Algorithms and Approximation Schemes for Machine Scheduling Problems

by

Sudipta Sengupta

Bachelor of Technology (1997)
Computer Science and Engineering
Indian Institute of Technology, Kanpur

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1999

Author . . . .
Department of Electrical Engineering and Computer Science
May 11, 1999

Certified by . . . .
James B. Orlin
Edward Pennell Brooks Professor of Operations Research
Sloan School of Management
Thesis Supervisor

Accepted by . . . .
Arthur Smith
Chairman, Departmental Committee on Graduate Students

Algorithms and Approximation Schemes for Machine Scheduling Problems

by

Sudipta Sengupta

Submitted to the Department of Electrical Engineering and Computer Science on May 11, 1999, in Partial Fulfillment of the Requirements for the Degree of Master of Science in Electrical Engineering and Computer Science

Abstract

We present our results in the following two broad areas: (i) Algorithms and Approximation Schemes for Scheduling with Rejection, and (ii) Fixed-precision and Logarithmic-precision Models and Strongly Polynomial Time Algorithms.

Most of traditional scheduling theory begins with a set of jobs to be scheduled in a particular machine environment so as to optimize a particular optimality criterion. At times, however, a higher-level decision has to be made: given a set of tasks, and limited available capacity, choose only a subset of these tasks to be scheduled, while perhaps incurring some penalty for the jobs that are not scheduled, i.e., "rejected". We focus on techniques for scheduling a set of independent jobs with the flexibility of rejecting a subset of the jobs in order to guarantee an average good quality of service for the scheduled jobs.

Our interest in this *rejection model* arises from the consideration that in actual manufacturing settings, a scheduling algorithm is only one element of a larger decision analysis tool that takes into account a variety of factors such as inventory and potential machine disruptions. Furthermore, some practitioners have identified as a critical element of this larger decision picture the "pre-scheduling negotiation" in which one considers the capacity of the production environment and then agrees to schedule certain jobs at the requested quality of service and other jobs at a reduced quality of service, while rejecting other jobs altogether. Typically, the way that current systems handle the integration of objective functions is to have a separate component for each problem element and to integrate these components in an ad-hoc heuristic fashion. Therefore, models that integrate more than one element of this decision process while remaining amenable to solution by algorithms with performance guarantees have the potential to be very useful elements in this decision process.

We consider the *offline* scheduling model where all the jobs are available at time zero. For each job, we must decide either to schedule it or to reject it. If we schedule a job, we use up machine resources for its processing. If we reject a job, we pay its rejection cost. Our goal is to choose a subset S of the jobs to schedule on the machine(s) so as to minimize an objective function which is the sum of a function $F(S)$ of the set S of scheduled jobs and the total rejection penalty of the set $\bar{S}$ of rejected jobs. Thus, our general optimization problem may be denoted as

$$\min_{S} \left[F(S) + \sum_{j \in \bar{S}} e_j \right]$$

We consider four possible functions $F(S)$, giving rise to the following four problems: (i) sum of weighted completion times with rejection $[F(S) = \sum_{j \in S} w_j C_j]$, (ii) maximum lateness with rejection $[F(S) = L_{max}(S)]$, (iii) maximum tardiness with rejection $[F(S) = T_{max}(S)]$, and (iv) makespan with rejection $[F(S) = C_{max}(S)]$. For each of these problems, we give hardness ($\mathcal{NP}$-completeness) results, pseudo-polynomial time algorithms (based on dynamic programming), and fully polynomial time approximation schemes (FPTAS). For the problem of sum of weighted completion times with rejection, we present simple and efficient polynomial time algorithms based on greedy methods for certain special cases. Observe that the notion of an approximation algorithm (in the usual sense) does not make much sense for maximum lateness with rejection since the optimal objective function value could be negative. Hence, we give *inverse approximation* algorithms for this problem. We also point out the application of the maximum tardiness with rejection problem to hardware/software codesign for real-time systems.

We introduce a new model for algorithm design which we call the *L-bit precision model.* In this model, the input numbers are of the form $c * 2^t$, where $t \geq 0$ is arbitrary and $c < c^* = 2^L$. Thus, the input numbers have a precision of L bits. The L-bit precision model is realistic because it incorporates the format in which large numbers are stored in computers today. In this L-bit precision model, we define a *polynomial time algorithm* to have running time polynomial in $c^* = 2^L$ and n, where n is the size of the problem instance. Depending on the value of L, we have two different models of precision. For the *fixed-precision model*, L is a constant. For the *logarithmic-precision model*, L is $O(\log n)$. Under both these models, a polynomial time algorithm is also a *strongly polynomial time algorithm*, i.e., the running time does not depend on the sizes of the input numbers.

