_j$. To refer to the variants of the above scheduling problem, we use the standard notation scheme by Graham at al. [GLLK79]. Here, $P|\mathrm{fix}_j, r_j|C_{\max}$ denotes the above problem itself in the offline case. In offline scheduling, the scheduler has full information of the problem instance. In contrast, in online scheduling, information about the problem instance is made available during the course of scheduling. Thus, $P|\mathrm{online}, \mathrm{fix}_j, p = 1|C_{\max}$ denotes the online variant where all $p_j = 1$ and the tasks *arrive over list* and $P|\mathrm{online}, \mathrm{fix}_j, r_j|C_{\max}$ denotes the online variant where all p_j are arbitrary and the tasks *arrive over time*. In the first case, the tasks of T are ordered in some list (sequence) and presented one by one to the scheduler according to this list. The existence of a job is not known until all its predecessors in the list have already been scheduled. In the second variant, tasks j in T arrive at their release dates r_j. The tasks can be started at a time bigger than or equal to their release dates, and at any point of time, the scheduler only has knowledge of the released tasks.

Known Results and Our Contribution. Variants of the multiprocessor tasks scheduling problem have been studied, but the previous research has mainly focused on the offline case. Hoogeveen et al. [HdVV94] showed that already the three-processor problem $P3\,|\,\mathrm{fix}_j|C_{\max}$ is strongly NP-hard, and proved that even if all tasks have unit processing times, there exists no polynomial approximation algorithm for $P\,|\,\mathrm{fix}_j, p_j = 1|C_{\max}$ with performance ratio smaller than $\frac{4}{3}$, unless P=NP. However, in the same work it was shown that if the number of processors m is fixed, i.e. $Pm|\,\mathrm{fix}_j, p_j = 1|C_{\max}$, then the problem is solvable in polynomial time. Later, Amoura et al. [ABKM97] proposed a polynomial time approximation scheme (PTAS) for problem $Pm|\,\mathrm{fix}_j|C_{\max}$, and Bampis & Kononov [BK01] extended this result to a PTAS for $Pm|\,\mathrm{fix}_j, r_j|C_{\max}$. A $\frac{7}{6}$-approximation algorithm for $P3|\,\mathrm{fix}_j|C_{\max}$, which is due to Goemans [Goe95], remained the best low time complexity approximation algorithm for a long time, until very recently it has been improved to a $\frac{9}{8}$-approximation algorithm by Chen & Huang [CH01]. Fishkin et al. [FJP01b] extended the negative results presented in [HdVV94]. It has been proven that $P\,|\,\mathrm{fix}_j, p_j = 1|C_{\max}$ cannot be approximated within a factor of $m^{1/2-\varepsilon}$, neither for some $\varepsilon > 0$, unless P=NP; nor for any $\varepsilon > 0$, unless NP=ZPP.

In the online context, there are several results known for the *parallel* variant of the multiprocessor tasks scheduling problem, see e.g. [FST94, FKST93, BM98]. Up to our best knowledge, no online algorithms with guaranteed competitive ratio are known for the dedicated variant of the multiprocessor model, considered in this paper. Although some algorithms have been recently proposed in the literature, their performances are not evaluated analytically, but by using simulations [CDI01b, CDI01a].

Here we present several results, important from both theoretical and practical point of view. First, we deal with the problems where tasks have unit processing times. Using the so-called *first-fit* technique, we give an online algorithm for $P|\,\mathrm{online}, fix_j, p_j = 1|C_{\max}$, which is k-competitive if the maximum task size $\Delta_{\max}$ is bounded by some constant k. It is interesting to point out that our analysis is based on a transformation of our scheduling problem to the classical (vertex) coloring problem of a particular class of graphs, the intersection graphs of k-tuples. We propose simple extensions of this algorithm to a $2\sqrt{m}$-competitive algorithm for $P\,|\,\mathrm{online}, \mathrm{fix}_j, p_j = 1|C_{\max}$ and $(2\sqrt{m}+1)$-competitive algorithm for $P\,|\,\mathrm{online}, \mathrm{fix}_j, p_j = 1, r_j|C_{\max}$. Clearly, the $2\sqrt{m}$-competitive algorithm and the first-fit algorithm are complementary. For instance, if $k = 2$ then the first-fit algorithm is better. From another side, whenever $k = m$ the first algorithm outperforms the second one.

Next, we switch to the general problem with arbitrary processing times. We show that any online algorithm which schedules tasks arriving over-list and leaves no

unnecessary idles cannot be better than m-competitive. In some sense, this leaves no hope for contracting any good on-line algorithm based on the first-fit technique. However, using our *split-round* technique, we first give an off-line $2k$-approximation algorithm for $P|$ online, $fix_j|C_{\max}$ if the maximum task size $\Delta_{\max}$ is bounded by some constant k, and modify it to an off-line $3\sqrt{m}$-approximation algorithm for $P \,|\, \text{fix}_j|C_{\max}$. Then, by using the so-called *active-passive-bins* scheduling technique by Shmoys, Wein & Williamson [SWW95], we give an online $6\sqrt{m}$-competitive algorithm for $P \,|\, \text{online, fix}_j, r_j|C_{\max}$ in which the existence of a task is unknown until its release date.

Last Notes. We say that a polynomial algorithm A is a ρ-*approximation* algorithm if for all problem instances it outputs a schedule with makespan at most $\rho \cdot OPT$, where OPT is the makespan of the optimal schedule. The value of ρ is called the approximation ratio of A. If A is an online ρ-approximation algorithm, then we say that it is a ρ-*competitive* algorithm or A is ρ-*competitive*. In this case, ρ is called the competitive ratio of A.

The rest of this chapter is organized as follows. In Section 3.1.1 we give some preliminary results. In Section 3.2 we consider scheduling dedicated tasks with unit processing times, and then in Section 3.3 we consider scheduling dedicated tasks with arbitrary processing times. Finally, in Section 3.4 we give some concluding remarks.

3.1 PRELIMINARIES

Here we introduce some general definitions from the graph theory and prove some simple results that will be used throughout this chapter. We first discuss the problem of coloring of k-tuple graphs. Then, we discuss the problem of coloring of the task conflict graph. The properties proven pertain to the case when the tasks have unit processing times. However, the ideas will be used in a generalized form for arbitrary processing times as well.

3.1.1 Coloring of k-tuple Graphs

Here we first give the definition for a graph clique, a graph coloring, and a k-tuple graph. Then, we preset a simple online algorithm for coloring of k-tuple graphs.

Clique. Given an undirected graph $G = (V,E)$, a subset $K \subseteq V$ is a *clique* if every two vertices in K are joined by an edge in E. A *maximum clique* is, naturally,

a clique whose number of vertices is at least as large as that for any other clique in the graph, and its size $\omega(G)$ is called the *clique number* of G.

Coloring. A (vertex) k-coloring of a graph $G = (V,E)$ is a function $c : V(G) \to \{1,\ldots,k\}$ such that $c(u) \neq c(v)$ whenever u is adjacent to v. If a k-coloring of G exists, then G is called k-colorable. A *minimum coloring* of G is a coloring that uses as few different colors as possible, and its number of colors

$$\chi(G) \;=\; \min\{k \;:\; G \text{ is } k-\text{colorable}\}$$

is called the *chromatic number* of G.

Clique and coloring problems are very closely related. It is straightforward to see that the clique number of G is a lower bound on the chromatic number of G, i.e.

$$\omega(G) \;\leq\; \chi(G). \tag{3.1}$$

Graph of k-tuples. Let X be a finite set. A *k-tuple* of X is a set having k (or less) elements of X. A graph $G = (V,E)$ is a *k-tuple graph* if there exists a set X such that each node of V corresponds to a k-tuple of X and there is an edge in E between any two vertices iff the intersection of the corresponding k-tuples is not empty. Notice that several vertices can correspond to a single k-tuple.

First Fit Coloring. Different variants of the graph (vertex) coloring problem have been studied. It is widely known that coloring of arbitrary graphs is strongly NP-hard. Furthermore, there is no ρ-approximation algorithm with $\rho = n^{1/7-\varepsilon}$ unless P=NP, where n is the number of vertices of the graph [BGS98]. However, for restricted graph classes, there can exist either exact polynomial time algorithms or better approximation algorithms, as well as online coloring algorithms [FW98].

For a graph $G = (V,E)$, an online coloring algorithm colors the vertices in V one vertex at a time in the externally determined order $v_1 \prec v_2 \prec \cdots \prec v_n$. At each time t the algorithm must irrevocably assign a color to vertex v_t, while it can only see the subgraph of G induced by the vertices in $V_t := \{v_i | i = 1,\ldots,t\}$.

Now we are ready to present a very simple variant of an online coloring algorithm:

> FIRST FIT COLORING (FFC):
> Select vertices v from V in an arbitrary order while coloring with an initial sequence of colors $1,2,\ldots$. Assign the vertex v the least color that has not already been assigned to any vertex adjacent to v.

Lemma 3.1.1. *For a k-tuple graph G, the number of colors used by the algorithm* FFC *is at most* $k \cdot (\omega(G) - 1) + 1$*, where* $\omega(G)$ *is the clique number of G.*

Proof. Let X be a finite set and $G = (V, E)$ be a k-tuple graph with the corresponding k-tuples of X. Take a vertex v in V. Let $\{a_1, \ldots, a_k\}$ be the k-tuple of v. Let

$$N(v) = \{u \in V \mid (u, v) \in E\}$$

be the neighborhood of v.

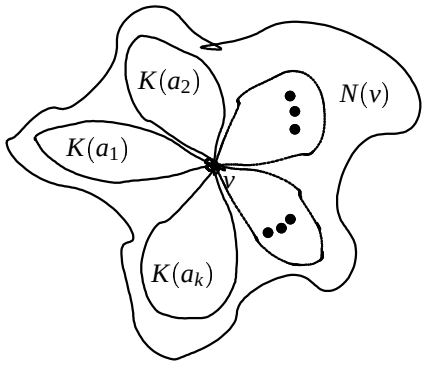


Figure 3.4: A vertex v and $N(v)$

We select elements from $\{a_1, \ldots, a_k\}$ one by one. For each selected a_i ($i = 1, \ldots, k$) we put all vertices from $N(v)$ whose corresponding k-tuples of X contain a_i into set $K(a_i)$, and then $N(v) := N(v) \setminus K(a_i)$. In the end, we have sets $K(a_1), K(a_2), \ldots, K(a_k)$. For an illustration see Figure 3.4.

By the definition of k-tuple graphs, each $K(a_i)$ $i = 1, \ldots, k$) is a clique in G. By the construction, these cliques $K(a_1), K(a_2), \ldots, K(a_k)$ are disjoint and cover $N(v)$. Furthermore, each $K(a_i) \cup \{v\}$ ($i = 1, \ldots, k$) is also a clique. Hence, $|K(a_i) \cup \{v\}| \leq \omega(G)$ ($i = 1, \ldots, k$). Combining, we have $|N(v)| = |\cup_{i=1}^{k} K(a_i)| \leq k(\omega(G) - 1)$. Thus, FFC uses at most $k(\omega(G) - 1) + 1$ colors on $N(v) \cup \{v\}$ for any vertex $v \in V$. □

3.1.2 Coloring of the Conflict Graph

We are given a task set $T = \{1, \ldots, n\}$ and a processor set $M = \{1, 2, \ldots, m\}$. Each task $j \in T$ has a processing time p_j, a release date r_j, and a type $\text{fix}_j \subseteq M$. The goal is to find a non-preemptive schedule which minimizes the makespan $C_{\max} = \max_j C_j$.

Two tasks i and j in T are called *incompatible* iff $\text{fix}_j \cap \text{fix}_j \neq \emptyset$. The *conflict* graph of T is a graph G_T such that $V(G_T) = T$ and there is an edge $\{j,i\} \in V(G_T)$ iff tasks j and i are incompatible.

Consider the case when for all tasks j in T it holds $r_j = 0$ and $p_j = 1$. Then, any clique in G_T corresponds to a set of pairwise incompatible tasks, and any coloring of G_T corresponds to assigning every incompatible pair to distinct colors. Furthermore, if there is a coloring of G_T with L colors, then there is a schedule for T of makespan L. For an illustration see Figure 3.5. Hence, by (3.1), the maximum clique number $\omega(G_T)$ is a lower bound on the schedule makespan for T.

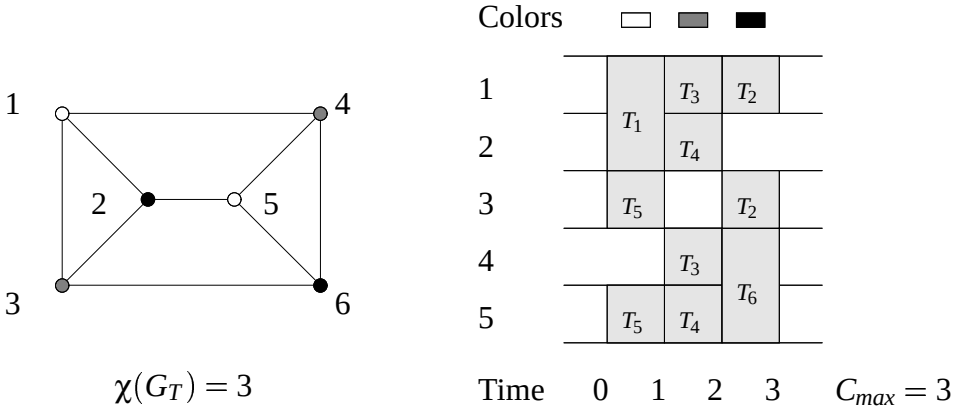


Figure 3.5: Scheduling of $T = \{1,2,3,4,5,6\}$ and coloring of G_T

Now we can give the following simple algorithm:

> FIRST FIT SCHEDULING (FFS):
> Schedule the tasks of T by starting the task j at the earliest interval $[t,t+1)$ in which the processors of fix_j are not busy.

Consider the case when the maximum task size $\Delta_{\max} = \max_{j \in T} |\text{fix}_j|$ is bounded by some constant k. Then, the conflict graph G_T a *k-tuple graph*, and by using the result of Lemma 3.1.1 we have:

Lemma 3.1.2. *If the maximum task size $\Delta_{\max}$ is bounded by some constant k, then the algorithm* FFS *is k-competitive for $P\,|\,\text{online}, \text{fix}_j, p_j = 1|C_{\max}$.*

Proof. First consider the case when the tasks of T arrive over-list. Let OPT be the optimum makespan for $P\,|\,\text{fix}_j, p_j = 1|C_{\max}$. Let G_T be the corresponding conflict

graph of task set T. One can see that FFC working on G_T cannot outperform FFS working on T, in the worst case analysis. Each interval $[t,t+1)$ corresponds to color t, the schedule length corresponds to the number of colors, and vice versa. Thus, since $\omega(G_T)$ is a lower bound on OPT, by Lemma 3.1.1 the makespan of the output schedule by FFS is at most $k \cdot OPT$. □

The above lemma does not guarantee a good performance of FFS on instances with $\Delta_{\max} = m$. Furthermore, there is no such a simple algorithm for the case when all $r_j = 0$, but all p_j are arbitrary. Indeed, we can add a weight p_j to each vertex j of the conflict graph G_T and deal with the maximum weighted clique problem. In this framework, the sum of vertex weights in a maximum weighted clique of G_T is a lower bound on the schedule makespan of T. However, we will not consider this approach.

3.2 SCHEDULING OF TASKS WITH UNIT PROCESSING TIMES

Now we consider the online version of the dedicated scheduling problem. In this section we study the simplest case when tasks have unit processing times, i.e. all $p_j = 1$.

As the first step, we modify the algorithm FFS as follows:

> FIRST FIT SCHEDULING+ (FFS+):
> Schedule the tasks of T by starting the task j at the earliest interval $[t,t+1)$, $t \geq r_j$ in which the processors of fix_j are not busy and there is no task in $[t,t+1)$
>
> - of size greater than $\sqrt{m}$ if $|\text{fix}_j| \leq \sqrt{m}$, and
> - of size at most $\sqrt{m}$ if $|\text{fix}_j| > \sqrt{m}$.

Less formally, the output schedule can be split into two parts. The first part includes the intervals during which each processed task has size at most $\sqrt{m}$, we call these intervals "red" colored. The second part includes the intervals during which each processed task has size greater than $\sqrt{m}$, we call these intervals "blue" colored. For an illustration see Figure 3.6 on the next page. Accordingly, one can think in terms of assigning the tasks of T, depending on their sizes, to either blue or red colored intervals.

We can prove the following:

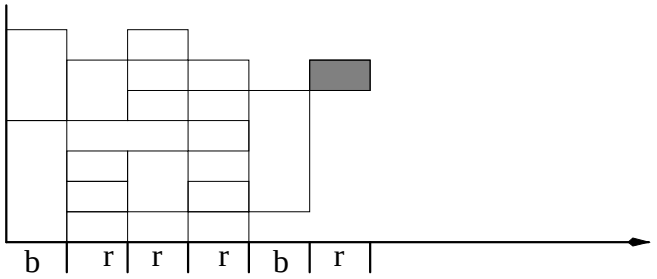


Figure 3.6: Red and blue intervals in a schedule

Lemma 3.2.1. *The algorithm* FFS+ *is* $2\sqrt{m}$*-competitive for* $P\,|\,\text{online, fix}_j,\ p_j = 1|C_{\max}$ *and is* $(2\sqrt{m}+1)$*-competitive for* $P\,|\,\text{online, fix}_j, p_j = 1, r_j|C_{\max}$.

Proof. Let OPT denote the optimal makespan for $P\,|\,\text{fix}_j, p_j = 1|C_{\max}$. Let $T^+ := \{j \in T : |\,\text{fix}_j| > \sqrt{m}\}$ and $T^- := \{j \in T : |\,\text{fix}_j| \leq \sqrt{m}\}$.

Assume that the tasks of T arrive over-list. Consider the "red" part of the output schedule, which corresponds to T^-. The maximum task size $\Delta_{\max}(T^-) \leq \sqrt{m}$. Thus, the number of "red" colored intervals is at most $\sqrt{m}OPT$ (Lemma 3.1.2). Consider the "blue" part of the output schedule, which corresponds to T^+. There is at least one task appears in each "blue" colored interval. For each task $j \in T^+$ we have $|\,\text{fix}_j| > \sqrt{m}$. There are m processors. Hence, we can place at most $m/\sqrt{m} = \sqrt{m}$ tasks in each "blue" colored interval. Thus, the number of "blue" colored interval is at most $\sqrt{m}OPT$. Combining the two bounds we get that the length of the output schedule is at most $2\sqrt{m}\,OPT$.

Assume that the tasks of T arrive over-time. Let OPT' denote the optimal makespan for $P\,|\,\text{fix}_j, p_j = 1, r_j|C_{\max}$. Let $r_{\max} := \max r_j$ be the last release date. Clearly, $OPT \leq OPT'$ and $r_{\max} \leq OPT'$. Assume that each task j in T arrives at $r_{\max}$. Then, as we saw above, the length of the schedule output by FFS+ is at most $2\sqrt{m}\,OPT$. Hence, if each task j in T arrives at r_j, the length of the schedule output by FFS+ is at most $r_{\max} + 2\sqrt{m}\,OPT$. That is,

$$r_{\max} + 2\sqrt{m}\,OPT \leq OPT' + 2\sqrt{m}\,OPT' = (2\sqrt{m}+1)OPT'.$$

□

3.3 SCHEDULING OF TASKS WITH ARBITRARY PROCESSING TIMES

Here we consider the general case when all processing times p_j are arbitrary. In the first part we analyze the first-fit technique. In the second part, we consider the

offline version of the problem in the case when all release dates $r_j = 0$. We present our split-round technique, and derive an approximation algorithm. In the final part, we consider the online version where tasks arrive over-time. We present an online algorithm which is based on the well-known passive-active-bin technique.

3.3.1 First-Fit Technique

Consider m tasks $T_1, T_2 \ldots, T_m$ with processing times $p_k = 1 + \frac{k}{m}\varepsilon$ and types $\text{fix}_k = \{k\}$ $(k = 1, \ldots, m)$, and, in addition, $m-1$ tasks T_{m+1}, T_{m+2}, ..., T_{2m-1} with the same type $\text{fix}_k = \{1, \ldots, m\}$ and processing time $p_k = \frac{\varepsilon}{m-1}$ $(k = m+1, \ldots, 2m-1)$.

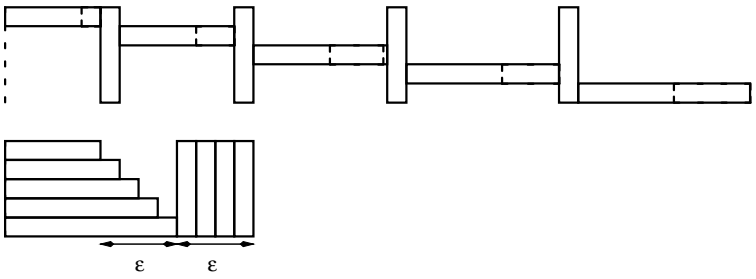


Figure 3.7: The output and optimal schedules

If all these $2m-1$ tasks arrive in the order $1, (m+1), 2, (m+2), \ldots, i, (m+i), \ldots, (m-1), (2m-1), m$, then any deterministic on-line algorithm which leaves no unnecessary idles produces a schedule of makespan greater than $m+2\varepsilon$, while an optimal schedule has a makespan equal to $1+2\varepsilon$. For an illustration see Figure 3.7. Thus, m is a lower bound on the competitive ratio of any on-line algorithm which uses the first-fit technique.

We define $P(T) = \sum_{j \in T} p_j$ to be the total processing time of tasks of T, and $L(T) = \frac{1}{m} \sum_{j \in T} p_j |\text{fix}_j|$ to be the average load of T. Then, it holds that

$$P(T) \leq m \cdot L(T) \qquad \text{and} \qquad L(T) \leq OPT \leq \max_{j \in T} r_j + P(T),$$

where OPT is the minimum makespan for T.

Consider the following algorithm:

GENERAL FIRST FIT SCHEDULING (GFFS):
Schedule the tasks of T by starting the task j at the earliest time $t \geq r_j$ such that the processors of fix_j are not busy in interval $[t, t+p_j)$.