We focus on designing algorithms for $\mathcal{NP}$-complete problems under the above models. In particular, we give strongly polynomial time algorithms for the following $\mathcal{NP}$-complete problems: (i) the *knapsack* problem, (ii) the *k- partition* problem for a fixed k, and (iii) *scheduling to minimize makespan* on a fixed number of identical parallel machines. Our results show that it is possible for $\mathcal{NP}$-complete problems to have strongly polynomial time algorithms under the above models.

Thesis Supervisor: James B. Orlin
Title: Edward Pennell Brooks Professor of Operations Research
Sloan School of Management

Acknowledgments

I would first like to thank my advisor Prof. Jim Orlin for his constant encouragement, support, and guidance. He made himself frequently available to help me identify new research directions, clarify my ideas and explanations, and often add an entirely new perspective to my understanding of the research problem.

I must also thank Daniel Engels and David Karger with whom I collaborated on part of the work contained in the thesis. I would especially like to thank Daniel for always being available as a sounding board for many of the ideas that developed into this thesis.

Special thanks to Be for keeping the office well stocked with chocolates, cookies, and candies. My former and present apartment mates, Pankaj, Debjit, and Evan, have made MIT and Cambridge a much more fun and enjoyable place. A very special thanks to Raj for being a friend, philosopher, and patient listener during hard times.

Most of all, I must thank my family – my parents, my brother, my late grandparents, and my grandmother. My parents, for instilling in me a love for books and knowledge and for their encouragement and dedicated support towards my setting and achieving higher goals in life; my brother, for his confidence in me and for always looking up to me; my late grandparents, who would have been the happiest persons to see my academic career developing; my grandmother, for her love and affection. They laid the foundations on which this thesis rests.

To my parents

Contents

List of Figures

Chapter 1

Introduction

In this chapter, we introduce and motivate the research problems considered in the thesis, define the notation that will be used throughout the thesis, and summarize the results that we have obtained. This thesis consists of results in the following broad research areas:

- Algorithms and Approximation Schemes for Scheduling with Rejection
- Fixed-precision and Logarithmic-precision Models and Strongly Polynomial Time Algorithms

In Section 1.1, we introduce and motivate the concept of *scheduling with rejection* and summarize our results in this area. In Section 1.2, we introduce and motivate the *fixed-precision* and *logarithmic-precision models* and state the problems for which we have obtained strongly polynomial time algorithms under these models.

The rest of this thesis is organized as follows. Chapters 2, 3, and 4 contain our results in the area of scheduling with rejection. In Chapter 2, we discuss our results for the problem of sum of weighted completion times with rejection. In Chapter 3, we discuss our results for the problems of maximum lateness with rejection and maximum tardiness with rejection. These two problems are very closely related to each other and hence, we have combined our results for each of them into a single chapter. In Chapter 4, we discuss our results for the problem of makespan with rejection. Chapter 5 contains our results on strongly polynomial time algorithms under the fixed-precision and logarithmic-precision models.

1.1 Scheduling with Rejection.

We begin with a short introduction to the area of scheduling in Section 1.1.1. In Section 1.1.2, we introduce and motivate the problem of scheduling with rejection. In Section 1.1.3, we introduce the scheduling notation that will be used throughout this thesis. In Section 1.1.4, we give formal definitions for the scheduling with rejection problems that we consider. In Section 1.1.5, we give pointers to some relevant previous work in the area of scheduling with rejection. In Sections 1.1.6, 1.1.7, and 1.1.8, we summarize our results for each of the following problems respectively: (i) sum of weighted completion times with rejection, (ii) maximum lateness/tardiness with rejection, and (iii) makespan with rejection.

1.1.1 Introduction to Scheduling.

Scheduling as a research area is motivated by questions that arise in production planning, in computer control, and generally in all situations in which scarce resources have to be allocated to activities over time. Scheduling is concerned with the *optimal allocation of scarce resources to activities over time.* This area has been the subject of extensive research since the early 1950's and an impressive amount of literature has been created. A good survey of the area is [24].

A *machine* is a resource that can perform at most one activity at any time. The activities are commonly referred to as *jobs*, and it is also assumed that a job requires at most one machine at any time. Scheduling problems involve *jobs* that must be scheduled on *machines* subject to certain *constraints* to optimize some *objective function.* The goal is to produce a *schedule* that specifies when and on which machine each job is to be executed.

1.1.2 Motivation for Scheduling with Rejection.

Most of traditional scheduling theory begins with a set of jobs to be scheduled in a particular machine environment so as to optimize a particular optimality criterion. At times, however, a higher-level decision has to be made: given a set of tasks, and limited available capacity, choose only a subset of these tasks to be scheduled, while perhaps incurring some penalty for the jobs that are not scheduled, i.e., rejected. We focus on techniques for scheduling a set of independent jobs with the flexibility of rejecting a subset of the jobs in order to guarantee an average good quality of service for the scheduled jobs.

Our interest in this *rejection model* arises primarily from two considerations. First, it is becoming widely understood that in actual manufacturing settings, a scheduling algorithm is only one element of a larger decision analysis tool that takes into account a variety of factors such as inventory, potential machine disruptions, and so on and so forth [15]. Furthermore, some practitioners [13] have identified as a critical element of this larger decision picture the "pre-scheduling negotiation" in which one considers the capacity of the production environment and then agrees to schedule certain jobs at the requested quality of service (e.g., job turnaround time) and other jobs at a reduced quality of service, while rejecting other jobs altogether.

Typically, the way that current systems handle the integration of objective functions is to have a separate component for each problem element and to integrate these components in an ad-hoc heuristic fashion. Therefore, models that integrate more than one element of this decision process while remaining amenable to solution by algorithms with performance guarantees have the potential to be very useful elements in this decision process.

Secondly, our work is directly inspired by that of Bartal, Leonardi, Marchetti-Spaccamela, Sgall and Stougie [1] and Seiden [20] who studied a multiprocessor scheduling problem with the objective of trading off between schedule *makespan* (length) and job rejection penalty. Makespan, sum of weighted completion times, and maximum lateness/tardiness are among the most basic and well-studied of all scheduling optimality criteria; therefore, it is of interest to understand the impact of the "rejection option" on these criteria.

1.1.3 Scheduling Notation.

We will use the scheduling notation introduced by Graham, Lawler, Lenstra and Rinnooy Kan [7]. We will describe their notation with reference to the models that we consider.

Each scheduling problem can be described as $\alpha|\beta|\gamma$ where α denotes the *machine environment*, β the *side constraints*, and γ the *objective function.* The machine environment α can be 1, P, Q or R, denoting one machine, m *identical* parallel machines, m *uniform* parallel machines, or m *unrelated* parallel machines respectively. On identical machines, job j takes processing time p_j regardless of which machine processes it. On uniform machines, job j takes processing time $p_{ji} = p_j/s_i$ on machine i, where p_j is a measure of the size of job j and s_i is the (job-independent) speed of machine i. On unrelated machines, job j takes processing time p_{ji} on machine i, i.e., there is no assumed relationship among the speeds of the machines with respect to the jobs. When the number of machines under consideration is fixed, say m, we denote this by appending an m to the machine environment notation, i.e., Pm, Qm, or Rm.

The side constraints β denote any special conditions in the model – it can be release dates r_j, precedence constraints $\prec$, both or neither indicating whether there are release dates, precedence constraints, both or neither. In this thesis, we consider the *offline* scheduling model where all the jobs are available at time zero and are independent, i.e., there are no precedence constraints. Hence, this field will usually be empty.

Let each job j have a processing time p_j and and a rejection cost e_j. Depending on the problem under consideration, each job j may also have the following parameters: due date d_j and associated weight w_j. The set of scheduled jobs is denoted by S, and the set of rejected jobs is denoted by $\bar{S} = N - S$, where $N = \{1, 2, \ldots, n\}$ is the original set of jobs. For job j, the completion time is denoted by C_j, the lateness is denoted by $L_j = C_j - d_j$, and the tardiness is denoted by $T_j = \max(0, L_j)$.

For any set S of scheduled jobs, the makespan of the schedule is denoted by $C_{max}(S) = \max_{j \in S} C_j$, the maximum lateness of the schedule is denoted by $L_{max}(S) = \max_{j \in S} L_j$, and the maximum tardiness of the schedule is denoted by $T_{max}(S) = \max_{j \in S} T_j$. Note that $T_{max}(S) = \max(0, L_{max}(S))$.

1.1.4 Problem Definition.

We are given a set of n independent jobs $N = \{1, 2, \ldots, n\}$, each with a positive processing time p_j and a positive *rejection cost* e_j. In the case of multiple machines, we are given the processing time p_{jk} of job j on machine k. Depending on the problem under consideration, each job j may also have the following parameters: due date d_j and associated weight w_j. We consider the *offline* scheduling model, where all the jobs are available at time zero. For simplicity, we will assume that the processing times, rejection costs, due dates, and weights are all integers.

For each job, we must decide either to schedule that job (on a machine that can process at most one job at a time) or to reject it. If we schedule job j, we use up machine resources for its processing. If we reject job j, we pay its rejection cost e_j. For most of this thesis, we define the *total rejection penalty* of a set of jobs as the sum of the rejection costs of these jobs. However, in Section 3.6.3, we consider a problem where the the total rejection penalty of a set of jobs is the *product* (and not the sum) of the rejection costs of these jobs.

Our goal is to choose a subset $S \subseteq N$ of the n jobs to schedule on the machine(s) so as to minimize an objective function which is the sum of a function $F(S)$ of the set of scheduled jobs and the total rejection penalty of the rejected jobs. We denote the set of rejected jobs by $\bar{S} = N - S$. Thus, our general optimization problem may be written as

$$\min_{S \subseteq N} \left[F(S) + \sum_{j \in \bar{S}} e_j \right]$$

In this thesis, we consider the following four functions for $F(S)$. We give both exact and approximation algorithms in each case.