Sure, the above algorithm uses the first-fit technique and cannot be better than m-competitive. However, it cannot be much worse than that as well.

Lemma 3.3.1. *The algorithm* GFFS *is m-competitive for $P\,|\,\text{online}, \text{fix}_j|C_{\max}$ and $(m+1)$-competitive for $P\,|\,\text{online}, \text{fix}_j, r_j|C_{\max}$.*

Proof. Consider the case where the tasks of T arrive over-list. Let OPT be the minimal makespan for T. Let C denote the makespan of the schedule found by GFFS working on T. Since for each task $j \in T$ the release date $r_j = 0$ and the algorithm GFFS uses the first-fit technique, $P(T)$ is an upper bound on C. Hence, it holds that

$$C \leq P(T) \leq m \cdot L(T) \leq m \cdot OPT.$$

Consider the case where the tasks of T arrive over-time. Let OPT' denote the optimal makespan for $P\,|\,\text{fix}_j, r_j|C_{\max}$. Let C' denote the makespan of the schedule found by GFFS working on T. Let $r_{\max} := \max r_j$ be the last release date. Clearly, $OPT \leq OPT'$ and $r_{\max} \leq OPT'$. Assume that each task j in T arrives at $r_{\max}$. Then, as we saw above, the length of the schedule (after $r_{\max}$) output by GFFS is at most $mOPT$. Hence, if each task j in T arrives at r_j, the length of the schedule output by GFFS is at most $r_{\max} + mOPT$. That is,

$$C' \leq r_{\max} + mOPT \leq OPT' + mOPT' = (m+1)OPT'.$$

□

3.3.2 Split-Round Technique

Here we consider the offline version $P\,|\,\text{fix}_j|C_{\max}$. We start with the following simple algorithm:

ROUND SCHEDULING (RS):
1. For each task $j \in T$ round p_j to the smallest power of two, say $[p_j]^a$.
2. Apply GFFS to tasks $j \in T$ ordered by non-increasing $[p_j]$'s.

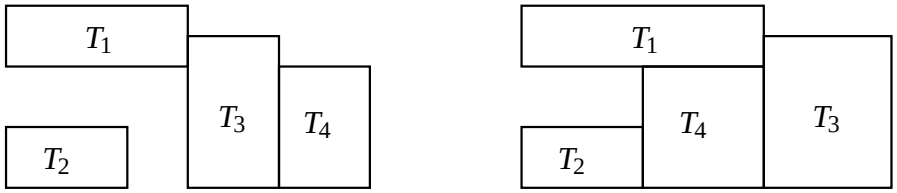
[a] Here $[p_j] = 2^a \geq p_j$ and $2^{a-1} < p_j$.

By using the result of Lemma 3.1.1 we can prove the following result:

Lemma 3.3.2. *If the maximum task size $\Delta_{\max}$ is bounded by some constant k, then the algorithm* RS *is $2k$-approximation algorithm for $P \mid \mathrm{fix}_j \mid C_{\max}$.*

Proof. Let OPT be the optimal makespan for the tasks in T. We round each processing time p_j, $j \in T$, to the smallest power of 2. This increases the objective function value by at most a factor of 2, i.e. the new minimal makespan $OPT' \leq 2\,OPT$. Now all processing times $[p_j]$, $j \in T$, are powers of 2. We schedule the tasks of T in non-increasing order of processing times.

Consider an illustration in Figure 3.8. We schedule tasks $1,2,3$ and 4. First, look at a schedule for the non-rounded tasks in a). The processors required by task 2 are free right after task 2, but task 4 cannot start. This creates a "gap". Now, look at a schedule for the rounded tasks in b). There is no such "gap". Task 4 starts right after task 2. Here, the processing time of task 1 is exactly the sum of processing times of task 2 and 4. If processing times are not powers of two, we cannot guarantee this property.



a) non-rounded tasks b) rounded tasks

Figure 3.8: Scheduling 4 tasks

Consider the schedule found by GFFS while working on the rounded tasks $j \in T$ ordered by non-increasing $[p_j]$'s. Let C be the makespan of the output schedule by SR.. Let z be a task such that $C = C_z$. Accordingly,

$$S_z \;=\; C_z \;-\; p_z \;=\; C \;-\; p_z$$

is the starting time of task z.

By the above observation, there are no "gaps" on the processors of fix_z up to time S_z. Hence, there is a sequence $s_1, s_2, \ldots, s_q$ of tasks in T such that at each moment $t \in [0, S_z)$ for exactly one task $s(t) \in \{s_1, s_2, \ldots, s_q\}$ it holds that $\mathrm{fix}_{s(t)} \cap \mathrm{fix}_z \neq \emptyset$. (See Figure 3.9 on the following page). Hence,

$$S_z \;=\; \sum_{i=1}^{q} p_{s_i} \qquad \text{and} \qquad C \;=\; p_z \;+\; \sum_{i=1}^{q} p_{s_i}.$$

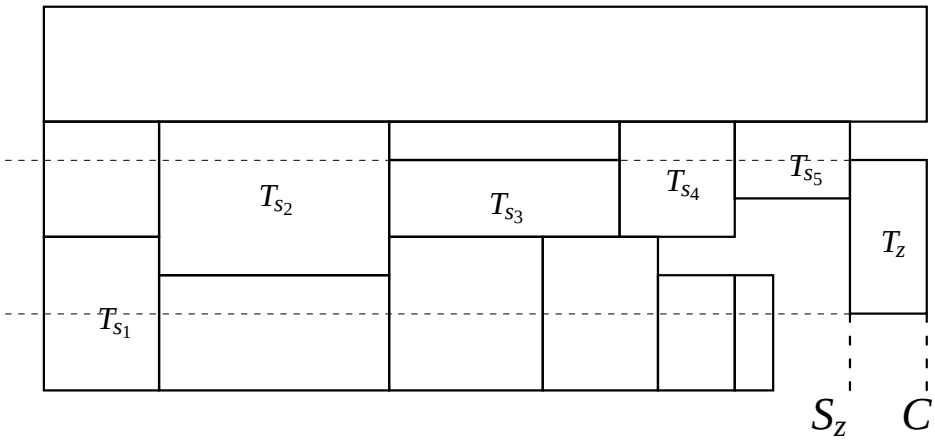


Figure 3.9: A task z and a sequence s_1, s_2, s_3, s_4, s_5

As we saw, each task s_i, $i = 1, \ldots, q$, requires at least one of the processors in fix_z. Assume that at each moment of time all processors in fix_z are busy processing only tasks $s_1, s_2, \ldots, s_q, z$. By $|\text{fix}_z| \leq \Delta_{\max} \leq k$, the processors in fix_z can be free only after

$$L' = \frac{1}{k}\left(p_z + \sum_{i=1}^{q} p_{s_i}\right).$$

From another side,

$$\frac{1}{k}\left(p_z + \sum_{i=1}^{q} p_{s_i}\right) = \frac{C}{k}.$$

Furthermore, all $s_1, s_2, \ldots, s_q$ and z are in T. Hence, $L' \leq OPT' \leq 2\,OPT$. Combining, we have

$$C \leq kL' \leq 2k\,OPT.$$

□

As before, the above lemma does not guarantee a good performance of RS on instances with $\Delta_{\max} = m$. However, we can modify the algorithm as follows:

SPLIT ROUND SCHEDULING (SRS):
1. Form $T^+ := \{j \in T : |\operatorname{fix}_j| > \sqrt{m}\}$ and $T^- := \{j \in T : |\operatorname{fix}_j| \leq \sqrt{m}\}$.
2. Apply GFFS to tasks $j \in T^+$ ordered in an arbitrary way.
3. For each task $j \in T^-$ round p_j to the smallest power of two, say $[p_j]$[a].
4. Apply GFFS to tasks $j \in T^-$ ordered by non-increasing $[p_j]$'s.

[a] Here $[p_j] = 2^a \geq p_j$ and $2^{a-1} < p_j$.

We can state the following

Theorem 3.3.3. *The algorithm* SRS *is a* $3\sqrt{m}$*-approximation algorithm for* $P|\operatorname{fix}_j|C_{\max}$.

Proof. Let *OPT* be the minimum makespan for $P|\operatorname{fix}_j|C_{\max}$. Let C^+ be the makespan of the schedule found by GFFS while working on the tasks of T^+ ordered arbitrary (Step 2), and C^- be the makespan of the schedule found by GFFS while working on the rounded tasks $j \in T^-$ ordered by non-increasing $[p_j]$'s (Steps 3 and 4).

Since for each task $j \in T^+$ the size $|\operatorname{fix}_j| > \sqrt{m}$, it holds that

$$L(T^+) = \frac{1}{m}\sum_{j\in T^+} p_j|\operatorname{fix}_j| \geq \frac{\sqrt{m}}{m}\sum_{j\in T^+} p_j = \frac{P(T^+)}{\sqrt{m}}.$$

Thus,

$$C^+ \leq P(T^+) \leq \sqrt{m}L(T^+) \leq \sqrt{m}OPT.$$

For the tasks of $T^- \subseteq T$ by Lemma 3.3.2 we have $C^- \leq 2\sqrt{m}OPT$. Hence, we get

$$C^+ + C^- \leq \sqrt{m}OPT + 2\sqrt{m}OPT \leq 3\sqrt{m}OPT,$$

i.e. the makespan of the output schedule by SRS is at most $3\sqrt{m}OPT$. □

3.3.3 Passive-Active-Bin Scheduling

We are ready to present the following main algorithm for $P|\text{online}, \operatorname{fix}_j, r_j|C_{\max}$:

SPLIT-ROUND SCHEDULING+ (SRS+):
1. $B := \emptyset$, $B' := \emptyset$, $t := 0$.
(B – active bin, B' – passive bin, t – time.)
2. B collects the tasks released at time t.
3. If $B \neq \emptyset$, then (a) the processors start at t and work until the time t' when the schedule for B defined by SRS completes, (b) B' collects the tasks released in $(t, t']$.
4. If $B' \neq \emptyset$, then $B := B'$, $B' := \emptyset$, $t := t'$. Go to Step 3.
5. If $B' = \emptyset$, then (a) all processors remain idle until the time t' when the next job is released, (b) $t := t'$ and $B := \emptyset$. Go to Step 2.

Less formally, we proceed as follows. At each moment of time we deal with two bins, one of which is set to "active" and the other one is set to "passive". For an illustration see Figure 3.10. At the first release date, we collect all released tasks into the active bin. Then, with the time flow, we schedule the tasks of the active bin by SRS and collect the tasks being released into the passive bin. At each moment of time when SRS has no task to schedule, we exchange the activities of bins (if there are any tasks to schedule at all).

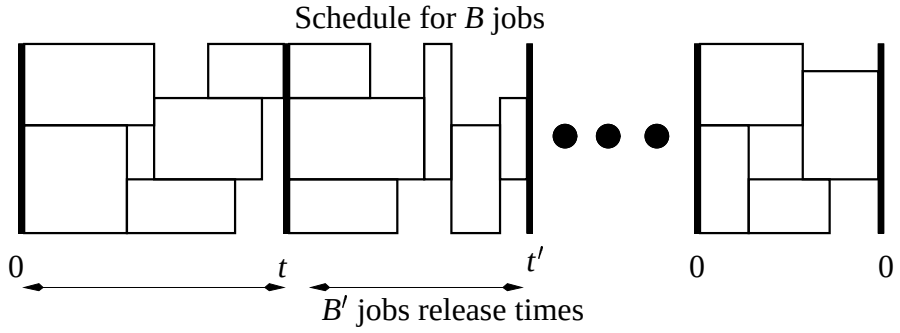


Figure 3.10: Passive-Active-Bin scheduling

Here we call the following result by Shmoys, Wein & Williamson in [SWW95]: "Let A be a polynomial-time scheduling algorithm that works in an environment in which each job to scheduled is available at time 0 and which always produces a schedule of length at most ρC^*. For the analogous environment in which the existence of a job is unknown until its release date, there exists another polynomial-time algorithm A' that works in this more general setting and produces a schedule of length at most $2\rho C^*$". In fact, SRS is similar to the algorithm constructed, and we can give the following result

Theorem 3.3.4. SRS+ *is* $6\sqrt{m}$*-competitive for* $P\,|\,\text{online}, \text{fix}_j, r_j|C_{\max}$ *in which the existence of a task is unknown until its release date.*

3.4 CONCLUDING REMARKS

In this chapter we considered the problem of scheduling independent dedicated tasks to minimize the makespan. The problem is well-known and there are a number of approximability results. However, most of them concern the case when the number of processors is equal to 2, 3 or when it is a fixed constant. Furthermore, no results were known for the online version of the problem, which applications can be found in WDM LANs. Here we presented the first results in this direction. We derived some simple online algorithms for both over-list and over-time scheduling concepts. The techniques used are very simple and all algorithms have quite low running time, that is important for applications.

One of interesting questions is an extension of the results for the cases when tasks can have precedence constrains. We believe that our algorithms and techniques can be generalized here. We are convinced that bound $O(\sqrt{m})$ will remain for the general case. However, the study of simple cases with the number of processors 2 and 3 is also very interesting.

CHAPTER 4

ON MAXIMIZING THE THROUGHPUT OF MULTIPROCESSOR TASKS

4.1 INTRODUCTION

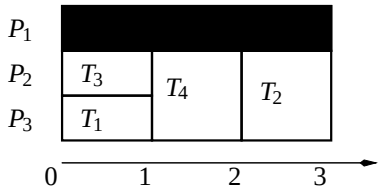
In the traditional theory of scheduling, each task is processed by only one processor at a time. However, due to the rapid development of parallel computer systems, new theoretical approaches have emerged to model scheduling on parallel architectures. One of these is scheduling multiprocessor tasks, see e.g. [Dro96, GLLK79].

In this paper we address the following multiprocessor scheduling problem. A set $T = \{T_1, T_2, \ldots, T_n\}$ of n tasks has to be executed by a set of m processors $P = \{P_1, P_2, \ldots, P_m\}$. Each task T_j ($j = 1, 2 \ldots, n$) has a unit processing time $p_j = 1$ and an integral due date d_j. Each processor can work on at most one task at a time, and each task can (or may need to) be processed simultaneously by several processors. Here we assume that all tasks are available at time zero and the objective is to maximize the *throughput* $\sum \bar{U}_j$, where $\bar{U}_j = 1$ if task T_j is completed before or at time d_j (T_j is said to be *on time* or *early*), and $\bar{U}_j = 0$ otherwise (T_j is said to be *late* or *tardy*).

We deal with two variants of this problem. In the *parallel* variant, the multiprocessor architecture is disregarded and for each task T_j ($j = 1, 2 \ldots, n$) there is given a prespecified number $\text{size}_j \in \{1, 2, \ldots, m\}$ which indicates that T_j requires the simultaneous use of size_j processors in P. In the *dedicated* variant, each task T_j ($j = 1, 2 \ldots, n$) there is given a prespecified set $\text{fix}_j \subseteq \{1, 2, \ldots, m\}$ which indicates that T_j requires the simultaneous use of the processors of P with indices in fix_j. For an illustration see Figure 4.1 and Figure 4.2 on the following page.

To refer to the variants of the problem, we use the standard notation scheme introduced in [GLLK79, LLKS93, Dro96]. We will write $P|\,\text{size}_j, p_j = 1|\sum \bar{U}_j$ and $P|\,\text{fix}_j, p_j = 1|\sum \bar{U}_j$ to denote the parallel and dedicated variants of the problem. If all tasks have a common due date D, i.e. all $d_j = D$, we will write $P|\,\text{size}_j, p_j = 1, d_j = D|\sum \bar{U}_j$ and $P|\,\text{fix}_j, p_j = 1, d_j = D|\sum \bar{U}_j$, respectively.

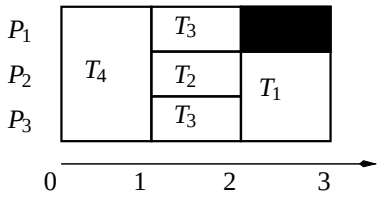
T_j	T_1	T_2	T_3	T_4
size_j	1	2	1	2
d_j	1	2	2	3



$$\sum \bar{U}_j = 1 + 0 + 1 + 1 = 3$$

Figure 4.1: Scheduling parallel tasks T_1, T_2, T_3 and T_4

T_j	T_1	T_2	T_3	T_4
fix_j	$\{2,3\}$	$\{2\}$	$\{1,3\}$	$\{1,2,3\}$
d_j	1	2	2	3



$$\sum \bar{U}_j = 0 + 1 + 1 + 1 = 3$$

Figure 4.2: Scheduling dedicated tasks T_1, T_2, T_3 and T_4

Known Results and Our Contribution. There are a lot of results known for the classical (non-multiprocessor) job scheduling problems, where the objective is either to minimize the (weighted) number of late (tardy) jobs or to maximize the (weighted) number of on time (early) jobs, see e.g. [Bru98, DP95, HPW00, KIM78, Law76, Law82, LM69, LKB77, Mon82, Moo68, RW98]. In the multiprocessor setting, the previous research has mainly focused on the objectives of minimizing *the makespan* and *the sum of completion times*. As a rule, scheduling multiprocessor tasks with unit processing times is a strongly *NP*-hard problem [Llo81, HdVV94]. However, a number of different approximation algorithms have been recently proposed in [ABKM97, BM98, CH01, CLL98, FST94, FKST93, Goe95, Llo81, TLWY94]. Up to our knowledge, no results are known for the multiprocessor tasks scheduling problem which concern either minimizing the number of late (tardy) tasks or maximizing the number of on time (early) tasks.

Here, focusing on the throughput objective, we present the first results in this direction. We derive the complexity results and present several approximation algorithms, for both parallel and dedicated variants of the problem.

In the first part of the chapter we consider the parallel variant of the problem. Each parallel task requires a prespecified number of processors, and there are at most m processors available at any the same time slot. By adopting the complexity result for 3-PARTITION [GJ79], we prove that problem $P|\,\mathrm{size}_j, p_j = 1|\sum \bar{U}_j$ is strongly NP-hard. Next, we propose two simple greedy algorithms, namely FFIS and LFIS. We prove that the worst-case ratio of FFIS is 2, and the worst-case ratio of LFIS is $2-1/m$, respectively. Finally, by refining both algorithms, we introduce an improved algorithm HA with the worst-case ratio at most $3/2-1/(2m-2)$.

In the second part of the chapter we consider the dedicated variant. Each dedicated task requires a prespecified subset of processors, and any two tasks that share a processor cannot be executed at the same time slot. By adopting the complexity result for MAXIMUM CLIQUE [Has99], we prove that problem $P|\,\mathrm{fix}_j, p_j = 1, d_j = 1|\sum \bar{U}_j$ with the common due date $D = 1$ is strongly NP-hard, and for any given $\varepsilon > 0$ it cannot be approximated within a factor of $m^{1/2-\varepsilon}$ unless NP = ZPP, where m is the number of processors. Next, for the common due date problem $P|\,\mathrm{fix}_j, p_j = 1, d_j = D|\sum \bar{U}_j$ we show that both algorithms FFIS and LFIS have the worst-case ratio at least $\sqrt{m}$ but at most $\sqrt{m}+1$. At the same time, both algorithms are optimal in the case when the number of processors $m = 2$, and their worst case ratio is $3/4$ in the case when the number of processors $m = 3$. Finally, we show bounds $\sqrt{m}$ and $\sqrt{m}+1$ on the worst case ratio of FFIS and LFIS remain valid for the general problem problem $P|\,\mathrm{fix}_j, p_j = 1|\sum \bar{U}_j$.

Interestingly, there are a number of different relations to some well-known com-

binatorial problems. Just beyond the relation to 3-PARTITION and MAXIMUM CLIQUE, we can also find that BIN PACKING and MULTIPLE KNAPSACK correspond to the parallel variant of our problem. We will discuss this in successive sections.

Last Notes. The quality of an approximation algorithm ALG is measured by its *worst-case ratio* defined as

$$R_{\mathrm{ALG}} = \sup_{T}\{N_{\mathrm{OPT}}(T)/N_{\mathrm{ALG}}(T)\},$$

where $N_{\mathrm{OPT}}(T)$ denotes the number of early tasks produced by an optimal algorithm OPT for a task set T, and $N_{\mathrm{ALG}}(T)$ denotes the number of early tasks in the schedule produced by ALG for T. For simplicity, throughout of this chapter we will write N_{OPT} and N_{ALG} if no confusion is caused.

Notice that if we know the number of tardy tasks in an optimal schedule, we can simply find the number of early tasks, and vice versa. In fact, it is convectional to consider minimizing the number of tardy tasks in investigating optimal algorithms. From another side, it can happen that there are no tardy tasks in an optimal schedule, but almost all tasks are early in a near-optimal schedule obtained by an approximation algorithm. In this case, even if the algorithm performs well from a practical point of view, its formal performance is evaluated as quite bad. The same situation can be also observed in online scheduling [HPW00]. Thus, for investigating approximation algorithms it is reasonable to choose the objective of maximizing the number of early tasks, i.e. the throughput objective.

The rest of this chapter is organized as follows. In Section 4.2 we introduce some notations and provide some preliminary results which will be used throughout the chapter. In Section 4.3 we present the results for the parallel model. In Section 4.4 for the dedicated model. Finally, in Section 4.5, we give some concluding remarks.

4.2 PRELIMINARIES

An instance of our scheduling problem is given as follows. There are a set $T = \{T_1, T_2, \ldots, T_n\}$ of n tasks and a set of m processors $P = \{P_1, P_2, \ldots, P_m\}$. For each task T_j ($j = 1, 2 \ldots, n$), there are a unit processing time $p_j = 1$ and an integral due date $d(T_j)$. In the parallel model, for each task T_j ($j = 1, 2 \ldots, n$) there is a prespecified number $\mathrm{size}_j \in \{1, 2, \ldots, m\}$. In the dedicated model, for each task T_j ($j = 1, 2 \ldots, n$) there is a prespecified subset $\mathrm{fix}_j \subseteq \{1, 2, \ldots, m\}$. If task T_j *meets* its due date $d(T_j)$ (T_j starts before or at $d_j - 1$ and $U_j = 1$) it is said to be early,

otherwise (T_j starts at or after d_j and $\bar{U}_j = 0$) it is said to be late. Our objective is to maximize the *throughput* $\sum \bar{U}_j$.