Sum of Weighted Completion Times with Rejection: For this problem, $F(S) = \sum_{j \in S} w_j C_j$. Thus, the objective function is the sum of the weighted completion times of the scheduled jobs and the total rejection penalty of the rejected jobs, and the optimization problem can be written as

$$\min_{S \subseteq N} \left[\sum_{j \in S} w_j C_j + \sum_{j \in \bar{S}} e_j \right]$$

Maximum Lateness with Rejection: For this problem, $F(S) = L_{max}(S) = \max_{j \in S} L_j$. Thus, the objective function is the sum of the maximum lateness of the scheduled jobs and the total rejection penalty of the rejected jobs, and the optimization problem can be written as

$$\min_{S \subseteq N} \left[L_{max}(S) + \sum_{j \in \bar{S}} e_j \right]$$

Maximum Tardiness with Rejection: For this problem, $F(S) = T_{max}(S) = \max_{j \in S} T_j$. Thus, the objective function is the sum of the maximum tardiness of the scheduled jobs and the total rejection penalty of the rejected jobs, and the optimization problem can be written as

$$\min_{S \subseteq N} \left[T_{max}(S) + \sum_{j \in \bar{S}} e_j \right]$$

Makespan with Rejection: For this problem, $F(S) = C_{max}(S) = \max_{j \in S} C_j$. Thus, the objective function is the sum of the makespan of the scheduled jobs and the total rejection penalty of the rejected jobs, and the optimization problem can be written as

$$\min_{S \subseteq N} \left[C_{max}(S) + \sum_{j \in \bar{S}} e_j \right]$$

Note that if the rejection cost e_j for a job j is zero, we can reject job j and solve the problem for the remaining jobs. Hence, we make the reasonable assumption that every rejection cost is positive. This, together with the integrality assumption, implies that $e_j \geq 1$ for all j.

Observe also that when we make the rejection costs arbitrarily large (compared to the other parameters of the problem like processing times, weights, and due dates), each of the above problems reduces to the corresponding problem *without rejection.* We say that problem version with rejection can be *restricted* to the problem version without rejection in this manner. Thus, the problem version with rejection is *at least as hard* as the problem version where rejection is not considered. We will make use of this simple fact in some of the hardness results that we prove.

1.1.5 Relevant Previous Work.

There has been much work on scheduling without rejection to minimize $F(N)$ for a set N of jobs and the choices for the function F mentioned in the previous section. For the problem $1 | | \sum w_j C_j$ (scheduling jobs with no side constraints on one machine to minimize $\sum w_j C_j$), Smith [21] proved that an optimal schedule could be constructed by scheduling the jobs in non-decreasing order of p_j/w_j ratios. More complex variants are typically $\mathcal{NP}$-hard, and recently there has been a great deal of work on the development of approximation algorithms for them (e.g., [16, 8, 6, 2, 19, 18]). A ρ-*approximation algorithm* is a polynomial-time algorithm that always finds a solution of objective function value within a factor of ρ of optimal (ρ is also referred to as the *performance guarantee*).

For the problem $1|\,|L_{max}$ (scheduling jobs with no side constraints on one machine to minimize the maximum lateness L_{max}), Lawler [22] proved that an optimal schedule could be constructed by scheduling the jobs in non-decreasing order of due dates d_j. This rule is also called the *Earliest Due Date (EDD)* rule. An excellent reference for the four problems mentioned in Section 1.1.4 when rejection is not considered is the book by Peter Brucker [23].

To the best of our knowledge, the only previous work on scheduling models that include rejection in our manner is that of [1, 20]. Bartal et. al. seem to have formally introduced the notion. They considered the problem of scheduling with rejection in a multiprocessor setting with the aim of minimizing the makespan of the scheduled jobs and the sum of the penalties of the rejected jobs [1]. They give a $(1 + \phi)(\approx 2.618)$-competitive algorithm for the on-line version, where ϕ is the golden ratio. Seiden [20] gives a 2.388-competitive algorithm for the above problem when preemption is allowed. Our work on makespan with rejection differs from the above in that we focus on the offline model where all the jobs are available at time zero.

1.1.6 Results on Sum of Weighted Completion Times with Rejection.

In Chapter 2, we consider the problem of sum of weighted completion times with rejection for which the objective function is the sum of weighted completion times of the scheduled jobs and the total rejection penalty of the rejected jobs. The one machine version of this problem is denoted as $1|\,|(\sum_S w_j C_j + \sum_{\bar{S}} e_j)$. If rejection is not considered, the problem is solvable in polynomial time using *Smith's rule*: schedule the jobs in non-decreasing order of p_j/w_j. In Section 2.2, we show that adding the option of rejection makes the problem weakly $\mathcal{NP}$-complete, even on one machine. This result reflects joint work with Daniel Engels [3, 4]. In Section 2.3, we give two pseudo-polynomial time algorithms, based on dynamic programming, for $1| |(\sum_S w_jC_j + \sum_{\bar{S}} e_j)$. The first runs in $O(n \sum_{j=1}^{n} w_j)$ time and the second runs in $O(n \sum_{j=1}^{n} p_j)$ time. In Section 2.3.3, we generalize our algorithms to any fixed number of uniform parallel machines and solve $Qm| |(\sum_S w_jC_j + \sum_{\bar{S}} e_j)$.