4.2.1 Task Size and Common Due Date

Throughout of the chapter we will use the following notations. For simplicity, we will write s_j instead of $size_j$ and τ_j instead of fix_j. For a task T_j ($j = 1,2,\ldots,n$) the value of s_j (for parallel) and $|\tau_j|$ (for dedicated) is called the *size* of T_j. Then, T_j is called *large* if its size is greater than $m/2$, and *small* otherwise.

In addition, we will write $0 < d_1 < \cdots < d_g = D$ to denote all distinct due dates, where D is the largest due date $\max_j d(T_j)$. Thus, each task T_j ($j = 1,2,\ldots,n$) has its integral due date $d(T_j) \in \{d_1, d_2, \ldots, d_g\}$. We say that tasks have a *common* due date D if $d(T_j) = D$ for all tasks T_j, $j = 1,2,\ldots,n$.

4.2.2 Scheduling on Time Slots

Informally, in order to construct a schedule for T we proceed as follows. We first partition interval $[0,D)$ into *time slots* $I_t = [t,t-1)$, $t = 1,\ldots,D$. Then, we define an order on the tasks in T and process them one by one in this order. For task T_j, we try to add it to a partial schedule in one of time slots I_t, $t \le d(T_j)$. If this can be done, we declare T_j be early ($\bar{U}_j = 1$), otherwise we declare T_j be late ($\bar{U}_j = 0$). We finish when all tasks are processed and output the final schedule.

Let ALG be a scheduling algorithm. For simplicity, we assume that ALG either *accepts* or *rejects* the tasks in T. Every accepted task in T is scheduled by ALG before its due date, and it is early in the output schedule. From another side, every rejected task in T is not scheduled by ALG, and it is *lost* (late) the output schedule. Thus, the number of tasks in the output schedule is equal to the number of early tasks $N_{\mathrm{ALG}}(T)$.

As we discussed, an algorithm ALG schedules the accepted tasks of T in time slots $I_t = [t,t-1)$, $t = 1,\ldots,D$. We will write N_t and $m(I_t)$ to denote the total number and the total size of the accepted tasks by algorithm ALG in time slot I_t. Clearly, for each time slot I_t, $t = 1,\ldots,D$, it holds that

$$m(I_t) \;\le\; m,$$

where m is the number of processors in P. Thus, the total number of accepted (early) tasks

$$N_{\mathrm{ALG}}(T) \;=\; \sum_{t=1}^{g} N_t,$$

and the total size of the accepted (early) tasks

$$\sum_{t=1}^{D} m_t \;\leq\; m \cdot D.$$

Finally, we say that a time slot I_t is *closed* if algorithm ALG meets the first task for which there is no room in I_t, and we say that I_t is *open* if it is not closed yet. For an illustration see Figure 4.3.

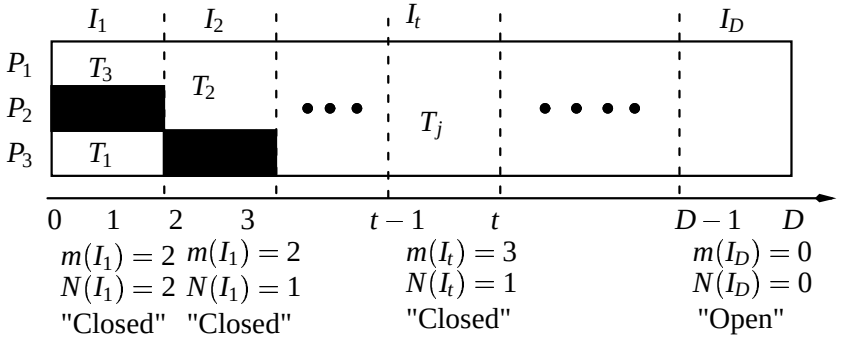


Figure 4.3: Scheduling on time slots

4.2.3 First-Fit and Last-Fit

We will consider two basic scheduling techniques: *First-Fit* and *Last-Fit*. Informally, First-Fit uses the concept of scheduling tasks *as early as possible*, whereas Last-Fit uses the concept of scheduling tasks *as late as possible*.

The difference can be seen from the following example. Let ALG be a scheduling algorithm and T be a task set. We first define an order on the tasks of T, and then let ALG process the tasks in this order. Let T_j be a task being processed by ALG. For an illustration see Figure 4.4 on the facing page.

Task T_j must be scheduled in one of time slots $I_1, I_2, \ldots, I_{d(T(j))}$. From one side, if non of time slots can "accommodate" T_j, then algorithm ALG rejects T_j. From another side, it can happen that in several time slots there are "free" processors which can be assigned to task T_j. Which time slot should be selected for T_j?

Indeed, there are two natural strategies for algorithm ALG: either select the first (earliest) time slot or the last (latest) time slot. If ALG always selects the the first time slot, we say that ALG is a FIRST-FIT algorithm. If ALG always selects the the last time slot, we say that ALG is a LAST-FIT algorithm.

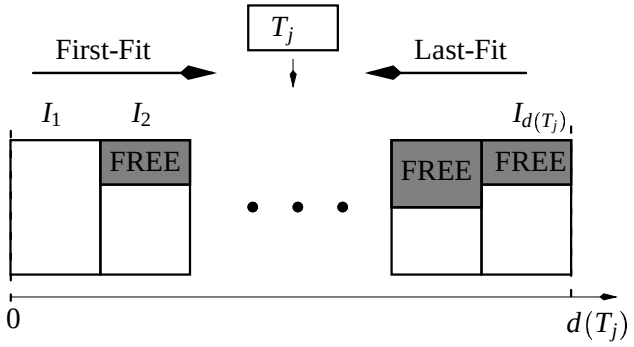


Figure 4.4: First-Fit and Last-Fit for task T_j

T_j	T_1	T_2	T_3	T_4
size$_j$	1	1	2	3
d_j	3	3	2	1

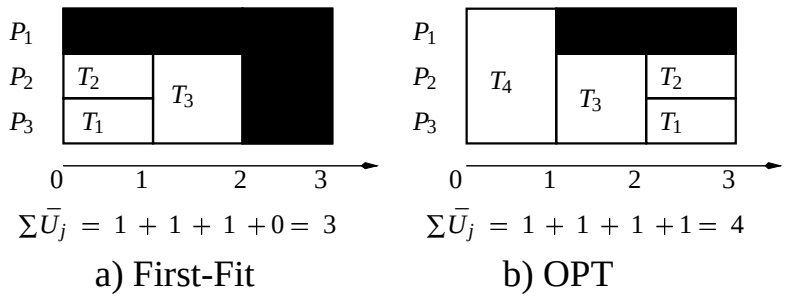


Figure 4.5: First-Fit scheduling of parallel tasks T_1, T_2, T_3 and T_4

T_j	T_1	T_2	T_3	T_4	T_5
size$_j$	1	1	1	3	3
d_j	1	2	3	2	3

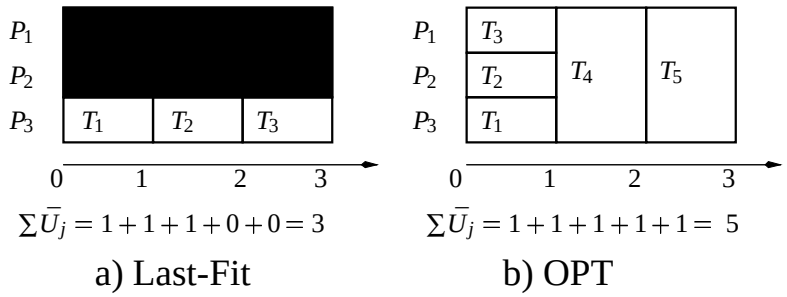


Figure 4.6: Latest-Fit scheduling of parallel tasks T_1, T_2, T_3, T_4 and T_5

Unfortunately, we can give simple examples showing that neither FIRST-FIT nor LAST-FIT is optimal. See Figure 4.5 on the page before. There are only tree tasks accepted in First-Fit scheduling, whereas all four tasks can be scheduled on time. See Figure 4.6. There are only four tasks accepted in First-Fit scheduling, whereas all five tasks can be scheduled on time. We will also refer to these examples later.

4.2.4 Scheduling in EDD and LDD

Here we discuss two main techniques for processing multiprocessor tasks. Consider one machine scheduling problem $1|p_j = 1|\sum \bar{U}_j$. There are n jobs. Each job j ($j = 1,\ldots,n$) has a unit processing time and a due date d_j. The goal is to maximize the throughput. In [Mon82], it was observed that, by creating sets

$$S_n = \{j \mid d_j \geq n\} \text{ and } S_k = \{j \mid k \leq d_j < k+1\}, \text{for } k = 1,\ldots,n-1,$$

and processing jobs in the order $S_1, S_2, \ldots, S_n$, rejecting a job when it is late, an optimal solution is obtained in $O(n)$ time. Similarly, an optimal solution cab be obtained by processing jobs in the reverse order $S_n, S_{n-1}, \ldots, S_1$ and rejecting a job when it is late.

Indeed, in both algorithms processing of tasks takes place either in EARLIEST DUE DATE (EDD) order – *in non-decreasing order of due dates* or in LATEST DUE DATE (LDD) order – *in non-increasing order of due dates*. Furthermore, by using First-Fit or Last-Fit we can generalize the algorithms for multiprocessor task scheduling. We can prove the following:

Lemma 4.2.1. *If all multiprocessor tasks are large or all multiprocessor tasks have size one, then using* FIRST-FIT *in* EDD *order or* LAST-FIT *in LDD order, an optimal solution is obtained in $O(nm)$ time.*

Proof. If all multiprocessor tasks are large, then no two tasks, either parallel or dedicated, can be processed in one time slot. Hence, the problem can be reformulated as $1|p_j = 1|\sum \bar{U}_j$. For an illustration see Figure 4.7.

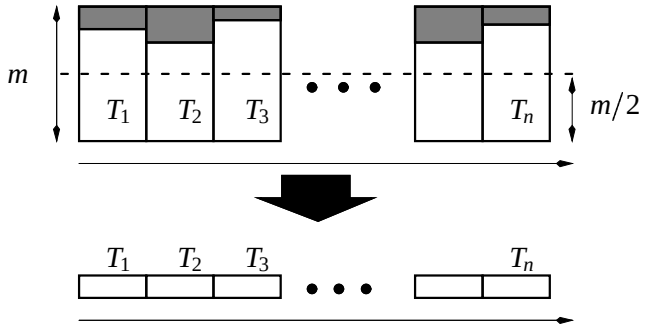


Figure 4.7: Scheduling large tasks

Now assume that all multiprocessor tasks have size one and there are m processors. If tasks are dedicated, then any two tasks with the same required processor cannot be scheduled in one time slot. Hence, the problem of scheduling tasks on m processors can be reformulated as m independent problems $1|p_j = 1|\sum \bar{U}_j$. For an illustration see Figure 4.8.

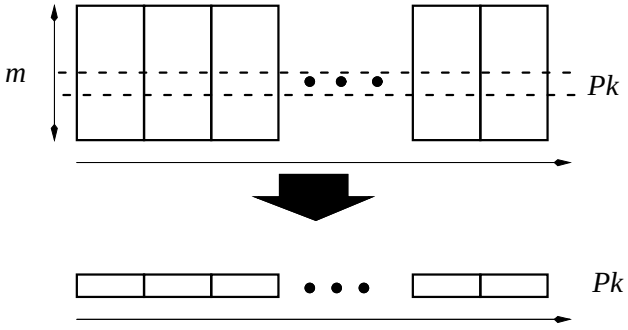


Figure 4.8: Scheduling dedicated tasks of size one

If tasks are parallel, then any m tasks can be scheduled in one time slot. Hence, by "scaling" the time line, the problem can be reformulated as $1|p_j = 1|\sum \bar{U}_j$. For an illustration see Figure 4.9. □

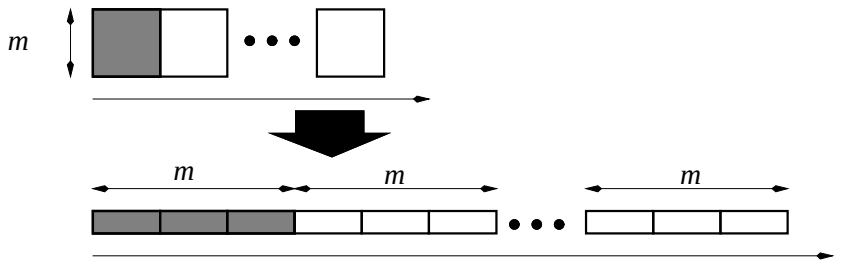
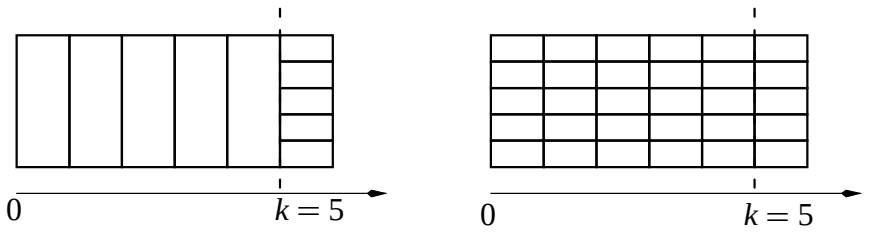


Figure 4.9: Scheduling parallel tasks of size one

Unfortunately, both EDD and LDD are not that good for the general case. See Figures 4.5 on page 165 and 4.5 on page 165. Furthermore, consider the following example. There are k large tasks T_j ($j = 1,\ldots,k$) with $s_j = m$ and $d(T_j) = k$, and $m(k+1)$ small tasks T_j ($j = k+1,\ldots,(m+1)(k+1)$) with $s_j = 1$ and $d(T_j) = k+1$. Then, the number of tasks in the *OPT* schedule $N_{\mathrm{OPT}} = m(k+1)$, but EDD schedules only k large tasks and m small tasks. For an illustration see Figure 4.10. Thus, as $k \to \infty$, the ratio tends to m.



a) EDD schedule b) OPT schedule

Figure 4.10: Scheduling k large and $m(k+1)$ small tasks in EDD order, for $k = 5$ and $m = 5$

4.2.5 Scheduling in Increasing Size

Consider the following single bin packing problem:

INSTANCE: A set $A = \{a_1, a_2, \ldots, a_n\}$ of n items and a single bin of size $m \in N$. Each item a_j ($j = 1,\ldots,n$) has size $s_j \in N$.

OBJECTIVE: Find a subset $A' \subseteq A$ such that $\sum_{j \in A'} s_j \leq m$ and such that $|A'|$ is maximized.

Informally, we are given a set of items an one bin, and out goal is to maximize the number of items in the bin.

It is not hard to see that 2-PARTITION can be simply reduced to the problem, that gives NP-hardness [GJ79]. Furthermore, the problem is just a version of KNAPSACK and BIN PACKING [GJ79], and there are a number of approximation algorithms are known [Hoc96, Vaz00, Hro01]. We consider a very simple one:

> FIRST FIT INCREASING (FFI):
> Add items into the bin in order of non-decreasing sizes. If the addition of an item results in the bin being "overloaded", reject this item and all later items.

Lemma 4.2.2. *Let OPT be the number of items in the optimal packing. Then, the number of items accepted by* FFI *is at least OPT* -1.

Proof. Assume w.l.o.g. that items $a_1, a_2, \ldots, a_k$ are accepted and all items a_{k+1}, a_{k+2}, $\ldots$, a_n are rejected by FFI. Then, there are k items in the bin, and the total size of packed items

$$S = \sum_{j=1}^{k} s_j \leq m \text{ and } S + s_{k+1} > m.$$

Clearly, replacing any of items $a_1, a_2, \ldots, a_k$, a_{k+1} by one of later rejected items

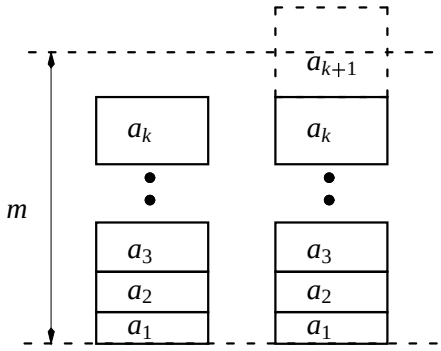


Figure 4.11: Items $a_1, a_2, \ldots, a_k$ and a_{k+1}

$a_{k+2}, \ldots, a_n$ can only increase the value of $S + a_{k+1}$. For an illustration see Figure 4.11. Hence, any set of $k+2$ items will "overload" the bin. Thus, the optimal value *OPT* at most $k+1$. □

Unfortunately, the algorithm FFI cannot be formally generalized for scheduling multiprocessor tasks. However, its main idea, that is processing of tasks by INCREASING SIZE (IS) – *non-decreasing order of sizes*, will be analyzed later in Section 4.3 for the parallel model and in Section 4.4 for the dedicated model. We will also use the main idea of "overloading", which is described in Lemma 4.2.2.

4.3 SCHEDULING OF PARALLEL TASKS

In this section we consider the following problem of scheduling parallel multiprocessor tasks. We are given a set $T = \{T_1, T_2, \ldots, T_n\}$ of n tasks and a set $P = \{P_1, P_2, \ldots, P_m\}$ of m processors. Each task T_j has a unit processing time $p_j = 1$, an integral due date $d(T_j) \in \{d_1, d_2, \ldots, d_g\}$, where $0 < d_1 < d_2 < \ldots < d_g = D$, and requires $s_j \in \{1, \ldots, m\}$ processors for its processing. The goal is to maximize the *throughput*, i.e. the number of *early* tasks T_j that meet their due dates $d(T_j)$.

4.3.1 Complexity

We start with the following result:

Theorem 4.3.1. *Problem* $P|\,\text{size}_j, p_j = 1, d_j = D|\sum \bar{U}_j$ *is strongly* NP*-hard.*

Proof. Problem 3-PARTITION can be formulated as follows [GJ79]:

INSTANCE: Set A of $3N$ elements, a bound $B \in \mathbf{Z}^+$, and a size $s(a) \in \mathbf{Z}^+$ for each $a \in A$ such that $B/4 < s(a) < B/2$ and such that $\sum_{a \in B} s(a) = NB$.

QUESTION: Can A be partitioned into N disjoint sets $A_1, A_2, \ldots, A_N$ such that, for $1 \leq i \leq N$, $\sum_{a \in A_i} s(a) = B$?

We transform 3-PARTITION to our problem as follows. We first define $m = B$ and form a set $P = \{P_1, P_2, \ldots, P_m\}$ of m processors. Next, we replace each element $a \in A$ by a single task T_a which has a unit processing time, a due date $d(T_a) = N$, and requires $s(a)$ processors in P. In total, there are $n = 3N$ tasks in $T = \{T_a : a \in A\}$ and all of them have a common due date $D = N$. Clearly, such an instance of our problem can be constructed in polynomial time. Furthermore, the answer to a given instance of 3-PARTITION is *YES* if and only if all tasks meet the common due date. Since 3-PARTITION is strongly NP-complete [GJ79], our problem is strongly NP-hard. □

The common due date problem $P|\,\text{size}_j, p_j = 1, d_j = D|\sum \bar{U}_j$ can be reformulated as the problem of finding a maximum cardinality subset of the given list of items which can be packed into a given number of bins with a given capacity. This is a special variant of the BIN PACKING problem [CLT78, ECL79] and the MULTIPLE KNAPSACK problem [Kel99, CK00, CKP00]. The later problem admits a polynomial approximation scheme (PTAS). Hence, we can conclude with the following:

Theorem 4.3.2. *There is a PTAS for problem* $P|\,\text{size}_j, p_j = 1, d_j = D|\sum \bar{U}_j$.

4.3.2 The Algorithm FFIS

Here we analyze the following algorithm for the parallel model:

> FIRST FIT INCREASING SIZE (FFIS):
> Select the tasks one by one in IS order. If the task can be completed before or at its due date, it is scheduled as early as possible. If the task cannot be assigned to meet its due date, it gets lost (it will not be scheduled).

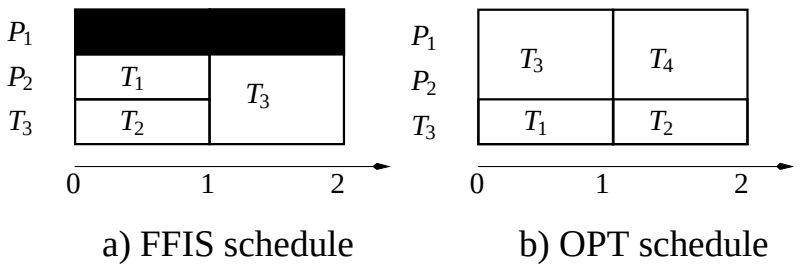
We start with the case when all tasks have a common due date:

Theorem 4.3.3. *For the common due date problem* $P|\,\text{size}_j, p_j = 1, d_j = D|\sum \bar{U}_j$, *the worst-case ratio*

$$R_{FFIS} = \frac{4}{3}.$$

Proof. As we discussed before, problem $P|\,\text{size}_j, p_j = 1, d_j = D|\sum \bar{U}_j$ can be reformulated as the bin packing problem for maximizing the number of items packed, which was studied by Coffman, Leung and Ting [CLT78]. They presented an algorithm called FFI and proved the tight asymptotic worst-case ratio is $4/3$. In fact, their proof is also valid for the absolute worst-case ratio. Furthermore, FFIS is similar to FFI. By using this result, we can prove that the worst-case ratio of FFIS is not greater than $4/3$.

We can also show that the bound $4/3$ for R_{FFIS} is tight if $m \geq 3$. Consider the following example. There are two small tasks, T_1 and T_2, each of which requires only one processor, and there are two large tasks, T_3 and T_4, each of which requires $m-1$ processors. Let $D = 2$ be a common due date. Clearly, FFIS can only schedule three of them. However, all four tasks can be scheduled. For an illustration see Figure 4.12 on the following page. □



a) FFIS schedule b) OPT schedule

Figure 4.12: Scheduling tasks T_1, T_2, T_3 and T_4

Now we can prove the following main result:

Theorem 4.3.4. *For the general problem* $P|\,\text{size}_j, p_j = 1|\,\sum \bar{U}_j$*, the worst-case ratio*

$$R_{FFIS} \;=\; 2.$$

Proof. We first prove that $R_{FFIS} \leq 2$. Consider the OPT schedule with N_{OPT} tasks and the FFIS schedule with N_{FFIS} tasks, respectively. Remove from the OPT schedule all tasks which are involved in the FFIS schedule. Let ℓ_t be the number of left tasks in each time slot I_t.