We also develop a fully polynomial time approximation scheme (FPTAS) for $1| |$-$(\sum_S w_jC_j + \sum_{\bar{S}} e_j)$ in Section 2.4. The FPTAS uses a *geometric rounding* technique on the job completion times and works with what we call *aligned schedules.* In our FPTAS, we constrain each job to finish at times of the form $(1+\epsilon/2n)^i$, where $(1+\epsilon)$ is the factor of approximation achieved by the algorithm. This might introduce idle time in a schedule, but we argue that the objective function value increases by a factor of at most $(1+\epsilon)$. We also generalize our FPTAS to any fixed number of uniform parallel machines and solve $Qm| |(\sum_S w_jC_j + \sum_{\bar{S}} e_j)$. This is joint work with David Karger [4].

In Section 2.5, we consider two special cases for which simple greedy algorithms exist to solve $1| |(\sum_S w_jC_j + \sum_{\bar{S}} e_j)$. In Section 2.5.1, we give an $O(n^2)$ time algorithm for the case when all weights are equal, i.e., for $1|w_j = w|(\sum_S w_jC_j + \sum_{\bar{S}} e_j)$. This algorithm also works for the case when all processing times are equal, i.e., for $1|p_j = p|(\sum_S w_jC_j + \sum_{\bar{S}} e_j)$. In Section 2.5.2, we define the concept of compatible processing times, weights, and rejection costs and then give an $O(n \log n)$ time algorithm for this case. The latter is joint work with Daniel Engels.

1.1.7 Results on Maximum Lateness/Tardiness with Rejection.

In Chapter 3, we consider the problem of maximum tardiness/lateness with rejection for which the objective function is the sum of the maximum lateness/tardiness of the scheduled jobs and the total rejection penalty of the rejected jobs. The one machine versions of these two problems are denoted as $1| \ |(L_{max}(S)+\sum_{\bar{S}} e_j)$ and $1| \ |(T_{max}(S)+\sum_{\bar{S}} e_j)$ respectively. In Section 3.2, we motivate the maximum tardiness with rejection problem by pointing out applications to hardware/software codesign for real-time systems. If rejection is not considered, the problems are solvable in polynomial time on one machine using the *Earliest Due Date (EDD)* rule: schedule the jobs in non-decreasing order of due dates d_j. In Section 3.3, we show that adding the option of rejection makes the problems weakly $\mathcal{NP}$-complete, even on one machine. This result reflects joint work with James Orlin. In Section 3.4, we give two pseudo-polynomial time algorithms, based on dynamic programming, for $1| \ |(L_{max}(S) + \sum_{\bar{S}} e_j)$ and $1| \ |(T_{max}(S) + \sum_{\bar{S}} e_j)$. The first runs in $O(n\sum_{j=1}^{n} e_j)$ time and the second runs in $O(n\sum_{j=1}^{n} p_j)$ time. We generalize the second algorithm in Section 3.4.3 to any fixed number of unrelated parallel machines and solve $Rm| \ |(L_{max}(S) + \sum_{\bar{S}} e_j)$ and $Rm| \ |(T_{max}(S) + \sum_{\bar{S}} e_j)$.

We also develop an FPTAS for $1| \ |(T_{max}(S) + \sum_{\bar{S}} e_j)$ in Section 3.5. The FPTAS uses a *geometric rounding* technique on the total rejection penalty and works with what we call the *inflated rejection penalty.* In our FPTAS, we constrain the total rejection penalty for each set of rejected jobs to be of the form $(1 + \epsilon/2n)^i$, where $(1 + \epsilon)$ is the factor of approximation achieved by the algorithm.

Observe that the notion of an approximation algorithm (in the usual sense) does not hold much meaning for $1| \ |(L_{max}(S) + \sum_{\bar{S}} e_j)$ because the optimal objective function value could be negative. In such a case, it makes sense to consider *inverse approximation* algorithms for the problem. We introduce and motivate the notion of an inverse approximation algorithm in Section 3.6. We then give an inverse approximation scheme for $1| |(L_{max}(S) + \sum_{\bar{S}} e_j)$ in Section 3.6.2 and a fully polynomial time inverse approximation scheme (IFPTAS) for the problem $1| |(L_{max}(S) + \prod_{\bar{S}} e_j)$ in Section 3.6.3, where the total rejection penalty is the *product* (and not the sum) of the rejection costs of the rejected jobs.