If we prove that

$$\sum_{t=1}^{D} \ell_t \;\leq\; \sum_{t=1}^{D} N_t = N_{FFIS}, \tag{4.1}$$

we will have

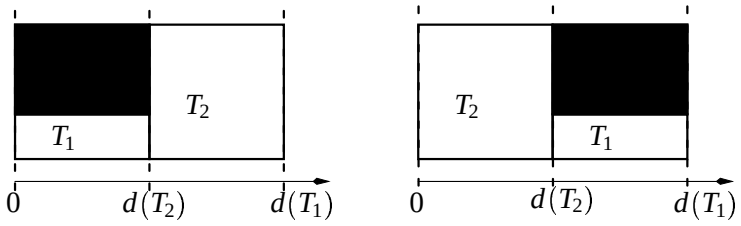
$$R_{FFIS} \;\leq\; \frac{N_{\text{OPT}}}{N_{FFIS}} \;\leq\; 2. \tag{4.2}$$

Recall that all ℓ_t left tasks in the OPT schedule are lost in the FFIS schedule. Since FFIS schedules the tasks in non-decreasing order of size, in each time slot I_t the left ℓ_t tasks of the OPT schedule are not smaller in size than those of the FFIS schedule. Hence, the number of scheduled tasks N_t cannot be less than ℓ_t. Thus, we have

$$N_t \;\geq\; \ell_t, \text{ for } t = 1, \ldots, D,$$

and both (4.1), (4.2) hold.

The bound is tight. Consider the following example. There are two tasks T_1 and T_2. Task T_1 has size $s_1 = 1$ and date $d(T_1) = 2$, whereas task T_2 has size $s_2 = m$ and due date $d(T_2) = 1$. In the optimal schedule both T_1 and T_2 are scheduled, but the algorithm FFIS only accepts task T_2. For an illustration see Figure 4.13 on the next page. □



a) FFIS schedule b) OPT schedule

Figure 4.13: Scheduling tasks T_1 and T_2

4.3.3 The Algorithm LFIS

Here we analyze the following algorithm for the parallel model:

> LAST FIT INCREASING SIZE (LFIS):
> Select the tasks one by one in IS order. If the task can be completed before or at its due date, it is scheduled as later as possible. If the task cannot be assigned to meet its due date, it gets lost (it will not be scheduled).

Indeed, the performance ratio of FFIS and LFIS is the same for the common due date problem. However, LFIS performs better for the general problem. We start with the following simple result:

Lemma 4.3.5. *For the general problem* $P|\,\text{size}_j, p_j = 1|\sum \bar{U}_j$, *the worst-case ratio*

$$R_{LFIS} \;\geq\; 2 \;-\; \frac{1}{m},$$

where $m \geq 2$ *is the number of processors.*

Proof. Consider the following example. For an illustration see Figure 4.14 on the following page. There are m small tasks T_j ($j = 1, \ldots, m$) with size $s_j = 1$ and due date $d(T_j) = j$. These small tasks are denoted by s in Figure 4.14 a) and b). In addition, there are $m-1$ large tasks T_j ($j = m+1, \ldots, 2m-1$) with size $s_j = m$ and due date $d(T_j) = m$. These large tasks are denoted by L in Figure 4.14 b).

Clearly, all tasks are early in the optimal schedule. We can schedule all small tasks in the first time slot I_1 and schedule all large tasks in later time slots, see

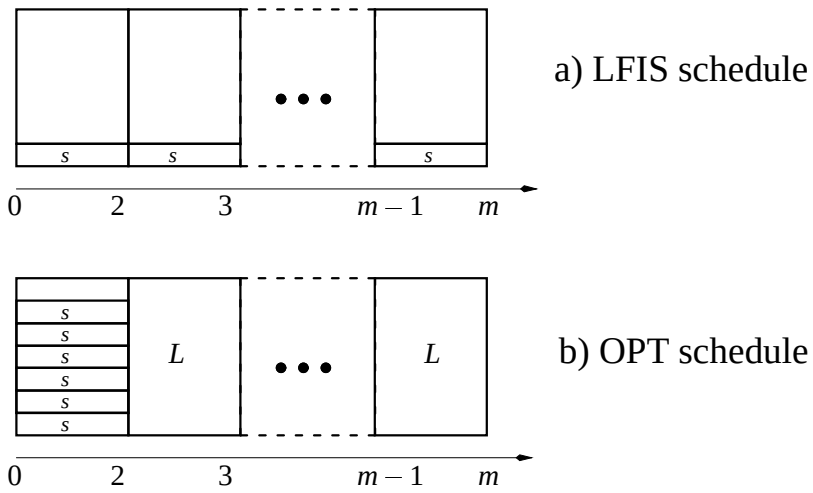


Figure 4.14: Scheduling small and large tasks

Figure 4.14 b). However, the algorithm LFIS only accepts small tasks, see Figure 4.14 a). Hence the worst-case ratio

$$R_{LFIS} \geq \frac{2m-1}{m} = 2-\frac{1}{m}.$$

□

Now we can prove the following main result:

Theorem 4.3.6. *For problem* $P|\,\text{size}_j, p_j = 1|\sum \bar{U}_j$*, the worst-case ratio*

$$R_{LFIS} = 2-\frac{1}{m}.$$

Proof. By Lemma 4.3.5, we only need to show that $R_{LFIS} \leq 2-1/m$. In the following we will prove this by a contradiction.

Assume that $R_{LFIS} > 2-1/m$. Accordingly, let $T_{\min}$ be the minimum task set, in terms of the number of tasks, such that

$$\frac{N_{\text{OPT}}(T_{\min})}{N_{LFIS}(T_{\min})} > 2 - \frac{1}{m}. \tag{4.3}$$

Then, for all task sets T with $|T| < |T_{\min}|$, it follows

$$\frac{N_{\text{OPT}}(T)}{N_{LFIS}(T)} \leq 2 - \frac{1}{m}. \tag{4.4}$$

Consider the OPT schedule for $T_{\min}$. There are D time slots I_t, $t = 1,\ldots,D$. Assume that there exists one task T_j in $T_{\min}$ which is lost by the optimal algorithm OPT. In this case we have that

$$N_{\mathrm{OPT}}(T_{\min}) \;=\; N_{\mathrm{OPT}}(T_{\min} \setminus \{T_j\}).$$

Furthermore, since LFIS accepts T_j in $T_{\min}$, the lost tasks have larger size than T_j. Hence, LFIS working on $T_{\min} \setminus \{T_j\}$ cannot accept more than $N_{LFIS}(T_{\min})$ tasks. Thus, from (4.3) we have that

$$\frac{N_{\mathrm{OPT}}(T_{\min} \setminus \{T_j\})}{N_{LFIS}(T_{\min} \setminus \{T_j\})} \;\geq\; \frac{N_{\mathrm{OPT}}(T_{\min})}{N_{LFIS}(T_{\min})} \;\geq\; 2 \;-\; \frac{1}{m}.$$

This gives a contradiction to (4.4).

Thus, w.l.o.g. we can assume that all tasks in $T_{\min}$ are accepted by the optimal algorithm OPT. Then,

$$|T_{\min}| \;=\; N_{\mathrm{OPT}}(T_{\min}), \tag{4.5}$$

and the total size of tasks in $T_{\min}$

$$S_{\min} \;\leq\; m \cdot D. \tag{4.6}$$

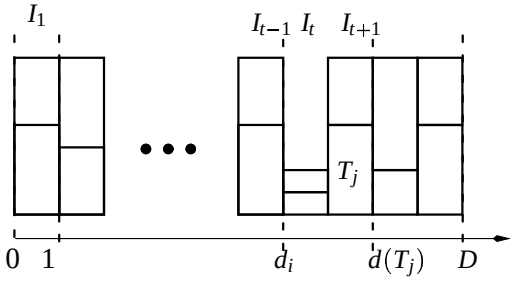


Figure 4.15: A tasks T_j with $d(T_j) > d_i$

Consider the LFIS schedule for $T_{\min}$. There are D time slots I_t, $t = 1,\ldots,D$. Assume that there is exactly one time slot I_1 in the schedule. If it is open, then all the tasks of $T_{\min}$ are accepted. In this case,

$$N_{LFIS}(T_{\min}) \;=\; |T_{\min}| \;\geq\; N_{\mathrm{OPT}}(T_{\min}).$$

This gives a contradiction to (4.3).

Now assume that there is a time slot I_t in the LFIS schedule such that $1 < d_i < t \leq D$ and it is open. Let k be the number of tasks T_j with due dates $d(T_j) > d_i$.

Clearly, all these k tasks are accepted by LFIS. For an illustration see 4.15 on the page before. From (4.3) we have

$$N_{\mathrm{OPT}}(T_{\min}) \;>\; \left(2 \;-\; \frac{1}{m}\right) N_{LFIS} \;-\; k \;+\; \frac{k}{m}$$

and

$$\frac{N_{\mathrm{OPT}}(T_{\min}) \;-\; k}{(N_{LFIS} \;-\; k)} \;>\; \left(2 \;-\; \frac{1}{m}\right).$$

Hence, by removing these k tasks from $T_{\min}$ we obtain a smaller set $T_{\min}$. This gives a contradiction to (4.4).

Thus, w.l.o.g. we can assume that there are no open time slots in the LFIS schedule for $T_{\min}$. Then, each time slot has at least one task. Hence, the total number of accepted tasks

$$N_{LFIS}(T_{\min}) \geq D. \tag{4.7}$$

Let h be such that

$$|T_{\min}| \;=\; N_{\mathrm{OPT}}(T_{\min}) \;=\; N_{LFIS}(T_{\min}) \;+\; h.$$

Informally, h is the number of tasks in $T_{\min}$ which are accepted by OPT, but lost by LFIS. Then, from (4.3) we have

$$N_{LFIS}(T_{\min}) \;+\; h \;>\; N_{LFIS}(T_{\min}) \left(2 \;-\; \frac{1}{m}\right)$$

and from (4.7) it follows that

$$h \;>\; N_{LFIS}(T_{\min}) \left(1 \;-\; \frac{1}{m}\right) \;\geq\; D \cdot \left(1 \;-\; \frac{1}{m}\right). \tag{4.8}$$

Consider the LFIS schedule for $T_{\min}$. There are D time slots I_t, $t = 1, \ldots, D$. Consider a lost task T_j. For illustration see Figure 4.16 on the facing page. There are three simple properties for the schedule. Task T_j cannot fit into any of time slots I_t, $t = 1, \ldots, d(T_j)$. The size of T_j is not less than the size of any accepted task in time slots I_t, $t = 1, \ldots, d(T_j)$. If there is a task T_k which is accepted not later than time slot $I_{d(T_j)}$ and its due date $d(T_k) > d(T_j)$, then T_k cannot fit into any of time slots I_t, $t = d(T_j) + 1, \ldots, d(T_k)$.

Assume that we "load" this lost task T_j into the LFIS schedule. For illustration see Figure 4.17 on the next page. There are two possibilities. Either T_j "overloads"

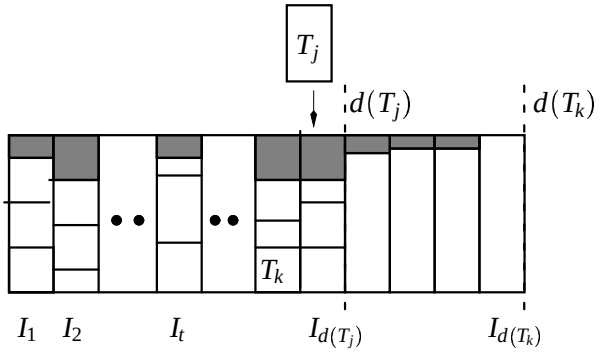


Figure 4.16: Tasks T_j and T_k in the LFIS schedule

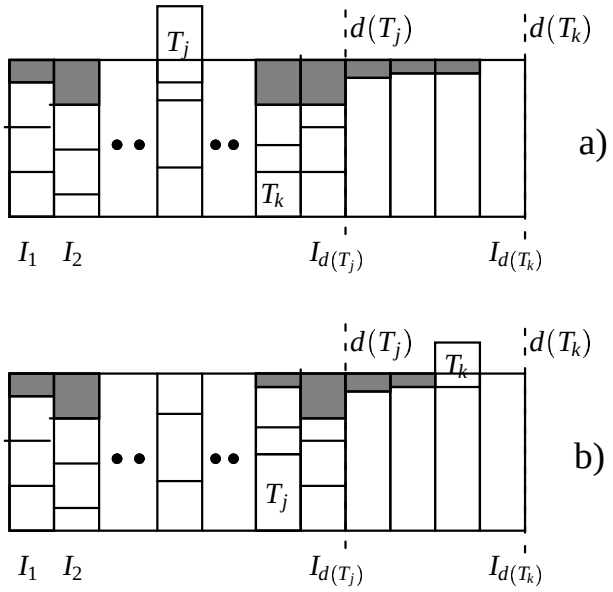


Figure 4.17: Loading tasks T_j and T_k

one of time slots I_t, $t = 1, \ldots, d(T_j)$, see Figure 4.17 a), or T_k "overloads" one of time slots I_t, $t = d(T_j)+1, \ldots, d(T_k)$, see Figure 4.17 b). In both cases we create at least one "overloaded" time slot.

By using this simple procedure we can load lost tasks one by one into the LFIS schedule. Furthermore, we can ensure that in every time slot there is at most one "overloading" task. If we load all h lost tasks, there are at least h "overloaded" time slots, and the total size of tasks in each of them is at least $m+1$. If $h = D$, then the total size of tasks is at least $D(m+1)$, that gives a contradiction to (4.6). Hence, the number of lost tasks and the number "overloaded" time slots $h < D$. Furthermore, all tasks of $T_{\min}$ are scheduled.

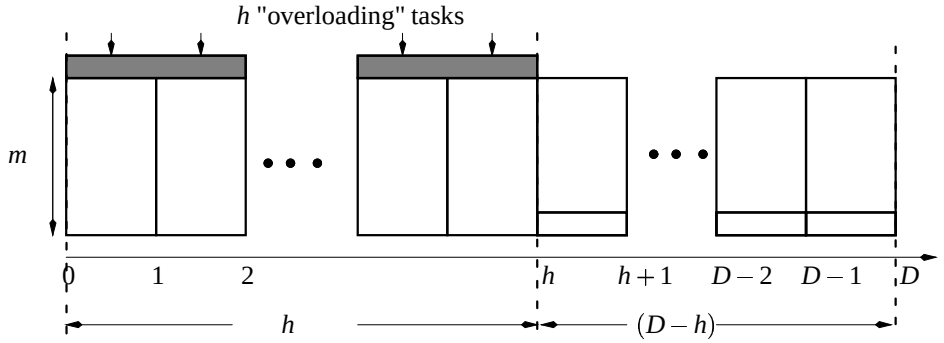


Figure 4.18: "Overloading"

Remember that all time slots are closed in the LFIS schedule. Hence, there is at least one task in each time slot I_t, $t = 1, \ldots, D$. For an illustration see Figure 4.18. Here, for simplicity, we have put h "overloaded" time slots at the beginning of the schedule. Then, the total size of the tasks in the first h time slots is at least $(m+1)h$ and the total size of the tasks in the last $D-h$ time slots is at least $D-h$.

Finally, the total size of $T_{\min}$ is bounded by

$$h(m+1) + (D-h) \leq S_{\min} \leq m \cdot D. \tag{4.9}$$

Hence,

$$h \leq \frac{D(m-1)}{m} = D \cdot \left(1 - \frac{1}{m}\right).$$

This gives a contradiction to (4.8). Hence, our assumption is wrong. This completes the proof. □

By using the ideas from Theorem 4.3.6, we can also prove the following result:

Theorem 4.3.7. *If all tasks are small in the general problem* $P|\,\mathrm{size}_j, p_j = 1|\,\sum \bar{U}_j$, *then the worst-case ratio*

$$R_{LFIS} \;\leq\; \frac{3}{2} \;-\; \frac{1}{2m-2}.$$

Proof. As in the proof of Theorem 4.3.6, let $T_{\min}$ be the minimum task set such that

$$\frac{N_{\mathrm{OPT}}(T_{\min})}{N_{LFIS}(T_{\min})} \;>\; \frac{3}{2} \;+\; \frac{1}{2m \;-\; 2}. \tag{4.10}$$

Similarly, we can show the following. The optimal algorithm OPT accepts all tasks in $T_{\min}$. Hence,

$$N_{\mathrm{OPT}}(T_{\min}) \;=\; |T_{\min}|,$$

and the total size of $T_{\min}$ is bounded by

$$S_{\min} \;\leq\; m \cdot D.$$

Regarding the LFIS schedule, all time slots are closed. Remembering that all tasks are small, there are at least two tasks in each time slot and

$$N_{LFIS}(T_{\min}) \;\geq\; 2D.$$

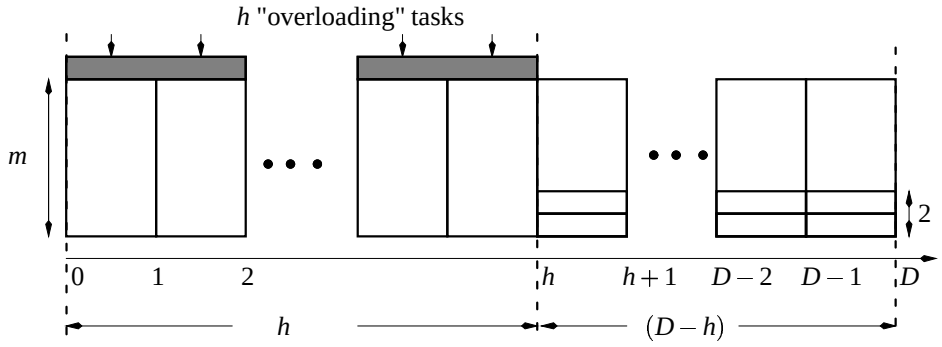


Figure 4.19: "Overloading"

Let h be the number of tasks in $T_{\min}$ which are lost by LFIS. For an illustration see Figure 4.19. Then, as in (4.9), the total size is bounded by

$$h(m \;+\; 1) \;+\; 2(D \;-\; h) \;\leq\; S_{\min} \;\leq\; m \cdot D,$$

and

$$h \leq D \cdot \frac{(m-2)}{(m-1)}.$$

Thus,

$$\begin{aligned}\frac{N_{\mathrm{OPT}}(T_{\min})}{N_{LFIS}(T_{\min})} &= \frac{N_{LFIS}(T_{\min}) + h}{N_{LFIS}(T_{\min})} \\ &\leq 1 + \frac{m-2}{2m-2} \\ &= \frac{3}{2} + \frac{1}{2m-2}.\end{aligned}$$

This gives a contradiction to (4.10). Hence, our assumption is wrong. This completes the proof. □

4.3.4 A Hybrid Algorithm

It seems that both FFIS and LFIS algorithms attach too much importance to the task size. In some sense, FFIS "groups" small tasks together, whereas LFIS "spreads" them. Can we do something better?

Indeed, we can combine all our ideas together. Informally, we proceed as follows. First, we split all tasks into small and large ones. Then, we schedule the set of small tasks by the algorithm LFIS. For an illustration see Figure 4.20 a). Clearly, scheduling a large tasks in a closed time slot can only decrease the number of tasks accepted. Hence, we need to schedule large tasks in open time slots. It can happen that there are single small tasks in open time slots. From one side, these small tasks can "block" some large tasks. From another side, they can be scheduled together. Here, we simply reschedule small tasks in open time slots. We select these tasks one by one as they appear in the schedule and reschedule them by First-Fit. For an illustration see Figure 4.20 b). One can see that all new closed time slots include at least two small tasks and there is at most one open time slot with small tasks at the end. Finally, we schedule the set of large tasks by using either First-Fit in EDD order or Last-Fit in LDD order.

Now we summarize the algorithm as follows:

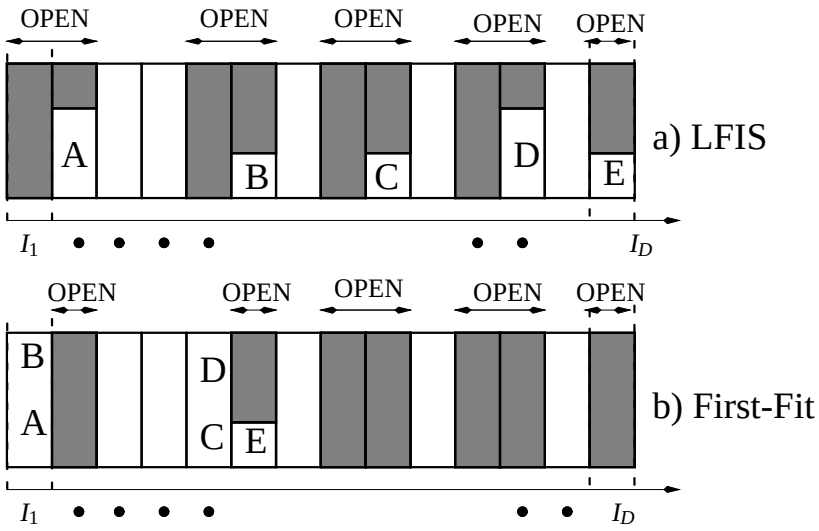


Figure 4.20: Scheduling small tasks

HYBRID ALGORITHM (HA):

1. Define small and large tasks.
2. Schedule the set of small tasks by LFIS.
3. If there are no open time slots with small tasks, go to Step 6.
4. Reindex small tasks in open time slots with an initial sequence $1,2,3,\ldots$ in the "left to right" and "bottom to top" manner. For an illustration see Figure 4.21 a).
5. Reschedule small tasks in open time slots by using First-Fit in the given order. For an illustration see Figure 4.21 b).
6. Schedule the set of large tasks in open time slots by using either First-Fit in EDD order or Last-Fit in LDD order.