1.1.8 Results on Makespan with Rejection.

In Chapter 4, we consider the problem of makespan with rejection for which the objective function is the sum of the makespan of the scheduled jobs and the total rejection penalty of the rejected jobs. The one machine version of this problem is denoted as $1| |(C_{max}(S) + \sum_{\bar{S}} e_j)$. If rejection is not considered, the problem is trivial on one machine and $\mathcal{NP}$-complete on more than one machine. In Section 4.2, we give a simple $O(n)$ time algorithm for $1| |(C_{max}(S) + \sum_{\bar{S}} e_j)$. In Section 4.3, we give a pseudo-polynomial time algorithm, based on dynamic programming, for the problem on a fixed number of unrelated parallel machines, i.e., $Rm| |(C_{max}(S) + \sum_{\bar{S}} e_j)$.

We also develop a fully polynomial time approximation scheme (FPTAS) for $Rm| |(C_{max}(S) + \sum_{\bar{S}} e_j)$ in Section 4.4. The FPTAS uses a *geometric rounding* technique on the job completion times and works with *aligned schedules*. In our FPTAS, we constrain each job to finish at times of the form $(1 + \epsilon/2n)^i$, where $(1 + \epsilon)$ is the factor of approximation achieved by the algorithm.

1.2 Fixed-precision and Logarithmic-precision Models and Strongly Polynomial Time Algorithms.

1.2.1 Introduction.

We introduce a new model for algorithm design which we call the *L-bit precision model.* In this model, the input numbers (for the problem) are of the form $c * 2^t$, where $t \geq 0$ is arbitrary and $c < c^* = 2^L$. Thus, the input numbers have a precision of L bits. The L-bit precision model is realistic because it incorporates the format in which large numbers are stored in computers today. One such format which is becoming increasingly popular in the computer industry is the *IEEE Standard for Binary Floating-point Arithmetic* [28, 29, 30].

In this L-bit precision model, we define a *polynomial time algorithm* to have running time polynomial in $c^* = 2^L$ and n, where n is the size of the problem instance. Depending on the value of L, we have two different models of precision which we describe below.

Fixed-precision Model: In this model, L is a constant, so that a polynomial time algorithm has running time polynomial in n under this model, and is, hence, a *strongly polynomial time algorithm*, i.e., the running time does not depend on the sizes of the input numbers.

Logarithmic-precision Model: In this model, L is $O(\log n)$, where n is the size of the problem instance. This implies that $c^* = 2^L$ is polynomial in n. Hence, a polynomial time algorithm is also a *strongly polynomial time algorithm* under this model.

We focus on designing algorithms for $\mathcal{NP}$-complete problems under the above models. In particular, we give strongly polynomial time algorithms for the following

$\mathcal{NP}$-complete problems: (i) the *knapsack* problem, (ii) the *k-partition* problem for a fixed k, and (iii) *scheduling to minimize makespan* on a fixed number of identical parallel machines. This is joint work with James Orlin. Our results show that it is possible for $\mathcal{NP}$-complete problems to have strongly polynomial time algorithms under the above models.

1.2.2 The Knapsack Problem.

The *knapsack* problem is defined as follows:

> Given sets $A = \{a_1, a_2, \ldots, a_n\}$ and $C = \{c_1, c_2, \ldots, c_n\}$ of n numbers each and a number b, find a set $S \subseteq \{1, 2, \ldots, n\}$ which maximizes $\sum_{j \in S} c_j$ subject to the condition that $\sum_{j \in S} a_j \leq b$.

We can interpret the above definition as follows. Suppose there are n items, with item i having weight c_i and volume a_i. The items are to be placed in a knapsack with (volume) capacity b. We want to find the subset S of items which can be put into the knapsack to maximize the total weight of items in the knapsack.

In Section 5.2, we give an $O(c^* n^2)$ time algorithm for this problem, where $c^* = 2^L$ and the c_i's have L bits of precision (the a_i's could be arbitrary). This is a strongly polynomial time algorithm.

1.2.3 The k-Partition Problem.

The *k-partition* problem is defined as follows:

> Given a set $A = \{a_1, a_2, \ldots, a_n\}$ of n numbers such that $\sum_{i=1}^{n} a_i = kb$, is there a partition of A into k subsets $A_1, A_2, \ldots, A_k$ such that $\sum_{a_i \in A_j} a_i = b$ for all $1 \leq j \leq k$?

In Section 5.3, we give an $O(kn(c^*n)^k)$ time algorithm for this problem, where $c^* = 2^L$ and the a_i's have L bits of precision. For a fixed k, this is a strongly polynomial time algorithm.

1.2.4 Scheduling to Minimize Makespan on Identical Parallel Machines.

We consider the problem of scheduling n jobs on any fixed number m of identical parallel machines in order to minimize the makespan. Each job j has a processing time p_j on any machine. In scheduling notation, this problem is denoted as $Pm| |$-C_{max}.

The decision problem formulation of $Pm| |C_{max}$ is defined as follows:

> Given a set of n independent jobs, $N = \{J_1, \dots , J_n\}$, with processing times p_j, $\forall\ 1 \leq j \leq n$, a fixed number m of identical parallel machines, and a number b, is there a schedule of the n jobs on the m machines such that the makespan on each machine is at most b ?

In Section 5.4, we give an $O(nm(c^*n)^m)$ time algorithm for this problem, where $c^* = 2^L$ and the p_i's have L bits of precision. For a fixed m, this is a strongly polynomial time algorithm.

Chapter 2

Sum of Weighted Completion Times with Rejection

2.1 Introduction.

In this chapter, we consider the problem of sum of weighted completion times with rejection for which the objective function is the sum of weighted completion times of the scheduled jobs and the total rejection penalty of the rejected jobs. The one machine version of this problem is denoted as $1| |(\sum_S w_jC_j + \sum_{\bar{S}} e_j)$. If rejection is not considered, the problem is solvable in polynomial time using *Smith's rule*: schedule the jobs in non-decreasing order of p_j/w_j. In Section 2.2, we show that adding the option of rejection makes the problem $\mathcal{NP}$-complete, even on one machine. This result reflects joint work with Daniel Engels [3, 4]. In Section 2.3, we give two pseudo-polynomial time algorithms, based on dynamic programming, for $1| |$-$(\sum_S w_jC_j + \sum_{\bar{S}} e_j)$. The first runs in $O(n\sum_{j=1}^{n} w_j)$ time and the second runs in $O(n\sum_{j=1}^{n} p_j)$ time. In Section 2.3.3, we generalize our algorithms to any fixed number of uniform parallel machines and solve $Qm| |(\sum_S w_jC_j + \sum_{\bar{S}} e_j)$.

We also develop a fully polynomial time approximation scheme (FPTAS) for $1| |$-$(\sum_S w_jC_j + \sum_{\bar{S}} e_j)$ in Section 2.4. The FPTAS uses a *geometric rounding* technique on the job completion times and works with what we call *aligned schedules.* In our FPTAS, we constrain each job to finish at times of the form $(1+\epsilon/2n)^i$, where $(1+\epsilon)$ is the factor of approximation achieved by the algorithm. This might introduce idle time in a schedule, but we argue that the objective function value does not increase by more than a $(1+\epsilon)$ factor. We also generalize our FPTAS to any fixed number of uniform parallel machines and solve $Qm|\,|(\sum_S w_jC_j + \sum_{\bar{S}} e_j)$. This is joint work with David Karger [4].

In Section 2.5, we consider two special cases for which simple greedy algorithms exist to solve $1|\,|(\sum_S w_jC_j+\sum_{\bar{S}} e_j)$. In Section 2.5.1, we give an $O(n^2)$ time algorithm for the case when all weights are equal, i.e., for $1|w_j = w|(\sum_S w_jC_j + \sum_{\bar{S}} e_j)$. This algorithm also works for the case when all processing times are equal, i.e., for $1|p_j = p|(\sum_S w_jC_j+\sum_{\bar{S}} e_j)$. In Section 2.5.2, we define the concept of compatible processing times, weights, and rejection costs and then give an $O(n \log n)$ time algorithm for this case. The latter is joint work with Daniel Engels.

2.2 Complexity of Sum of Weighted Completion Times with Rejection.

In this section, we show that the problem $Pm|\,|(\sum_S w_jC_j + \sum_{\bar{S}} e_j)$ is $\mathcal{NP}$-complete for a fixed number of machines $m \geq 1$. For any fixed number of machine $m > 1$, $Pm|\,|(\sum_S w_jC_j + \sum_{\bar{S}} e_j)$ is trivially seen to be $\mathcal{NP}$-complete by restricting the problem to $Pm|\,|\sum w_jC_j$, a known $\mathcal{NP}$-complete problem [5]. This problem is solvable in polynomial time on one machine using Smith's rule *when rejection is not considered.* However, we prove that adding the rejection option makes even the *single* machine problem $\mathcal{NP}$-complete.

The decision problem formulation of $1|\,|(\sum_S w_jC_j + \sum_{\bar{S}} e_j)$ is defined as follows:

Given a set of n independent jobs, $N = \{J_1, \ldots, J_n\}$, with processing times p_j, $\forall\ 1 \leq j \leq n$, weights w_j, $\forall\ 1 \leq j \leq n$, and rejection penalties e_j, $\forall\ 1 \leq j \leq n$, a single machine, and a number K, is there a schedule of a subset of jobs $S \subseteq N$ on the machine such that $\sum_{j \in S} w_j C_j + \sum_{j \in \bar{S} = N-S} e_j \leq K$?

We reduce the Partition Problem [5] to this problem proving that even on one machine, scheduling with rejection is $\mathcal{NP}$-complete.