We fisrt start with the following simple result:

Lemma 4.3.8. *For the general problem problem* $P|\,\text{size}_j, p_j = 1|\sum \bar{U}_j$, *the worst-case ratio*

$$R_{HA} \geq \begin{cases} 3/2 - 1/(2k-2), & \text{if} \quad m = 3k, \\ 3/2 - 1/(2k-1), & \text{if} \quad m = 3k+1, \\ 3/2 - 1/(2k), & \text{if} \quad m = 3k+2. \end{cases}$$

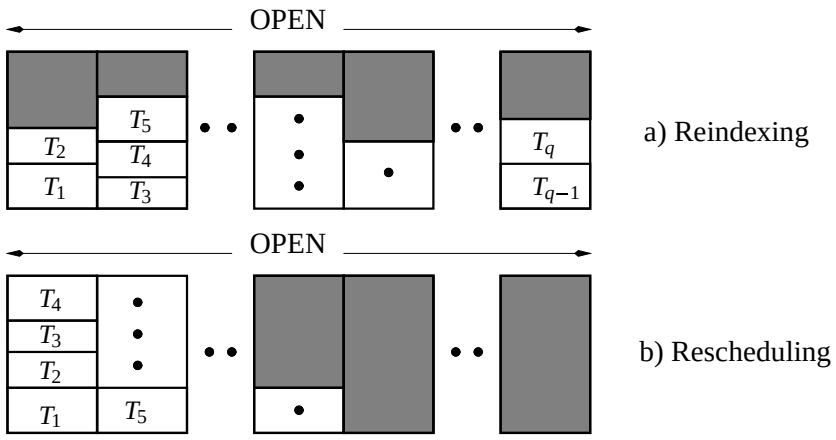


Figure 4.21: Reindexing and rescheduling of small tasks

Proof. Consider the following example. There are $3n$ tasks. The value of n relates with the value of m and it will be specified later. For $i = 1, \ldots, n$, there are three tasks X_i, Y_i and Z_i which have the due date equal to i. Their sizes are denoted by x_i, y_i and z_i, respectively. We define the values of x_i, y_i and z_i, $i = 1, \ldots, n-1$, such that

- $x_i, y_i \leq z_i$,
- $x_i \geq x_{i+1}$; $y_i \geq y_{i+1}$; $z_i < z_{i+1}$,
- $x_i + y_i + z_i = m + 1$,
- and $x_{i+1} + y_{i+1} + z_i = m$.

The exact values are specified below.

Clearly, the algorithm HA schedules two tasks X_i and Y_i in time slot I_i, $i = 1, \ldots, n$, and all n tasks Z_i are lost. Then, the number of tasks accepted

$$N_{HA} = 2n.$$

For an illustration see Figure 4.22 a). Clearly, the optimal algorithm OPT can schedule three tasks X_{i+1}, Y_{i+1} and Z_i together in time slot I_i, $i = 1, \ldots, n-1$, and schedule task Z_n in time slot I_n. Then, the number of tasks accepted

$$N_{\mathrm{OPT}} = 3n - 2.$$

For an illustration see Figure 4.22 b). Thus, the worst case ratio

$$R_{HA} \geq \frac{N_{\mathrm{OPT}}}{N_{HA}} = \frac{3n - 2}{2n} = \frac{3}{2} - \frac{1}{n}.$$

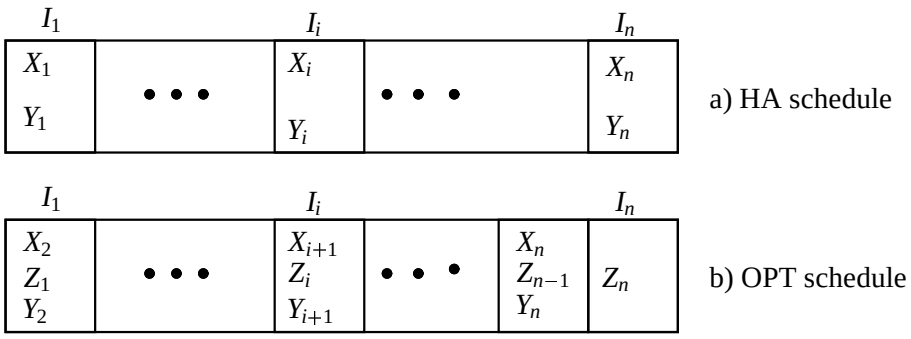


Figure 4.22: Scheduling tasks X_i, Y_i, Z_i

Now we specify the exact values by considering the following three cases:

1. If $m = 3k$, then $n = 2k-2$, and $x_i + y_i = 2k - i$, $z_i = k + i + 1$, for $i = 1, \ldots, 2k-2$.
2. If $m = 3k+1$, then $n = 2k-1$, and $x_i + y_i = 2k - i + 1$, $z_i = k + i + 1$, for $i = 1, \ldots, 2k-1$.
3. If $m = 3k+2$, then $n = 2k$, and $x_i + y_i = 2k - i + 2$, $z_i = k + i + 1$, for $i = 1, \ldots, 2k$.

This completes the proof. □

Now we are ready to prove the following main result:

Theorem 4.3.9. *For the general problem* $P|\,\text{size}_j, p_j = 1|\,\sum \bar{U}_j$, *the worst-case ratio*

$$R_{HA} \leq \frac{3}{2} - \frac{1}{2m-2}.$$

Proof. We first consider the following two cases: (I): There are no large tasks after Step 1; (II) There are no open time slots with small tasks after Step 2.

Case (I). Only small tasks are lost by the algorithm HA. By Lemma 4.3.7,

$$R_{HA} \leq \frac{3}{2} - \frac{1}{2m-2}.$$

Case (II). Only small tasks close time slots at Step 2. Hence, no large task can fit into a closed time slot. Furthermore, replacing any small task in closed time slots

can only decrease the number of tasks accepted. By Lemma 4.2.1, the algorithm HA is optimal at Step 6. Hence, as in case (I), we have

$$R_{HA} \leq \frac{3}{2} - \frac{1}{2m-2}.$$

Now we consider the general case when the algorithm HA proceeds all Steps 1, 2, 3 and Steps 4, 5, 6. Consider the LFIS schedule. For an illustration see Figure 4.23. For simplicity, we assume that after Step 2 all time slots $I_1, \ldots, I_{t-1}$ are closed with small tasks, after Step 5 all time slots $I_t, \ldots, I_k$ include rescheduled small tasks, and after Step 6 all time slots $I_k, \ldots, I_D$ include large tasks. Here time slot I_k can include several small task and one large task.

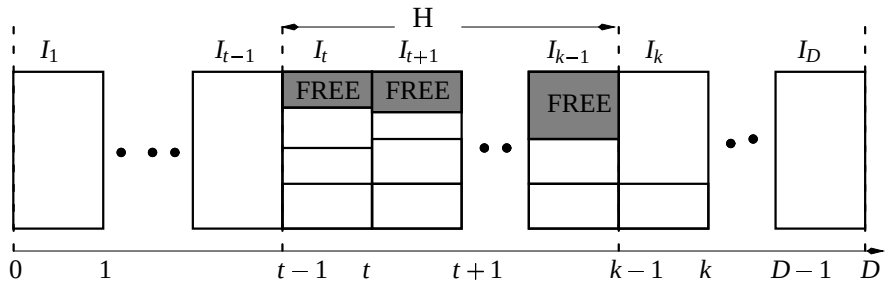


Figure 4.23: The LFIS schedule

Clearly, the LFIS schedule in time slots $I_1, \ldots, I_{t-1}$ and $I_k, \ldots, I_D$ can be handled as in case (I) and (II). We only need to deal with the the LFIS schedule in time slots $I_t, \ldots, I_{k-1}$.

Each of time slots $I_t, \ldots, I_{k-1}$ is closed and consists of at least two small tasks. Let $H = k - t + 1$ be the number of these time slots. Then, the number of small tasks accepted

$$\#S \geq 2H. \tag{4.11}$$

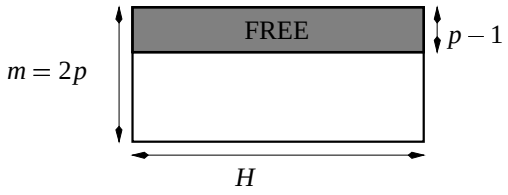


Figure 4.24: Free time for $m = 2p$

Assume that the number of processors m is even, and let $m = 2p$. Then, any small task is at most p in size, and any large task is at least $p+1$ in size. Since small tasks are scheduled by using First-Fit, there are at most $p-1$ "free time" in each of these H time slots. For an illustration see Figure 4.23 on the facing page. The total "free time" is at most

$$H(p\ -\ 1) \tag{4.12}$$

Hence, the total number of large tasks which can be scheduled by the optimal algorithm OPT is at most

$$\#L = H(p\ -\ 1)/(p\ +\ 1). \tag{4.13}$$

Then, the number of small and large tasks accepted by OPT is at most $\#S + \#L$. Hence, from (4.11) and (4.13), we have that

$$\begin{aligned}
\frac{N_{\mathrm{OPT}}}{N_{HA}} &\leq \frac{\#S\ +\ \#L}{\#S} \\
&= 1\ +\ \frac{\#L}{\#S} \\
&\leq 1\ +\ \frac{H(p\ -\ 1)}{2H(p\ +\ 1)} \\
&\leq 1\ +\ \frac{H(p+1) - 2H}{2H(p\ +\ 1)} \\
&= \frac{3}{2}\ -\ \frac{1}{p\ +\ 1} \\
&= \frac{3}{2}\ -\ \frac{2}{m+2}.
\end{aligned} \tag{4.14}$$

Notice if $p = 1$, there is no "free time". Hence, we assume that $p \geq 2$ and $m = 2p \geq 4$.

Assume that the number of processors m is odd. Let $m = 2p+1$. Then, any small task is at most p in size, and any large task is at least $p+1$ in size. Hence, both

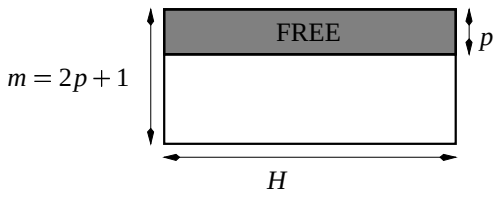


Figure 4.25: Free time for $m = 2p+1$

(4.12) and (4.13) remain the same. As in (4.14), we have that

$$\begin{aligned} \frac{N_{\mathrm{OPT}}}{N_{HA}} &\leq \frac{\#S + \#L}{\#S} \\ &= 1 + \frac{\#L}{\#S} \\ &\leq \frac{3}{2} - \frac{1}{p+1} \\ &= \frac{3}{2} - \frac{2}{m+1}. \end{aligned} \tag{4.15}$$

Notice if $p = 1$, there is no "free time". Hence, we assume that $p \geq 2$ and $m = 2p+1 \geq 5$.

Finally, for all cases we can conclude that the algorithm HA only fails on small tasks, and the worst case ratio

$$R_{HA} \leq \frac{3}{2} - \frac{1}{2m-2}.$$

□

4.4 SCHEDULING OF DEDICATED TASKS

Here we consider the following problem of scheduling dedicated multiprocessor tasks. We are given a set $T = \{T_1, T_2, \ldots, T_n\}$ of n tasks and a set $P = \{P_1, P_2, \ldots, P_m\}$ of m processors. Each task T_j has a unit processing time $p_j = 1$, an integral due date $d(T_j) \in \{d_1, d_2, \ldots, d_g\}$, where $0 < d_1 < d_2 < \ldots < d_g = D$, and requires the processors with indices in $\tau_j \subseteq \{1, \ldots, m\}$. The goal is to maximize the *throughput*, i.e. the number of *early* tasks T_j that meet their due dates $d(T_j)$.

4.4.1 Complexity

We start with the following result:

Theorem 4.4.1. *The common due date problem* $P|\text{fix}_j, p_j = 1, d_j = 1|\sum \bar{U}_j$ *is* NP*-hard. Furthermore, it cannot be approximated within a factor of* $m^{\frac{1}{2}-\varepsilon}$ *for any given* $\varepsilon > 0$*, unless* NP = ZPP.

Proof. Our problem can be formulated as follows:

INSTANCE: A set $T = \{T_1, T_2, \ldots, T_n\}$ of n tasks and a set $P = \{P_1, P_2, \ldots, P_m\}$ of m processors. Each task T_j has a unit processing time $p_j = 1$ and requires the processors with indices in $\tau_j \subseteq \{1, \ldots, m\}$.

OBJECTIVE: Find a subset $T' \subseteq T$ such that for every pair of tasks T_i and T_j in T' it holds $\tau_i \cap \tau_j = \emptyset$ and such that $|T'|$ is maximized.

Problem MAXIMUM CLIQUE can be formulated as follows [GJ79]:

INSTANCE: A graph $G(V,E)$ with $|V| = n$.

OBJECTIVE: Find a subset $V' \subseteq V$ such that every pair of vertices v and u in V' it holds $\{v,u\} \in E$ and such that $|V|$ is maximized.

We can transform MAXIMUM CLIQUE to our problem as follows. We define

$$T(G) = \{T_v | v \in V\} \text{ and } P(G) = \{P_e | e \notin E\}. \tag{4.16}$$

Foe each task T_v we define $p_v = 1$ and

$$\tau_v = \{e | e = \{v,u\} \notin E\}. \tag{4.17}$$

Clearly,

$$|T(G)| = |V| = n \tag{4.18}$$

and

$$m = |P(G)| = |V|(|V| - 1)/2 - |E| \leq n(n-1)/2. \tag{4.19}$$

Let $T' \subseteq T(G)$ be a solution to our problem. Then, for every pair of tasks T_v and T_u in T' it holds

$$\tau_u \cap \tau_v = \emptyset.$$

We can define

$$V' = \{v | T_v \in T'\}. \tag{4.20}$$

By (4.16) and (4.17), for every pair v and u in V' their edge $e = (u,v)$ is in E. Hence, V' is a clique in G, by (4.20) its size

$$|V'| \;=\; |T'|. \tag{4.21}$$

In other words, finding a maximal clique in graph $G(V,E)$ is equivalent to finding a maximal subset of tasks in $T(G)$ with respect to $P(G)$. Furthermore, the objective value remains the same.

It is well-known that MAXIMUM CLIQUE is NP-hard [GJ79], and it cannot be approximated within a factor of $n^{1-\varepsilon}$ for any given $\varepsilon > 0$, unless NP = ZPP [Has99]. Due to the transformation, our problem is NP-hard. Furthermore, due to (4.19), (4.20) and (4.21), if for some $\varepsilon_0 > 0$ the value of $|T'|$ is within a factor of $m^{\frac{1}{2}-\varepsilon_0}$ of the optimum, then for some $\varepsilon > 0$ the value of $|V'|$ is within a factor of

$$m^{\frac{1}{2}-\varepsilon_0} \leq (n(n-1)/2)^{\frac{1}{2}-\varepsilon_0} = n^{1-\varepsilon}$$

of the optimum. Hence, our problem cannot be approximated within a factor of $m^{\frac{1}{2}-\varepsilon}$ for any given $\varepsilon > 0$, unless NP = ZPP. □

4.4.2 The FFIS and LFIS Algorithms

Here we analyze both FFIS and LFIS algorithms presented in Sections 4.3.2,4.3.3, respectively. We first consider the common due date problem, and prove the following main result:

Theorem 4.4.2. *For problem* $P|\text{fix}_j, p_j = 1, d_j = D|\sum \bar{U}_j$, *the worst case ratio of* FFIS *and* LFIS *is at least* $\sqrt{m}$ *and at most* $\sqrt{m}+1$.

Proof. We only prove the result for the algorithm FFIS. However, the result for LFIS will follow from our proof as well.

As before, we use N_{OPT} and N_{FFIS} to denote the number of early tasks accepted by the optimal algorithm *OPT* and by algorithm FFIS, respectively. For time slot $I_t = [t-1,t]$ ($t = 1,\ldots,D$), let acc_t be the set of tasks accepted by FFIS in I_t. Assume w.l.o.g. that there are k tasks in acc_t, denoted by $T_1,\ldots,T_k$, and $|\tau_1| \leq |\tau_2| \leq \cdots \leq |\tau_k|$.

Observe the following facts for task T_i, $i = 1,\ldots,k$:

- Task T_i occupies $|\tau_i|$ processors. Thus, at most $|\tau_i|$ lost tasks can be accepted if T_i is removed.
- For any lost task T_j it holds $|\tau_j| \geq |\tau_i|$. Thus, at most $m/|\tau_i|$ lost tasks can be accepted if T_i is removed.

Combining, at most

$$\min\{|\tau_i|, m/|\tau_i|\} \leq \sqrt{m}$$

tasks can be accepted if task T_i is removed.

Hence, for each time slot I_t, $t = 1, \ldots, D$, the maximum number of tasks which can be accepted is at most

$$|acc_t| + |acc_t| \cdot \sqrt{m} \leq (1 + \sqrt{m}) \cdot |acc_t|.$$

Summarizing, we have

$$\begin{aligned} N_{\text{OPT}} &\leq \sum_{t=1}^{D} (1 + \sqrt{m}) \cdot |acc_t| \\ &= (1 + \sqrt{m}) \cdot \left(\sum_{t=1}^{D} |acc_t| \right) \\ &= (1 + \sqrt{m}) \cdot N_{FFIS}. \end{aligned}$$

Thus, the worst-case ratio

$$R_{FFIS} \leq \sqrt{m} + 1.$$

Consider the following simple example. Let $m = q^2$. There are T_1, T_2, ..., T_q, T_{q+1} tasks, and the common due date $D = 1$. For each task T_j ($j = 1, \ldots, q+1$) we define $\tau_j \subseteq \{1, 2, \ldots, m\}$ such that $|\tau_j| = \sqrt{m}$ and such that the first q tasks T_1, T_2, ..., T_q are compatible with in pairs, but the last task T_{q+1}. For an illustration see Figure 4.26.

Then, in the worst case, FFIS only accepts task T_{q+1}. However, the optimal algorithm OPT can accept q tasks T_1, T_2, ..., T_q. Thus,

$$R_{FFIS} \geq \frac{N_{\text{OPT}}}{N_{FFIS}} = \frac{\sqrt{m}}{1} = \sqrt{m}.$$

□

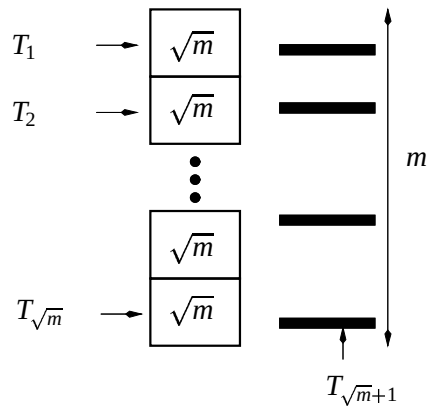


Figure 4.26: Tasks $T_1, T_2, \ldots, T_q$ and T_{q+1}, where $m = q^2$

Both bounds on the performance ratio in Theorem 4.4.2 are valid only for general m. However, for some specified value of m, algorithms can have a better performance ratio. Indeed, for $m = 2$ algorithms output an optimal schedule. Furthermore, we can prove the following:

Lemma 4.4.3. *For the three-processor problem* $P3|\,\mathrm{fix}_j, p_j = 1, D = d_j|\sum \bar{U}_j$, *both algorithms* FFIS *and* LFIS *have the worst case ratio* $4/3$.

Proof. We only prove the result for the algorithm FFIS. However, the result for LFIS will follow from our proof as well.

Consider the following simple example. There are four tasks T_1, T_2, T_3, T_4, where $\tau_1 = \{1\}$, $\tau_2 = \{3\}$, $\tau_3 = \{2,3\}$ and $\tau_4 = \{1,2\}$. The common due date $D = 2$. The optimal algorithm OPT accepts all tasks, whereas FFIS rejects either T_3 or T_4. Hence, the worst case ratio

$$R_{FFIS} \geq \frac{4}{3}.$$

Clearly, if there is an empty time slot I_t in the FFIS schedule, we can claim that it is optimal.

Assume that there is an time slot I_t with a single task T_j. Assume also that the size of T_j is equal to one. Then, all tasks which are compatible with T_j are accepted by FFIS before I_t, whereas all task accepted by FFIS after I_t are incompatible with T_j. Since the size of T_j is one, all later tasks are incompatible in pairs and can be only scheduled one per time slot. Hence, the FFIS schedule is optimal.

In the worst case, the FFIS schedule is not optimal. However, any time slot contains either at least two tasks of size one or one task of size two. In other

words, there are three processors and in any time slot there is at most one idle processor. Clearly, all lost tasks are greater in size than the accepted ones. Hence, the number of accepted tasks by the optimal algorithm *OPT*

$$N_{OPT} \leq N_{FFIS} + \frac{1}{3} \cdot N_{FFIS} \leq \frac{4}{3} \cdot N_{FFIS}.$$

Hence, the worst case ratio

$$R_{FFIS} \leq \frac{4}{3}.$$

□

Finally, we consider the general case that each task has individual due date.

Theorem 4.4.4. *For the general problem* $P|\text{fix}_j, p_j = 1|\sum \bar{U}_j$*, both algorithms* FFIS *and* LFIS *have the worst case ratio at least* $\sqrt{m}$ *and at most* $\sqrt{m}+1$.

Proof. Informally, we follow similar ideas as in Theorem 4.4.2. Consider FFIS or LFIS. If there are open time slots in the output schedule, then some parts of the schedule are optimal and cab be discarded in the worst case analysis. For every closed time slot, the number of lost tasks which can be "potentially" accepted is a factor of $\sqrt{m}$ of the number of accepted tasks. Hence, we can bound the number of tasks accepted by the optimal algorithm OPT as a factor of $(1+\sqrt{m})$ the number of tasks accepted by FFIS or LFIS. The worst case ratio is bounded by $(1+\sqrt{m})$. □

4.5 CONCLUDING REMARKS

In this chapter we initiate the study of the problem of scheduling multiprocessor tasks with the throughput objective, that is, scheduling to maximize the number of early tasks in the schedule. Although multiprocessor task scheduling problems have been studied extensively, the throughput objective is new. We presented the first results in this direction. For both dedicated and parallel models, the complexity of the problem was established, and several approximation algorithms have been proposed and analyzed. However, many interesting questions remain. As we pointed out, in the parallel model there is a PTAS for the common due date problem. Is there a PTAS for the general problem? Is it APX-Hard? What happens if we add release dates or precedence constraints? We have only considered the case when tasks have unit processing times, and all problems with non-identical (arbitrary) processing times are open. Finally, the most interesting area is the design of online algorithms for the problem. We believe that our techniques and ideas can be very useful here.

APPENDIX A: CLASSIFICATION SCHEME

Because of a huge variety of machine scheduling problems, Graham, Lawler, Lenstra and Rinnooy Kan [GLLK79] proposed a classification scheme to make them easy to refer to. Later this scheme was also extended by Lawler, Lenstra, Rinnooy Kan and Shmoys [LLKS93] and by Drozdowski [Dro96]. Here, due to space limitations, we give only a short classification scheme. However, it suffices our purposes and we hope that the reader finds here all information needed.

Machine Scheduling. In general, the machine scheduling problems that we consider can be described as follows. There are m machines and n jobs. A *schedule* specifies, for each machine/processor i ($i = 1, \ldots, m$) and each job $j = 1, \ldots, n$, one or more time intervals throughout which processing is performed on j by i. A schedule is *feasible* if there is no overlapping of time intervals corresponding to the same job (so that a job cannot be processed by two machines at once), or time intervals corresponding to the same machine (so that a machine cannot process two tasks at the same time), and also if it satisfies various requirements relating to the specific problem type.

The problem type is specified by the machine environment, the job characteristics and optimality criterion (objective function). Accordingly, classification takes place by using of a three-field notation $\alpha|\beta|\gamma$.

The field $\alpha \in \{1, P, Q, R, O, F, J, D\}$ specifies the machine environment. If $\alpha \in \{1, P, Q, R\}$, we have a *single-stage* system where each job j consists of a single operation that can be processed on any machine. If $\alpha \equiv 1$, there is *single machine*. In the case of *identical parallel machines*, i.e., $\alpha \equiv P$, a job j has the same processing time p_j on each of the machines, whereas in the case of *uniform parallel machines* ($\alpha \equiv Q$), the processing time of each job j on machine i is p_j/s_i where s_i is the speed of machine i, and in the case of *unrelated parallel machines* ($\alpha \equiv R$) the processing time of each job j on machine i is equal to p_{ij}. If $\alpha \in \{O, F, J\}$, we have a *multi-stage* system where the processing of each job j is split into several *operations*. In an *open shop*, indicated by $\alpha \equiv O$, and in a *flow shop*, indicated by $\alpha \equiv F$, each job j has exactly m operations and its i-th operation has to be processed on the i-th machine during p_{ij} time units. The difference is that in an open shop the order in which the operations of a job to be executed is immaterial, whereas in a flow shop the operations execution order of a job is fixed, and is the

same for all jobs. If $\alpha \equiv J$, we have a job shop where each job j consists of a chain of operations, and chains of different jobs may be distinct. Sometimes, one uses D in α. In this case we have a dag shop in which the operation precedence constraints of a job are given as a *dag* - directed acyclic graph. In general, the number of machines is specified as a part of the problem instance. However, if the symbols P, Q, R, O, F, J, D are immediately followed by an integer or m, then the number of machines is specified as a part of the problem type and equal to this integer or m, respectively.

The field β contains the job characteristics. In this field, there may occur the entries *pmtn*, r_j, d_j, $p_j = 1/p_{ij} = 1$, and $op \leq \mu$. Accordingly, it indicates that preemption is allowed (the processing of any operation/job may be interrupted and resumed at a later time on the same or on a different machine), that jobs have release dates (the availability of each job j is restricted by its integer r_j that defines when it becomes available for processing), that jobs have due dates (each job j has due date d_j), that all jobs/operations have unit processing times, and that there are at most a constant number of operations per job (this is only for multi-stage scheduling problems). If field β includes non of these entries, then the default assumption applies. This means that *preemptions* of jobs are not allowed, that there are no release dates and due dates, and that the processing requirements are arbitrary positive integers.

To denote online problems, one can put *online* into filed β to denote online scheduling *over list*, and both *online* and r_j to indicate online scheduling *over time*. In the model of *scheduling over list*, the scheduler is confronted with the jobs one-by-one as they appear on a list. The existence of a job is not known until all its predecessors in the list have already been scheduled. In the model of *scheduling over time*, all jobs arrive at their release dates. The jobs are scheduled with passage of time and, at any point of time, the scheduler only has knowledge of those jobs that have already arrived.

Lastly, the third field γ refers to the optimality criterion (objective function). We are mainly interested in *makespan* $C_{\max} = \max C_j$, *average (weighted) completion time* $\sum(w_j)C_j$, and *throughput* $\sum \bar{U}_j$, where w_j is the weight of job/task j, C_j is the completion time of job/task j, $U_j = 1$ if $C_j \leq d_j$, and $\bar{U}_j = 0$ otherwise.

To illustrate the three-field descriptor, we present four examples: $1|r_j|\sum w_j C_j$ is the problem of scheduling jobs with release dates on a single machine to minimize the average weighted completion time, $J3|p_{ij} = 1|C_{\max}$ is the problem of scheduling unit operations in a three-machine job shop to minimize the makespan, $R|pmtn|\sum C_j$ is the problem of preemptive scheduling on unrelated machines, $P|on-line, r_j|C_{\max}$ is the problem of on-line scheduling over time on identical parallel machines.

Multiprocessor Task Scheduling. In order to model scheduling on multiprocessor architectures one must assume that a job can require more than one machine at a time. However, this would conflict to the "classical" job-machine scheduling assumptions. To make things simply, but remain correct, it is widely accepted to speak about scheduling a set $T = \{1,2,\ldots,n\}$ of *n multiprocessor tasks* on a set $M = \{1,2,\ldots,m\}$ of *m processors* extending the above $\alpha|\beta|\gamma$ notation.

It is assumed that each processor can work on at most one task at a time and a task can (or may need to be) processed simultaneously by several processors. In the *dedicated* variant of this model, denoted by $\alpha \equiv P$ and fix_j in β field, each task $j \in T$ requires the simultaneous use of a prespecified set $\mathrm{fix}_j \subseteq M$ of processors. In the *parallel* variant, denoted by $\alpha \equiv P$ and size_j in β field, the multiprocessor architecture is disregarded and for each task $j \in T$ there is given a prespecified number $\mathrm{size}_j \in M$ which indicates that the task can be processed by any subset of processors of the cardinality equal to this number. In the *general* model, $\alpha \equiv P$ and set_j in β field, each task can have a number of *alternative modes*, where each processing mode is specified by a subset of processors and the execution time of the task on that particular processor set.

We can give the following examples: $Pm|\,\mathrm{fix}_j|\sum w_jC_j$ denotes the problem of scheduling dedicated tasks on a fixed number of processors to minimize the the total weighted completion time, $P|\,\mathrm{set}_j, r_j|C_{\max}$ denotes the problem of scheduling general multiprocessor tasks with release dates to minimize the makespan, and $P|on-line, \mathrm{size}_j, p_j = 1|C_{\max}$ denotes the problem of on-line over list scheduling of unit parallel tasks. Notice that in some works the words "scheduling on dedicated/parallel processors" are used, e.g. in the book by Brucker [Bru98] and in Krämer's Ph.D. thesis [Krä95]. However, we will avoid it here.

APPENDIX B: ROUNDING PROCEDURE

For a detailed treatment of linear programming we refer the reader to two nice books [Sai95, BT97]. Here, we just briefly sketch the rounding technique used in Chapter 1. The main ideas of the technique can be found in [JSOS99].

Linear System. Let K, M be constant, and let $N \gg K, M$. Given $M \times K$ matrices A_j ($j = 1, 2, \ldots, N$) and a vector $b = (b_1, b_2, \ldots, b_M)^T$, we consider the following linear system (LS):

$$\textbf{(1)} \quad \sum_j A_j x_j = b,$$

$$\textbf{(2)} \quad \sum_i x_{ij} = 1, \text{ for all } j,$$

$$\textbf{(3)} \quad x_{ij} \geq 0, \text{ for all } i \text{ and } j,$$

where x_{ij} ($i = 1, 2, \ldots, K$) is the ith part of $x_j = (x_{1j}, x_{2j}, \ldots, x_{Kj})^T$ ($j = 1, 2, \ldots, N$).

In the following we show how one can modify a solution $x = (x_j)$ to a new solution $x' = (x'_j)$.

Removing of $x_{kj} \in \{0, 1\}$. Consider a solution $x = (x_j)$. We update LS in two cases: (I) there is x_j with $x_{kj} = 1$ (here, $x_{ij} = 0$ for all $i \neq k$); (II) there is x_j with $x_{kj} = 0$ (here, $0 \leq x_{ij} < 1$ for all $i \neq k$), as follows:

(I) remove x_j from x, set b equal to $b - A_j x_j$ in **(1)**, remove the jth line in **(2)**, remove all the jth lines in **(3)**;

(II) remove this x_{kj} from x_j, remove kth column from A_j in **(1)**, remove the kth sum component in the jth line of **(2)**, remove the k, jth line in **(3)**.

Less formally, we always eliminate 0s and 1s from x first, and then modify the LS respectively.

Rounding of x_{kj} to 1 or 0. Let $e_K = (1,\ldots,1)$ be the K vector of all ones, $b' = \begin{pmatrix} b \\ \mathbf{1} \end{pmatrix}$ be the $M+N$ vector which extends b by ones, and A be the $(M+N)\times(K\cdot N)$ matrix as

$$A = \begin{pmatrix} A_1 & A_2 & \ldots & A_N \\ e_K & & & \\ & e_K & & \\ \ldots & \ldots & \ldots & \ldots \\ & & & e_K \end{pmatrix}.$$

Then, we can write the LS as

$$Ax \;=\; b' \quad x \geq 0.$$

Assume that after removing of $x_{kj} \in \{0,1\}$ there are no $x_{ij} \in \{0,1\}$ in a solution $x = (x_j)$ of the LS. We select the columns in A corresponding to $M+1$ variables x_j. (If there are less than $M+1$ variables x_j we are done.) Then, the induced matrix A' is just a singular matrix of constant size. (Since there are no $x_{ij} \in \{0,1\}$, due to **(2)** of the LS, for each x_j there are at least two $0 < x_{ij} < 1$. Thus, there are $M+(M+1)$ rows and between $2(M+1)$ and $K(M+1)$ columns in A'.) Hence, one can find a non-zero vector y in the null space of this matrix, i.e., $A'y = 0$.

Let $\delta \in \mathbb{R}$ and $x' = x + \delta y$. (If the dimension of y is smaller than the dimension of x, we augment it by adding an appropriate number of zero entries.) Then,

$$Ax' \;=\; Ax \;+\; \delta Ay \;=\; b' \;+\; \delta A'y \;=\; b'.$$

Since all $0 < x_{ij} < 1$, there exists δ (if δ tends to 0) such that all $0 < x_{ij} + \delta y_{ij} < 1$. Thus, one can increase or decrease the value of δ until at least one of variables x'_{kj} gets either 0 or 1.

This process rounds the value of at least one variable. Furthermore, since A' has constant size one can find y and δ in constant time (simple linear algebra).

Rounding of x. We repeat rounding and removing procedures for variables x_{kj} until there are at most M vectors x_j left. (Here A' can become non-singular.) At the end of this iterative process, all, but these M, vectors x_j have all $x_{ij} \in \{0,1\}$. The total number of iterations is at most $N-M$ and each iteration takes a constant number of steps. Thus, we can prove the following:

Lemma 4.5.1. *A solution $x = (x_j)$ for the LS can be transformed in $O(n)$ time to another solution $x' = (x'_j)$ in which there are at most M vectors x_j with $x'_{ij} \in (0,1)$, and all other x_j with $x_{ij} \in \{0,1\}$.*

In fact, it is enough to have $2(M+1)$ variables $x_{ij} \in (0,1)$ for building A'. Then, A' is singular and we can repeat rounding. Hence, we can also conclude the following:

Lemma 4.5.2. *A solution $x = (x_j)$ for the LS can be transformed in $O(n)$ time to another solution $x' = (x'_j)$ in which at most $2(M+1)$ variables $x_{ij} \in (0,1)$ and all other $x_{ij} \in \{0,1\}$.*

APPENDIX C: GRAPHS

Due to space limitations, here we give only some basic definitions and notations for graph. However, we hope that the reader will find here all information needed. For more details, we refer to a number of excellent books [CL86, Bol98, GY98, Wes01].

Simple Graphs. An undirected graph $G = (V, G)$ consists of a finite non-empty set V of *vertices* and a finite set E of *edges*. (Later we can use the notation $V(G)$ and $E(G)$ for the vertex set and edge set of G.) With every edge $e \in E$, an unordered pair $\{u, v\}$ of vertices is associated and we say that e is *incident* to u and v. We assume that the two vertices of an edge are distinct, i.e., G is *simple*. Consequently, we write $e = \{u, v\}$. Two vertices u and v that are joined by an edge are called *adjacent* or *neighbors*.

For a vertex v the *neighborhood* of v, $N(v)$, is defined as

$$N(v) \;=\; \{u \in V \mid u \text{ is adjacent to } v \text{ in } G\}.$$

The *degree* of a vertex v, $deg(v)$, is the number of edges which are incident with v, or equivalently,

$$\deg(v) \;=\; |N(v)|.$$

We use $\Delta(G)$ to denote the maximum degree of graph G.

We say that graph H is a *subgraph* of a graph G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. If $U \subseteq V(G)$, we use $G[U]$ to denote the *induced subgraph* of G whose vertex set is U and whose edge set is the subset of $E(G)$ consisting of those edges with both ends in U. If $S \subseteq E(G)$, we use $G[S]$ to denote the edge induced subgraph of G whose edge set is S and whose vertex set is the subset of $V(G)$ consisting of those vertices incident with any edge in S.

The complement, $G^{\sim}$ of a graph G is a graph with the same vertices as G and with the property that two vertices in $G^{\sim}$ are adjacent if and only if they are not adjacent in G.

Simple Structures. A *walk* is an alternating sequence $v_0, e_1, v_1, \ldots, e_k, v_k$ of vertices and edges, with each edge being incident to the vertices immediately preceding and succeeding it in the sequence, i.e. $e_i = v_{i-1}v_i$ for all i. A *trail* is a walk with no repeated edges. A *path* is a walk with no repeated vertices. A walk is *closed* if the initial vertex is also the terminal vertex. A *cycle* is a closed trail with at least one edge and with no repeated vertices except that the initial vertex is the terminal vertex. The *length* of a walk is the number of edges in the sequence defining the walk. Thus, the length of a path or cycle is also the number of edges in the path or cycle.

A graph G is *connected* if it has a u,v-path for each pair $v,u \in V(G)$. The connected parts of a disconnected graph are called the *connected components* of that graph. (A connected graph is therefore a graph with exactly one connected component.) If u and v are vertices, the distance from u to v, written $dist_G(u,v)$, is the minimum length of any path from u to v. (In an undirected graph, this is obviously a metric.)

A graph is *acyclic* if it has no cycles. An acyclic graph is also called a *forest*. A *tree* is a connected, acyclic graph. Thus every connected component of a forest is a tree. A *spanning tree* of a graph G is a subgraph T of G which is a tree and which satisfies $|V(T)| = V|(G)|$.

A graph is called *complete* if every two of its vertices are joined by an edge. A complete graph of order n has $n(n-1)/2$ edges and is denoted by K_n.

A graph G is *bipartite* if $V(G) = X \cup Y$ and $X \cap Y = \emptyset$ such that every edge in G joins a vertex in X and a vertex in Y. We write $G = (X,Y)$ sometimes. G is a *complete bipartite graph* if every vertex in X is joined to every vertex in Y. We use the notation $K_{m,n}$ for a complete bipartite graph with m vertices in X and n vertices in Y. $K_{1,n}$ is also called a *star*.

A graph G is called *planar* if it can be drawn on a plane in such a way that there are no "edge crossings", i.e. edges intersect only at their common vertices.

The *intersection* graph G of a system $S = (Sx, Sy, Sz, ...)$ has vertices $x,y,z,\ldots$. Two distinct vertices x,y are joined by an edge whenever Sx and Sy have nonempty intersection. S is called the *representation* of G.

The intersection graph G of a set of intervals $I = \{I_1, \ldots, I_n\}$, where each $I_j = [a_j, b_j] \subset \mathbb{R}$, is called an *interval graph*.

The intersection graph G of a set of disks $D = \{D_1, \ldots, D_n\}$ in the plane, where each D_j is defined by its center in $(a_j, b_j) \in \mathbb{R}^2$ and its diameter $d_j \in \mathbb{R}$, is called a *disk graph*. Then, D is called the *disk representation* of G. The value $\sigma(D) = \max d_j / \min d_j$ is called the *diameter ratio* of D. Accordingly, if $\sigma(D) = 1$, i.e. all disks of D have unit diameter, then G is called a *unit disk graph*, and if $1 <$

$\sigma(D) \leq \sigma$ for some constant σ, then G is called a σ-*disk graph*.

Let X be a finite set. A *k-tuple* of X is a set having k (or less) elements of X. The intersection graph G of a set of k-tuples of X is called a *k-tuple graph*. In this case, several vertices can correspond to a single k-tuple.

The *square product* of G and H, is the graph whose vertex-set is the cartesian product of $V(G)$ and $V(H)$, and where the pair (ax, by) is an edge if and only if either $a = b$ and $\{x,y\}$ is an edge of H, or $x = y$ and $\{a,b\}$ is an edge of G. The square product of two paths is frequently called the *grid* graph.

A *co-graph* is a graph which can be generated by disjoint union and join operations on graphs, starting with a single-vertex graph. The union $\cup(G_1,G_2)$ and the join $+(G_1,G_2)$ of two graphs $G_i = (V_i,E_i)$, $i = 1,2$ is defined by

$$\cup(G_1,G_2) \;=\; (V_1 \cup V_2, E_1 \cup E_2),$$

and

$$+(G_1,G_2) \;=\; (V_1 \cup V_2, E_1 \cup E_2 \cup \{\{x_1,x_2\} | x_i \in V_i\}).$$

Clique, Independent Set, Coloring and Labeling. For a graph G, a subset $V' \subseteq G(V)$ is a *clique* if every two vertices in V' are joined by an edge in $G(E)$. A *maximum clique* is, naturally, a clique whose number of vertices is at least as large as that for any other clique in the graph, and its size, $\omega(G)$, is called the *clique number* of G.

For a graph G, a subset $V' \subseteq G(V)$ is an *independent set* if no its vertices are adjacent. Similarly, a *maximum independent set* is an independent set whose number of vertices is at least as large as that for any other clique in the graph, and its size, $\alpha(G)$, is called the *independence number* of G.

A (vertex) k-coloring of a graph $G = (V,E)$ is a function $c : V(G) \to \{1,\ldots,k\}$ such that $c(u) \neq c(v)$ whenever u is adjacent to v. If a k-coloring of G exists, then G is called k-colorable. The *chromatic number* of G is defined as

$$\chi(G) \;:=\; \min\{k \mid G \text{ is } k-\text{colorable}\}.$$

Obviously, we have

$$\chi(G) \geq \omega(G) \;\text{ and }\; \chi(G) \leq \Delta(G) \;+\; 1.$$

A graph G is *perfect* if, for any induced subgraph H of G, the chromatic number of H is equal to the size of a maximum clique of H, that is, $\chi(H) = \omega(H)$.

Let $p_1, ..., p_k$ be a non-increasing sequence of positive integers, called *distance constraints*. An $L_{(p_1,...,p_k)}$*-labeling*, or a *distance-constrained* labeling, of a graph G is a function $c : V(G) \to \{1,...,L\}$ such that $|c(u) - c(v)| \geq p_i$ whenever u and v are at the graph distance i, for $i = 1, \ldots, k$. If an $L_{(p_1,...,p_k)}$-labeling of G exists, then G is called $L_{(p_1,...,p_k)}$-labeled. The $(p_1, \ldots, p_k)$*-labeling number* of G is defined as

$$\chi_{(p_1,...,p_k)}(G) := \min\{L \mid G \text{ is } L_{(p_1,\ldots,p_k)} - \text{labeled}\}.$$

APPENDIX D: COMPLEXITY AND NPO PROBLEMS

Here we give an overview of complexity theory for the algorithm designer. This only includes some main definitions. For more details we refer to the following excellent books [GJ79, Pap94, AGG+99].

Complexity Classes. Let $\{0,1,\}^*$ be the set of all possible strings over alphabet $\{0,1\}$. Denote by $|x|$ the length of a string x. A language $L \subseteq \{0,1\}^*$ is any collection of strings over $\{0,1\}$. The corresponding *language recognition* problem is to decide whether a given string $x \in \{0,1\}^*$ belongs to L. An algorithm solves a language recognition problem for a specific language L by *accepting* (output "yes") any input string contained in L, and *rejecting* (output "no") any input string not contained in L.

A complexity class is a collection of languages all of whose recognition problems can be solved under prescribed bounds on the the computational resources. We are primarily interested in various of efficient algorithms, where efficient is defined as being *polynomial time*. Recall that an algorithm has polynomial running time if it halts within $n^{O(1)}$ on any input of length n.

The class P consists of all languages L that have a polynomial time algorithm ALG such that for any input string $x \in \{0,1\}^*$,

- $x \in L \Longrightarrow \mathrm{ALG}(x)$ accepts, and
- $x \notin L \Longrightarrow \mathrm{ALG}(x)$ rejects.

The class NP consists of all languages L that have a polynomial time algorithm ALG such that for any input string $x \in \{0,1\}^*$,

- $x \in L \Longrightarrow$ there is a string $y \in \{0,1\}^*$, $\mathrm{ALG}(x,y)$ accepts, where length $|y|$ is polynomial in $|x|$.
- $x \notin L \Longrightarrow$ for any string $y \in \{0,1\}^*$, $\mathrm{ALG}(x,y)$ rejects.

Obviously, P $\subseteq$ NP, but it is not known whether P $=$ NP.

For any complexity class $\mathcal{C}$, we define the complexity class co-$\mathcal{C}$ as the set of languages whose complement is in class $\mathcal{C}$. That is

$$\text{co-}\mathcal{C} = \{L | \bar{L} \in \mathcal{C}\}.$$

It is obvious that P = co-P and P $\subseteq$ NP $\cap$ co-NP.

NP-**completeness.** A *polynomial reduction* from a language $L' \subseteq \{1,0\}^*$ to a language $L \subseteq \{1,0\}^*$ is function $f : \{1,0\}^* \to \{1,0\}^*$ such that:

- There is a polynomial time algorithm that computes f.
- For all $x \in \{1,0\}^*$, $x \in L'$ if and only if $f(x) \in L$.

Clearly, if there is a polynomial reduction from L' to L, then $L \in$ P implies that $L' \in$ P.

A language L is NP-*hard* if for every language $L' \in$ NP, there is a polynomial reduction from L to L'. A language L is NP-*complete* if $L \in$ NP and L is NP-hard.

Randomized Complexity Classes. The class RP (for Randomized Polynomial Time) consists of all languages $L \subseteq \{0,1\}^*$ that have a randomized algorithm ALG running in worst-case polynomial time such that for any $x \in \{0,1\}^*$:

- $x \in L \Longrightarrow \mathbf{Pr}[\text{ALG}(x) \text{ accepts}] \geq \frac{1}{2}$.
- $x \notin L \Longrightarrow \mathbf{Pr}[\text{ALG}(x) \text{ accepts}] = 0$.

Clearly,

$$\text{P} \subseteq \text{RP} \subseteq \text{NP}.$$

A language belonging to both RP and co-*RP* can be solved by a randomized algorithm with *zero-sided error*, i.e., a *Las Vegas* algorithm. The class ZPP (for Zero-error Probabilistic Polynomial time) is the class of all languages that have Las Vegas algorithms running in expected polynomial time. Clearly,

$$\text{ZPP} = \text{RP} \cup \text{co-RP}.$$

NP-**hard Decision Problems.** Informally, a *decision problem* is one whose answer is either "yes" or "no", and it can be treated as a language recognition problem.

Abstractly, a decision problem Π consists simply of a set D_Π of *instances* and a subset $Y_\Pi \subseteq D_\Pi$ of *yes-instances*. An *encoding scheme* for problem Π provides a way of describing each instance I in D_Π by an appropriate string in $\{0,1\}^*$. Then, the language assosited with Π is defined as

$$L[\Pi] := \{x \in 0,1^* | x \text{ is the encoding under } e \text{ of an instance } I \in Y_\Pi\}.$$

We say that a decision problem Π is NP-hard (complete) if $L[\Pi]$ is NP-hard (complete).

There are two common ways for encoding numbers (integers): *unary* and *binary*. Clearly, the hardness of a decision problem can change when one switches from binary to unary encoding.

We say that a decision problem Π is NP-hard (complete) in the *strong sense* or Π is *strongly* NP-hard (complete) if $L[\Pi]$ is NP-hard (complete) under an unary encoding scheme.

NPO Problems. An NP-*optimization problem* (NPO), Π, consists of:

- A set of *input instances*, $\mathcal{I}$, recognized in polynomial time. The *size* of instance $I \in \mathcal{I}$, denoted by $|I|$, is defined as the number of bits needed to write I under the assumption that all numbers occurring in I are written in binary.
- Each instance $I \in \mathcal{I}$ has a set of feasible solutions $F(I)$. We require that $F(I) \neq \emptyset$, and that every solution $S \in F(I)$ is of length polynomial in $|I|$. Furthermore, there is polynomial time algorithm that, given a pair (I,S), decides whether $S \in F(I)$.
- There is a polynomial time computable *objective function*, obj, that assigns a nonnegative rational number to each pair (I,S), where $I \in \mathcal{I}$ and $S \in F(I)$.
- Finally, Π is specified to be either a *minimization problem* or a *maximization problem*.

An *optimal solution* for an instance of a minimization (maximization) NPO problem is a feasible solution that achieves the smallest (largest) objective function value. $OPT(I)$ will denote the objective value of an optimal solution for instance I.

An algorithm ALG is said to be *optimal* for an NPO problem Π if, on each instance I, ALG computes an *optimal solution*, i.e. a feasible solution $S \in F(I)$ such that $obj(I,S) = OPT(I)$, and the running time of ALG is polynomial in I.

The decision version of an NPO problem Π consists of pairs (I,B), where I is an instance of I and B is a rational number. If Π is a minimization problem (maximization problem), then the answer to the decision problem is "yes" iff there is a feasible solution to I of the objective function value $\leq B$ ($\geq B$). If so, we will say that (I,B) is a yes-instance.

An NPO problem Π is said to be (strongly) NP-hard if its decision version is (strongly) NP-complete. Assuming P $\neq$ NP, no (strongly) NP-hard NPO problem has an optimal algorithm.

Approximation Algorithms. An approximation algorithm produces a feasibel "near-optimal" solution, and it is time efficient. The formal definition differs for minimization and maximization problems. Let Π be a minimization problem. An algorithm ALG is said to be a ρ-approximation algorithm for Π, if on every instance I of Π, ALG computes a feasible solution $S \in F(I)$ such that

$$obj(I,S) \ \leq \ \rho \cdot OPT(I),$$

and the running time of ALG is polynomial in $|I|$. For a maximization problem Π, a ρ-approximation algorithm satisfies

$$obj(I,S) \ \geq \ \frac{1}{\rho} \cdot OPT(I).$$

The asymmetry in the definition is due to ensure that $\rho \geq 1$. The value of $\rho \geq 1$ is called the *approximation ratio* or *performance ratio* or *worst-case ratio* of ALG and in general can be a function of $|I|$.

A family of approximation algorithms, $\{A_\varepsilon\}_{\varepsilon>0}$, for an NPO problem Π, is called a *polynomial time approximation scheme* or a PTAS, if algorithm A_ε is a $(1+\varepsilon)$-approximation algorithm and its running time is polynomial in the size of the instance for a fixed ε. If the running time of each A_ε is polynomial in the size of the instance and in $1/\varepsilon$, then $\{A_\varepsilon\}_{\varepsilon>0}$ is called a *fully polynomial time approximation scheme* or a FPTAS.

Assuming P $\neq$ NP, a PTAS is the best result we can obtain for a strongly NP-hard problem, and a FPTAS is the best result we can obtain for an NP-hard problem.

AP-Reduction. The concept of approximation preserving reductions primarily provides a method for proving that an NPO problem does not admit any PTAS, unless P = NP.

For a constant $\alpha \geq 0$ and two NPO problems A and B, we say that A is α-AP-reducible to B if two polynomial-time computable functions f and g exist such that the following holds:

- For any instance I of A, $f(I)$ is an instance of B.
- For any instance I of A, and any feasible solution S' for $f(I)$, $g(I,S')$ is a feasible solution for I.
- For any instance I of A and any $r \geq 1$, if S' is is an r-approximate solution for $f(I)$, then $g(I,S')$ is an $(1+(r-1)\alpha+o(1))$-approximate solution for I, where the o notation is with respect to $|I|$.

We say that A is AP-reducible to B if a constant $\alpha \geq 0$ exists such that A is α-AP-reducible to B. Clearly, if A is AP-reducible to B, then an ρ-approximate solution for B is mapped to an $h(\rho)$ approximate solution for A, where $h(\rho) \to 1$ as $\rho \to 1$.

The class APX consists of all NPO problems that have a constant factor approximation. Then, AP-reductions preserve membership in APX. Furthermore, if A is AP-reducible to B and there is a PTAS for B, there is a PTAS for A as well.

An NPO problem Π is APX-hard if every APX problem is AP-reducible to Π. An NPO problem Π is APX-*complete* if $\Pi \in$ APX and Π is APX-hard.

Assuming P $\neq$ NP, no APX-hard (complete) problem has a PTAS.

A Little Bit of History. In [Gra66] a simple algorithm for scheduling jobs on a single machine was presented: Suppose we are given a single machine and a list of n jobs in some order. Whenever a machine becomes available, it starts processing the next job on the list. Graham made a complete worst-case analysis of this algorithm and showed that the maximum job completion time (or *makespan*) of the schedule is at most twice the makespan of an optimal schedule. It was perhaps the first polynomial time approximation algorithm for an NP-hard optimization problem, and at the same time, the first competitive analysis of an on-line algorithm.

Only several years later, immediately after the concepts of NP-completeness and approximation algorithms were formalized [Coo71, GGU72]. However, a paper [Joh74] of Johnson may be regarded as the real starting point in the field. The terms “approximation scheme”, “PTAS”,“FPTAS” are due to a seminal paper [GJ78]. The first inapproximability results were also derived about this time, see e.g. [SG76, LK78].

Much of the work has been also devoted to classifying the optimization problems with respect to their polynomial time approximability. The notion of *strong* NP-*completeness* was introduced in [GJ78]. It was also shown that *strong* NP-*hard* problems do not have FPTASs unless P = NP [GJ79].. A *strongly* NP-hard problem is a problem that remains NP-hard even if the numbers in its input are unary encoded [GJ79].

In [PY91] the class MAX-SNP was introduced by a logical characterization and the notion of completeness for this class by using the so-called L-reduction. The idea behind this concept was that every MAX-SNP-complete optimization problem does not admit any PTAS iff MAX-3SAT does not admit any PTAS. A number of optimization problem were proven to be MAX-SNP-complete. In a remarkable line of work that culminated in [ALM$^+$92], it was shown that MAX-3SAT has no PTAS, unless P = NP.

Later, based on known results about the approximability thresholds of various problems, researches have classified problems into a number of classes [AL96]. One of these classes is APX. It was established in [KMSV94, CKST95, CT00] that MAX-3SAT is APX-complete under AP-reduction and under subtler notion of reductions. Many problems have been shown to be either APX-complete or APX-hard, and thus do not have a PTAS, unless P = NP.

Generalizing NP to allow for *randomized* algorithms has led to a number of new complexity classes, e.g. ZPP (Zero-error Probabilistic Polynomial) and PCP (Probabilistically Checkable Proofs). It was shown that the so-called PCP-theorem (NP = PCP$(\log n, 1)$) implies that the problem of finding a maximum clique in an n-vertex graph cannot be approximated within a factor of $n^{1-\varepsilon}$, neither for some $\varepsilon > 0$, unless P = NP; nor for any $\varepsilon > 0$, unless NP = ZPP [Aro94, AL96, AGG$^+$99, MPS98].

BIBLIOGRAPHY

[ABC+99] F. Afrati, E. Bampis, C. Chekuri, D. Karger, C. Kenyon, S. Khanna, I. Millis, M. Queyranne, M. Skutella, C. Stein, and M. Sviridenko, *Approximation schemes for minimizing average weighted completion time with release dates*, Proceedings 40th IEEE Symposium on Foundations of Computer Science, 1999, pp. 32–43.

[ABF+00] F. Afrati, E. Bampis, A. V. Fishkin, K. Jansen, and C. Kenyon, *Scheduling to minimize the average completion time of dedicated tasks*, Proceedings 20th Conference on Foundations of Software Technology and Theoretical Computer Science, LNCS 1974, Springer Verlag, 2000, pp. 454–464.

[ABKM97] A. K. Amoura, E. Bampis, C. Kenyon, and Y. Manoussakis, *Scheduling independent multiprocessor tasks*, Proceedings 5th European Symposium on Algorithms, LNCS 1284, Springer Verlag, 1997, pp. 1–12.

[ABKM00] F. Afrati, E. Bampis, C. Kenyon, and I. Milis, *A PTAS for the average weighted completion time problem on unrelated machines*, Journal of Scheduling, Special Issue on Approximation Algorithms **3** (2000), 323–332.

[AGG+99] G. Ausiello, P. Grescenzi, G. Gambosi, V. Kann, M. Marchetti-Spaccamela, and M. Protasi, *Complexity and approximation: Combinatorial optimization problems and their approximability properties*, Springer Verlag, 1999.

[AL96] S. Arora and C. Lund, *Approximation algorithms for NP-hard problems*, ch. Hardness of approximation (D. S. Hochbaum ed.), pp. 1–45, PWS Publishing Company, Boston, 1996.

[AL97] E. Aarts and J. K. Lenstra (eds.), *Local search in Combinatorial Optimization*, Willey-Interscience series in discrete mathematics and optimization, John wiley and Sons, 1997.

[ALM+92] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, *Proof verification and hardness of approximation problems*, Proceedings 33rd Annual IEEE Symposium on Foundations of Computer Science, 1992, pp. 14–23.

[AM01] F. Afrati and I. Milis, *Designing PTASs for MIN-SUM scheduling problems.*, Proceedings 13th International Symposium on Fundamentals of Computation Theory, LNCS 2138, Springer Verlag, 2001, pp. 432–444.

[APE97] M. Ammar, G. Polyzos, and S. Tripathi (Eds.), *Special issue on network support for multipoint communication*, IEEE Journal Selected Areas in Communications **15** (1997).

[Aro94] S. Arora, *Probabilistic checking of proofs and the hardness of approximation problems*, Ph.D. thesis, U.C. Berkley, 1994.

[BCF+02] E. Bampis, M. Caramia, J. Fiala, A. V. Fishkin, and A. Iovanella, *On scheduling of independent dedicated multiprocessor tasks*, Proceedings 13th International Symposium on Algorithms and Computation (Vancouver, BC, Canada), LNCS 2518, Springer Verlag, 2002, pp. 391–402.

[BCJS74] J. L. Bruno, E. G. Coffman, Jr., and R. Sethi, *Scheduling independent tasks to reduce mean finishing time*, Communications of the ACM **17** (1974), 382–387.

[BEY98] A. Borodin and R. El-Yaniv, *Online computation and competitive analysis*, Cambridge University Press, 1998.

[BGS98] M. Bellare, O. Goldreich, and M. Sudan, *Free bits, PCPs, and nonapproximability — towards tight results*, SIAM Journal on Computing **27** (1998), 804–915.

[BK96] P. Brucker and A. Kräsmer, *Polynomial algorithms for resource constrained and multiprocessor task scheduling problems*, European Journal of Operational Research **90** (1996), 214–226.

[BK98] H. Breu and D. G. Kirkpatrick, *Unit disc graph recognition is NP-hard*, Computational Geometry: Theory and Applications **9** (1998), 3–24.

[BK01] E. Bampis and A. Kononov, *On the approximability of scheduling multiprocessor tasks with time dependent processing and processor requirements*, Proceedings 15th International Parallel and Distributed Processing Symposium (San Francisco), 2001.

[BKTvL00] H. L. Bodlaender, T. Kloks, R. B. Tan, and J. van Leeuwen, *λ-coloring of graphs*, Proceedings 17th International Symposium on Theoretical Aspects of Computer Science, LNCS 1770, Springer Verlag, 2000, pp. 395–406.

[BM98] S. Bischof and E.W. Mayr, *On-line scheduling of parallel jobs with runtime restrictions*, Proceedings 9th Annual International Symposium on Algorithms and Computation, LNCS 1533, Springer Verlag, 1998, pp. 119–129.

[BNBH+98] A. Bar-Noy, M. Bellare, M. M. Halldórsson, H. Shachnai, and T. Tamir, *On chromatic sums and distributed resource allocation*, Information and Computation **140** (1998), 183–202.

[BNHK+99] A. Bar-Noy, M. M. Halldórsson, G. Kortsarz, R. Salman, and H. Shachnai, *Sum multicoloring of graphs*, Proceedings 7th European Symposium on Algorithms, LNCS 1643, Springer Verlag, 1999, pp. 390–401.

[Bol98] B. Bollobás, *Modern graph theory*, Graduate Texts in Mathematics 184, Springer Verlag, 1998.

[BR99] R. Balakrishnan and K. Ranganathan, *A textbook of graph theory*, Springer Verlag, 1999.

[BR02] E. Bampis and G. Rouskas, *The scheduling and wavelength assignment problem in optical wdm networks*, Journal of Lightwave Technology **20 (5)** (2002), 782–789.

[Bru98] P. Brucker, *Scheduling algorithms*, Springer Verlag, 1998.

[BT97] D. Bertsimas and J. N. Tsitsiklis, *Introduction to linear optimization*, Athena Scientific, Belmont, Massachusetts, 1997.

[CCJ90] B. N. Clark, C. J. Colbourn, and D. S. Johnson, *Unit disk graphs*, Discrete Mathematics **86** (1990), 165–177.

[CCLL95] P. Chrétienne, E. G. Coffman, J. K. Lenstra, and Z. Liu (eds.), *Scheduling theory and its applications*, John Wiley and Sons, 1995.

[CCPS98] W. J. Cook, W. H. Cunningham, W. R. Pulleyblank, and A. Schrijver, *Combinatorial optimization*, Wiley-Interscience series in discrete mathematics and optimization, John Wiley and Sons, Inc, 1998.

[CDI01a] M. Caramia, P. Dell'Olmo, and A. Iovanella, *Lower bound algorithms for multiprocessor task scheduling with ready times*, 2001, Personal communications.

[CDI01b] M. Caramia, P. Dell'Olmo, and A. Iovanella, *On-line algorithms for multiprocessor task scheduling with ready times*, 2001, Personal communications.

[CH01] J. Chen and J. Huang, *Semi-normal scheduling: Improvement on goemans' algorithm*, Proceedings 12th International Symposium on Algorithms and Computation, LNCS 2223, Springer Verlag, 2001, pp. 48–60.

[Che98] C. Chekuri, *Approximation algorithms for scheduling problems*, Ph.D. thesis, Department of Computer Science, Stanford University, 1998.

[CK96] G. J. Chang and D. Kuo, *The $L(2,1)$-labeling problem on graphs*, SIAM Journal of Discrete Mathematics **9** (1996), 309–316.

[CK00] C. Chekuri and S. Khanna, *A PTAS for the multiple knapsack problem*, Proceedings 9th Annual ACM-SIAM Symposium on Discrete Algorithms, 2000, pp. 213–222.

[CK01] C. Chekuri and S. Khanna, *A PTAS for minimizing weighted completion time on uniformly related machines*, Proceedings 28th International Colloquium on Automata, Languages and Programming, LNCS 2076, Springer Verlag, 2001, pp. 848–861.

[CKP00] A. Caprara, H. Kellerer, and U. Pferschy, *The multiple subset sum problem*, SIAM Journal on Optimization **11** (2000).

[CKST95] P. Crescenzi, V. Kann, R. Silvestri, and L. Trevisan, *Structure in approximation classes*, Proceedings 1st Computing and Combinatorics Conference, LNCS 959, Springer Verlag, 1995, pp. 539–548.

[CL86] G. Chartrand and L. Lesniak, *Graphs and digraphs*, Wadsworth and Brooks/Cole, 1986.

[CLL98] X. Cai, C.-Y. Lee, and C.-L. Li, *Minimizing the total completion time in two-processor task systems with prespecified processor allocation*, Naval Research Logistics **45** (1998), 231–242.

[CLT78] E. G. Coffman, J. Y-T. Leung, and D.W. Ting, *Bin packing: maximizing the number of pieces packed*, Acta Informatica **9** (1978), 263–271.

[CM99] J. Chen and A. Miranda, *A polynomial time approximation scheme for general multiprocessor job scheduling*, Proceedings 31st ACM Symposium on the Theory of Computing, 1999, pp. 418–427.

[CMI00] Clay Mathematics Institute, *Millennium Prize Problems*, URL: http://www.claymath.org/prizeproblems/index.htm, Announced 16:00, on Wednesday, May 24, 2000.

[CMNS97] C. Chekuri, R. Motwani, B. Natarajan, and C. Stein, *Approximation techniques for average completion time scheduling*, Proceedings of 8th Annual ACM-SIAM Symposium on discrete Algorithms, 1997, pp. 609–618.

[Coo71] S. A. Cook, *The complexity of theorem proving procedures*, Proceedings 30th ACM Symposium on the Theory of Computing, 1971, pp. 151–158.

[CPS+96] S. Chakrabarti, C. A. Phillips, A. S. Schulz, D. B. Shmoys, C. Stein, and J. Wein, *Improved scheduling algorithms for minsum criteria*, Proceedings 23rd International Colloquium on Automata, Languages and Programming, LNCS 1099, Springer Verlag, 1996, pp. 646–657.

[CPW98] B. Chen, C. N. Potts, and G. J. Woeginger, *Handbook of combinatorial optimization (D.-Z. Du and P. M. Paradalos eds.)*, ch. A review of machine scheduling: complexity, algorithms and approximability, pp. 21–169, Kluwer, 1998.

[CT00] P. Crescenzi and L. Trevisan, *On approximation scheme preserving reducibility and its applications*, Theory of Computer Systems **33** (2000), 1–16.

[DP95] S. Dauzère-Pérès, *Minimizing late jobs in the general one machine scheduling problem*, European Journal of Operational Research **81** (1995), 134–142.

[Dro96] M. Drozdowski, *Scheduling multiprocessor tasks - an overview*, European Journal on Operations Research (1996), 215–230.

[ECL79] Jr. E.G. Coffman and J.Y. Leung, *Combinatorial analysis of an efficient algorithms for processor and storage allocation*, SIAM Journal on Computing **8** (1979), 202–217.

[EF01] T. Erlebach and J. Fiala, *Independence and coloring problems on intersection graphs of disks*, manuscript, 2001.

[FFF01] J. Fiala, A. V. Fishkin, and F. Fomin, *Off-line and on-line distance constrained labeling of disk graphs*, Proceedings 9th Annual European Symposium on Algorithms (Arhus), LNCS 2161, 2001, pp. 464–475.

[Fia00] J. Fiala, *Locally injective homomorphisms*, Ph.D. thesis, Charles University, Prague, 2000.

[FJM01] A. V. Fishkin, K. Jansen, and M. Mastrolilli, *Grouping techniques for scheduling problems: Simpler and faster*, Proceedings 9th Annual European Symposium (Arhus), LNCS 2161, Springer Verlag, 2001, pp. 206–217.

[FJM02] A. V. Fishkin, K. Jansen, and M. Mastrolilli, *A PTAS for shop scheduling with release dates to minimize the average weighted completion time*, Tech. report, Kiel University, 2002.

[FJP01a] A. V. Fishkin, K. Jansen, and L. Porkolab, *On minimizing average weighted completion time: A PTAS for scheduling general multiprocessor tasks*, Proceedings 13th International Symposium on Fundamentals of Computation Theory (Riga), LNCS 2138, Springer Verlag, 2001, pp. 495–507.

[FJP01b] A. V. Fishkin, K. Jansen, and L. Porkolab, *On minimizing average weighted completion time of multiprocessor tasks with release dates*, Proceedings 28th International Colloquium on Automata, Languages and Programming (Crete), LNCS 2076, Springer Verlag, 2001, pp. 875–886.

[FK98] U. Feige and J. Kilian, *Zero-knowledge and the chromatic number*, Journal of Computer and System Science **57** (1998), 187–199.

[FK02] J. Fiala and J. Kratochvíl, *Partial covers of graphs*, Discussions Mathematics Graph Theory **22** (2002), 89–99.

[FKK99] J. Fiala, J. Kratochvíl, and T. Kloks, *Fixed-parameter tractability of λ-colorings*, Proceedings 25th International Workshop on Graph-Theoretic Concepts in Computer Science (Ascona), 1665, Springer Verlag, 1999, pp. 350–363.

[FKST93] A. Feldmann, M.-Y. Kao, J. Sgall, and S.H. Teng, *Optimal online scheduling of parallel tasks with dependencies*, Proceedings 25th ACM Symposium on the Theory of Computing, 1993, pp. 642–651.

[FST94] A. Feldmann, J. Sgall, and S-H. Teng, *Dynamic scheduling on parallel machines*, Theoretical Computer Science **130** (1994), 49–72.

[FW98] A. Fiat and G. J. Woeginger (eds.), *Online algorithms. The state of the art*, LNCS 1442, Springer Verlag, 1998.

[FZ02] A. Fishkin and G. Zhang, *On maximizing the throughput of multiprocessor tasks*, Proceedings 27th International Symposium on MFCS (Warsaw-Otwock, Poland), LNCS 2420, Springer Verlag, 2002, pp. 269–279.

[GGU72] M. R. Garey, R. L. Graham, and J. D. Ullman, *Worst case analysis of memory allocation algorithms*, Proceedings 4th ACM Symposium on Theory of Computing, 1972, pp. 143–150.

[GJ78] M. R. Garey and D. S. Johnson, *Strong NP-completeness results: Motivation, examples and applications*, Journal of the Association for Computing Machinery (1978), 499–508.

[GJ79] M. R. Garey and D. S. Johnson, *Computers and intractability: A guide to the theory of NP-completeness*, Freeman, San Francisco, CA, 1979.

[GJS76] M. R. Garey, D. S. Johnson, and R. Sethi, *The complexity of flowshop and jobshop scheduling*, Mathematics of Operation Research **1** (1976), 117–129.

[GL88] A. Gyárfás and J. Lehel, *On-line and first fit colorings of graphs*, Journal of Graph Theory **12** (1988), 217–227.

[GLLK79] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan, *Optimization and approximation in deterministic scheduling: A survey*, Annals of Discrete Mathematics (1979), 287–326.

[GLM+00] O. Gerstel, B. Li, A. McGuire, G. N. Rouskas, K. Sivalingam, and Z. Zhang (Eds.), *Special issue on protocols and architectures for next generations optical wdm networks*, IEEE Journal Selected Areas in Communications **18** (October 2000).

[GM96] J. P. Georges and D. W. Mauro, *On the size of graphs labeled with a condition at distance two*, Journal of Graph Theory **22** (1996), 47–57.

[Goe95] M. X. Goemans, *An approximation algorithm for scheduling on three dedicated processors*, Discrete Applied Mathematics **61** (1995), 49–59.

[GPSS97] L. A. Goldberg, M. Paterson, A. Srinivasan, and E. Sweedyk, *Better approximation guarantees for job-shop scheduling*, Proceedings 8th Symposium on Discrete Algorithms, 1997, pp. 599–608.

[Gra66] R. L. Graham, *Bounds for certain multiprocessing anomalies*, Bell system Technical Journal **45** (1966), 1563–1581.

[GS78] T. Gonzales and S. Sahni, *Flowshop and jobshop schedules: Complexity and approximation*, Operations Research **26** (1978), 36–52.

[GY92] J. R. Griggs and R. K. Yeh, *Labeling graphs with a condition at distance 2*, SIAM Journal of Discrete Mathematics **5** (1992), 586–595.

[GY98] L. Gross and J. Yellen, *Graph theory and its applications*, CRC Press, 1998, URL: http://www.graphtheory.com.

[Hal80] W. K. Hale, *Frequency assignment: Theory and Applications*, Proceedings of the IEEE, vol. 68, 1980, pp. 1497–1514.

[Has99] J. Hastad, *Clique is hard to approximate within* $n^{1-\varepsilon}$, Acta Mathematica **182** (1999), 105–142.

[HdVV94] J. A. Hoogeveen, S. L. Van de Velde, and B. Veltman, *Complexity of scheduling multiprocessor tasks with prespecified processor allocations*, Discrete Applied Mathematics **55** (1994), 259–272.

[Hel00] *The NGI Helious project: Regional Testbed Optical Access Network For IP Multicast and Differentiated Services*, http://projects.anr.mcnc.org/Helios/, 2000.

[HK73] J. E. Hopcroft and R. M. Karp, *A $n^{5/2}$ algorithm for maximum matchings in bipartite graphs*, SIAM Journal on Computing **2** (1973), 225–231.

[HK01] P. Hliněný and J. Kratochvíl, *Representing graphs by disks and balls*, Discrete Mathematics **229** (2001), 101–124.

[Hoc96] D. S. Hochbaum (ed.), *Approximation algorithms for NP-hard problems*, Thomson, 1996.

[HPW00] H. Hoogeveen, C. N. Potts, and G.J. Woeginger, *On-line scheduling on a single machine: maximizing the number of early jobs*, Operations Research Letters **27** (2000), 193–197.

[Hro01] J. Hromkovič, *Algorithms for Hard Problems*, Springer Verlag, 2001.

[HS78] E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms*, Computer Science Press, 1978.

[HSSW97] L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein, *Scheduling to minimize average time: Off-line and on-line algorithm*, Mathematics of Operation Research (1997), 513–544.

[HSW98] H. Hoogeveen, P. Schuurman, and G. Weoginger, *Non-approximability results for scheduling problems with minsum criteria*, Proceedings 6th Conference on Integer Programming and Combinatorial Optimization, LNCS 1412, Springer Verlag, 1998, pp. 353–366.

[Joh74] D. S. Johnson, *Approximation algorithms for combinatorial problems*, Journal of Computational System Science **9** (1974), 256–278.

[JP99a] K. Jansen and L. Porkolab, *General multiprocessor task scheduling: Approximate solution in linear time*, Proceedings 6th International Workshop on Algorithms and Data Structures, LNCS 1663, Springer Verlag, 1999, pp. 110–121.

[JP99b] K. Jansen and L. Porkolab, *Linear-time approximation schemes for scheduling malleable parallel tasks*, Proceedings 10th ACM-SIAM Symposium on Discrete Algorithms, 1999, To appear in Algorithmica, pp. 490–498.

[JSOS99] K. Jansen, R. Solis-Oba, and M. Sviridenko, *Makespan minimization in job shops: A polynomial time approximation scheme*, Proceedings 31st Annual ACM Symposium on Theory of Computing (Atlanta), 1999, To appear in SIAM Journal on Discrete Mathematics, pp. 394–399.

[JT95] T.R. Jensen and B. Toft, *Graph coloring problems*, Wiley-Interscience, 1995.

[Kar72] R. M. Karp, *Reducibility among combinatorial problems*, Proceedings of the Symposium on the Complexity of Computer Computations (York-town Heights, NY), Plenum Press, New York, March 1972, pp. 85–103.

[Kel99] H. Kellerer, *A polynomial time approximation scheme for the multiple knapsack problem*, Proceedings 2nd International Workshop on Approximation Algorithms for Combinatorial Optimization Problems & 3rd International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM-APPROX'99), LNCS 1671, Springer Verlag, 1999, pp. 51–62.

[KFH+00] M. Kuznetsov, N. Froberg, S. Henion, H. Rao, J. Korn, K. Rauschenbach, E. Modiano, and V. Chan, *A next-generation optical regional access networks*, IEEE Communications Magazine **38** (January 2000), 66 – 72.

[KIM78] H. Kise, T. Ibaraki, and H. Mine, *A solvable case of the one-machine scheduling with ready due times*, Operations Research **26** (1978), 121–126.

[KMSV94] S. Khanna, R. Motwani, M. Sudan, and U. Vazirani, *On syntactic versus computational views of approximability*, Proceedings 35th Annual Symposium on Foundations of Computer Science (Santa Fe, New Mexico), IEEE, 20-22 November 1994, pp. 819–830.

[Knu68] D. E. Knuth, *Fundamental algorithms*, The art of computer programming, vol. 1, Addison-Wesley, Reading, MA, 1968.

[Koe36] P. Koebe, *Kontaktprobleme der konformen Abbildung*, Math.-Phys. Klasse, vol. 88, Berichte Verhande. Saechs. Akad. Wiss. Leipzig, 1936, pp. 141–164.

[Kos99] A. Koster, *Frequency assignment: Models and algorithms*, Ph.D. thesis, Proefschrift Universiteit Maastricht, 1999.

[Krä95] A. Krämer, *Scheduling multiprocessor tasks on dedicated processors*, Ph.D. thesis, Fachbereich Mathematik/Informatik, Universität Osnabrück, 1995.

[Kub89] E. Kubicka, *The chromatic sum of a graph*, Ph.D. thesis, Western Michigan University, 1989.

[Law76] E.L. Lawler, *Sequencing to minimize the weighted number of tardy jobs*, RAIRO Recherche opérationnele **10** (1976), 27–33.

[Law82] E.L. Lawler, *Scheduling a single machine to minimize the number of late jobs*, Tech. report, Computer Science Division, University of California, Berkeley, 1982, Preprint.

[Lee98] R. A. Leese, *Radio spectrum: A raw material for the telecommunications industry*, Proceedings 10th Conference of the European Consortium for Mathematics in Industry (Goteborg), 1998.

[LK78] J. K. Lenstra and A. H. G. Rinnoy Kan, *Complexity of scheduling under precedence constraints*, Operations Research **26** (1978), 22–35.

[LKB77] J. K. Lenstra, A. H. G. Rinnoy Kan, and P. Brucker, *Complexity of machine scheduling problems*, Annals of Operation Research **4** (1977), 343–362.

[LLKS93] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, *Logistics of production and inventory*, Handbooks in Operation Research and Management Science, vol. 4, ch. Sequencing and scheduling: Algorithms and complexity, pp. 445–522, North-Holland, Amsterdam, 1993.

[Llo81] E.L. Lloyd, *Concurrent task systems*, Operations Research **29** (1981), 189–201.

[LM69] E.L. Lawler and J.M. Moore, *A functional equation and its application to resource allocation and sequencing problems*, Management Science **16** (1969), 77–84.

[LY94] C. Lund and M. Yannakakis, *On the hardness of approximating minimization problem*, Journal of the ACM **41** (1994), 960–981.

[Mal97] E. Malesińska, *Graph theoretical models for frequency assignment problems*, Ph.D. thesis, Technical University of Berlin, Germany, 1997.

[Maš01] K. Mašek, *Distance constrained labeling of disk graphs: An on-line algorithm*, http://www.ii.uib.no/~fiala/odcl, 2001.

[McC83] S. T. McCormick, *Optimal approximation of sparse Hessians and its equivalence to a graph coloring problem*, Mathematical Programming **26** (1983), 153–171.

[Mon82] C. L. Monma, *Linear-time algorithms for scheduling on parallel processors*, Operation Research **30** (1982), 116–124.

[Moo68] J. M. Moore, *A n jobs, one machine sequencing algorithm for minimizing the number of later jobs*, Management Science **15** (1968), 102–109.

[MPS98] E. W. Mayr, H. J. Prómel, and A. Steger (eds.), *Lectures on proof verification and approximation algorithms*, LNCS 1367, Springer Verlag, 1998.

[MS] M. Molloy and M. S. Salavatipour, *A bound on the chromatic number of the square of a planar graph*, manuscript.

[Muk92] B. Mukherjee, *WDM-Based local lightwave networks Part I: Single-hop systems*, IEEE Network Magazine (1992), 12–27.

[NW88] G. L. Nemhauser and L. A. Wolsey, *Integer and Combinatorial Optimization*, John Wiley and Sons, New York, 1988.

[Pap94] C. H. Papadimitriou, *Computational Complexity*, Addison Wesley, 1994.

[Pee91] R. Peeters, *On coloring j-unit sphere graphs*, Tech. report, Department of Economics, Tilburg University, 1991.

[Pin95] M. Pinedo, *Scheduling: Theory, algorithms and systems*, Prentice-Hall, 1995.

[PS82] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: Algorithms and complexity*, Prentice-Hall, 1982.

[PY91] C. H. Papadimitriou and M. Yannakakis, *Optimization, approximation, and complexity classes*, Journal of Computer and System Science **43** (1991), 425–440.

[QS00] M. Queyranne and M. Sviridenko, *New and improved algorithms for minsum shop scheduling*, Proceedings 11th Annual ACM-SIAM Symposium on Discrete Algorithms, 2000, pp. 871–878.

[RS01] V. Raghavan and J. Spinrad, *Robust algorithms for restricted domains*, Proceedings 12th Annual ACM-SIAM Symposium on Discrete Algorithms (Washington), 2001, pp. 460–467.

[RW98] G. Rote and G. J. Woeginger, *Minimizing the number of tardy jobs*, Acta Cybernetica **13** (1998), no. 4, 423–430.

[Sai95] R. Saigal, *Linear programming: A modern integrated analysis*, Kluwer Academic Publishers, 1995.

[Sch96] A. S. Schulz, *Polytopes and scheduling*, Ph.D. thesis, Technical University of Berlin, Germany, 1996.

[Sev86] S. V. Sevast'janov, *Bounding algorithm for the routing problem with arbitrary paths and alternative servers*, Cybernetics **22** (1986), 773–780, In Russian.

[SG76] S. Sahni and T. F. Gonzalez, *P-complete approximation problems*, Journal of the ACM **23** (1976), 555–565.

[Shm98] D. B. Shmoys, *Using linear programming in the design and analysis of approximation algorithms: Two illustrative examples*, Proceedings 1st International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, LNCS 1444, Springer Verlag, 1998, pp. 15–32.

[Sku98] M. Skutella, *Approximation and randomization in scheduling*, Ph.D. thesis, Technical University of Berlin, Germany, 1998.

[Smi56] W. E. Smith, *Various optimizers for single-stage production*, Naval Research Logistic Quarterly **3** (1956), 59–66.

[SS00] K. M. Sivalingam and S. Subramaniam (eds.), *Optical WDM networks: Principles and practice*, Kluwer Academic Publishers, 2000.

[SSW94] D. B. Shmoys, C. Stein, and J. Wein, *Improved approximation algorithms for shop scheduling problems*, SIAM Journal of Computing **23** (1994), 617–632.

[SU97] E. R. Scheinerman and D. Ullman, *Fractional graph theory*, John Wiley & Sons, 1997.

[SW97] C. Stein and J. Wein, *On the existence of schedules that are near-optimal for both makespan and total weighted completion time*, Operations Research Letters **21** (1997), 115–122.

[SW99] P. Schuurman and G. J. Woeginger, *Polynomial time approximation algorithms for machine scheduling: Ten open problems*, Journal of Scheduling **2** (1999), 203–213.

[SW02] P. Schuurman and G. J. Woeginger, *Approximation schemes – a tutorial*, To appear in the book Lectures on Scheduling, edited by R. H. Moehring, C. N. Potts, A. S. Schulz, G. J. Woeginger and L. A. Wolsey., 2002.

[SWW95] D.B. Shmoys, J. Wein, and D.P. Williamson, *Scheduling parallel machines on-line*, SIAM Journal on Computing **24** (1995), 1313–1331.

[TLWY94] J. Turek, W. Ludwig, J. Wolf, and P. Yu, *Scheduling parallel tasks to minimize average response times*, Proceedings 5th ACM-SIAM Symposium on Discrete Algorithms, 1994, pp. 112–121.

[TR00] D. Thaker and G. N. Rouskas, *Multi-destination communication in broadcast WDM networks: A survey*, Tech. Report 2000-08, North Caroline State University, 2000.

[TU99] E. Torng and P. Uthaisombut, *Lower bounds for SRPT-subsequence algorithms for non-preemptive scheduling*, Proceedings 10th ACM-SIAM Symposium on Discrete Algorithms, 1999, pp. 973–974.

[Vaz00] V. Vazirani, *Approximation algorithms*, Springer, New–York, 2000.

[vdHLS98] J. van den Heuvel, R. A. Leese, and M. A. Shepherd, *Graph labeling and radio channel assignment*, Journal of Graph Theory **29** (1998), 263–283.

[WASG96] R. E. Wagner, R. C. Alferness, A. A. M. Saleh, and M. S. Goodman, *MONET: Multiwavelength Optical Networking*, Journal of Lightwave Technology **14** (June 1996), 1349.

[Weg77] G. Wegner, *Graphs with a given diameter and a coloring problem*, Tech. report, University of Dortmund, Germany, 1977.

[Wes01] D. West, *Introduction to graph theory*, Prentice-Hall Inc., 2001.

[XL99] X.Cai and C.Y. Lee, *Scheduling multiprocessor tasks without prespecified processor allocations*, IIE Transactions - Scheduling and Logistics **31** (1999), 445–455.

INDEX

CONCLUSIONS

We presented approximation algorithms and online algorithms for several scheduling and labeling problems. Our work on the problem of minimizing average weighted completion time in Chapter 1 is primarily motivated by some theoretical questions which were open for a number of last years. We presented a general approximation method which leads to PTASs for two wide classes of scheduling problems with the average weighted completion time objective and release dates, which are strongly NP-hard even in very simple cases. The labeling problem in Chapter 2, which is a natural generalization of the classical graph coloring problem, and the multiprocessor task scheduling problems considered in Chapter 3 and Chapter 4 are new problems suggested by practical applications. We considered online and offline versions of the problems. In the offline setting, we first showed that the problems are NP-hard, and then presented approximation algorithms. In the online setting, we first presented online algorithms for the problems, and then derived upper and lower bounds on the competitive ratio.

Indeed, there are still many interesting research areas remain. At the end of each chapter we point out specific open problems related to the topic addressed in that chapter. Here we suggest some broader directions for future research.

Our first observation regards average (weighted) completion time scheduling, or similarly, the sum of (weighted) completion times. Minimizing makespan is a special case of the problem of minimizing average weighted completion time. Stein and Wein [SW97] show that, for a very general class of scheduling models, there exists a schedule that is simultaneously within a factor of 2 of the optimal schedule values for both average weighted completion time and makespan. Their proof is based on transforming on optimal schedule for average weighted completion time to a schedule that is approximately good for both objective functions. Chekuri [Che98] asked for a converse to their transformation. That is, is there a polynomial time algorithm that uses as a subroutine a procedure for minimizing makespan and outputs an approximate schedule for minimizing weighted completion time? In [CPS+96, QS00] and in this work a number of different approximation algorithms and methods have been presented which give answers to these interesting theoretical questions.

Here we raise the question of finding schedules that are simultaneously good for both the sum of completion times and the sum of weighted completion times.

This question seems to be non-trivial even in the case of scheduling on a single machine. Extending Chekuri's question, we ask for the design of algorithms in the multiple machine case which use approximation algorithms for the makespan objective as a subroutine.

Our next observation concerns the labeling problem. As we have observed in Chapter 2, the labeling problem is a good model for the frequency assignment problem [Kos99]. However, it is quite common in practice that neither coloring nor labeling of the interference graph fits well. The main problem behind this is that in the real frequency assignment some nodes, which are connected in the interference graph, can have the same frequency. In other words, nodes having either the same color or the same label can form a clique. In order to model this situation, we propose the following problem of minimizing the maximum *color clique*. We are given a graph G and k colors $1,2,3,\ldots,k$. A coloring is an assignment one of the colors to each node of G. (In a standard graph coloring it is required that very two connected nodes are colored distinctly.) Each set of the nodes having the same color induces a subgraph in G, and a clique in this subgraph is called a color clique. The maximum color clique in G is a color clique which has the maximum size. The goal is to find a coloring which minimizes the maximum color clique.

It is not hard to see that the complexity of the color clique problem depends on the number of available colors k. Furthermore, for a graph G and k colors, the maximum color clique is at least $\omega(G)/k$, where $\omega(G)$ is the clique number of G. In general, approximation algorithms for the chromatic number can be adopted for approximating the maximum color clique. The color clique problem is simple for approximation in planar graphs or trees. However, the problem seems to be non-trivial in the case of (unit) disk graphs.

Finally, we have an observation regarding multiprocessor task scheduling. We propose to consider the model in which task *splitting* is allowed. Informally, in splitting of a task, one first divides the set of required processors into a number subsets, and then consecutively executes of the task on each of these subsets of processors. From one side, this model fits more for applications in WDM LANs [BR02]. From another side, this model is strongly related to the multiprocessor task variant of flow, open and job shop scheduling problems [Krä95].

CURRICULUM VITAE

11. Aug. 1975:	born in Novokuznetsk, Russia.
1989-1991 :	student at Gymnasium No.11, Novokuznetsk, Russia.
1991 - 1992 :	student at Scientific Study Center, Novosibirsk, Russia.
1992 - 1996 :	student at Department of Mechanics and Mathematics, Novosibirsk State University, Novosibirsk, Russia.
June 1996:	BS degree in Mathematics (with honors).
1996 - 1998 :	master student at Department of Mechanics and Mathematics, and student at Department of Economics, Novosibirsk State University, Novosibirsk, Russia.
June 1998:	MS degree in Computer Science and Applied Mathematics, and Diploma of Economist.
Sept. 1998 : **April 1999**	field engineer trainee by Schlumberger Limited, Bryan , Texas, USA.
Nov. 1999 : **April 2002**	PhD student at Graduiertenkolleg 357 “Effiziente Algorithmen und Mehrskalenmethoden”, University of Kiel, Germany.
since May 2002:	research assistant at EU-Project CRESCO “Critical Resource Sharing for Cooperation in Complex Systems” Institute for Computer Science and Applied Mathematics, University of Kiel, Germany.