Theorem 2.2.1 $1| \ |(\sum_S w_j C_j + \sum_{\bar{S}} e_j)$ *is $\mathcal{NP}$-complete.*

Proof. $1| \ |(\sum_S w_j C_j + \sum_{\bar{S}} e_j)$ is clearly in $\mathcal{NP}$. To prove that it is also $\mathcal{NP}$-hard, we reduce the Partition Problem [5] to $1| \ |(\sum_S w_j C_j + \sum_{\bar{S}} e_j)$. The Partition Problem is defined as follows:

Given a set $A = \{a_1, a_2, \ldots, a_n\}$ of n numbers such that $\sum_{i=1}^n a_i = 2b$, is there a subset A' of A, such that $\sum_{a_i \in A'} a_i = b$?

Given an instance $A = \{a_1, \ldots, a_n\}$ of the partition problem, we create an instance of $1| \ |(\sum_S w_j C_j + \sum_{\bar{S}} e_j)$ with n jobs, $J_1, \ldots, J_n$. Each of the n elements a_i in the Partition Problem corresponds to a job J_i in $1| \ |(\sum_S w_j C_j + \sum_{\bar{S}} e_j)$ with weight and processing time equal to a_i and rejection cost equal to $ba_i + \frac{1}{2}a_i^2$, where $b = \frac{1}{2}\sum_{i=1}^n a_i$. Since $p_j = w_j, \forall\ 1 \leq j \leq n$, the ordering of the scheduled jobs does not affect the value of $\sum_{j \in S} w_j C_j$. Using this fact and substituting for the rejection penalty, we can rewrite the objective function as follows:

$$\begin{aligned}
\sum_{j \in S} w_j C_j + \sum_{j \in \bar{S}} e_j &= \sum_{j \in S} a_j \sum_{(i \leq j, i \in S)} a_i + \sum_{j \in \bar{S}} e_j \\
&= \sum_{j \in S} a_j^2 + \sum_{(i<j, i,j \in S)} a_i a_j + \sum_{j \in \bar{S}} (ba_j + \frac{1}{2}a_j^2) \\
&= \frac{1}{2}[(\sum_{j \in S} a_j)^2 + \sum_{j \in S} {a_j}^2] + b\sum_{j \in \bar{S}} a_j + \frac{1}{2}\sum_{j \in \bar{S}} {a_j}^2 \\
&= \frac{1}{2}(\sum_{j \in S} a_j)^2 + b\sum_{j \in \bar{S}} a_j + \frac{1}{2}\sum_{j=1}^{n} {a_j}^2
\end{aligned}$$

$$= \frac{1}{2}\Big(\sum_{j\in S} a_j\Big)^2 + b\Big(2b - \sum_{j\in S} a_j\Big) + \frac{1}{2}\sum_{j=1}^{n} a_j^{\,2}$$

Since $\sum_{j=1}^{n} a_j^2$ is a constant, minimizing the objective function is equivalent to minimizing the following function with $x = \sum_{j\in S} a_j$

$$\frac{1}{2}x^2 + b(2b - x)$$

This function has a unique minimum of $\frac{3}{2}b^2$ at $x = b$, i.e., the best possible solution has $\sum_{j\in S} a_j = b$, and hence, is optimum if it exists. Therefore, if the optimum solution to $1| \,|(\sum_S w_jC_j + \sum_{\bar{S}} e_j)$ is equal to $\frac{3}{2}b^2 + \frac{1}{2}\sum_{j=1}^{n} a_j^2$, then there exists a subset, A', of A such that $\sum_{i\in A'} a_i = b$, i.e., the answer to the Partition Problem is 'Yes,' and S is a witness. If the optimum solution to $1| \,|(\sum_S w_jC_j + \sum_{\bar{S}} e_j)$ is greater than $\frac{3}{2}b^2 + \sum_{j=1}^{n} a_j^2$, then there does not exist a partition of A such that $\sum_{i\in A'} a_i = b$, i.e., the answer to the Partition Problem is 'No.' Conversely, if the answer to the Partition Problem is 'Yes,' the optimum solution to $1| \,|(\sum_S w_jC_j + \sum_{\bar{S}} e_j)$ is clearly equal to $\frac{3}{2}b^2 + \frac{1}{2}\sum_{j=1}^{n} a_j^2$. If the answer to the Partition Problem is 'No,' the optimum solution to $1| \,|(\sum_S w_jC_j + \sum_{\bar{S}} e_j)$ is clearly greater than $\frac{3}{2}b^2 + \frac{1}{2}\sum_{j=1}^{n} a_j^2$. ∎

2.3 Pseudo-polynomial Time Algorithms.

In this section, we give pseudo-polynomial time algorithms for solving $1| \,|(\sum_S w_jC_j + \sum_{\bar{S}} e_j)$ and $Qm| \,|(\sum_S w_jC_j + \sum_{\bar{S}} e_j)$ exactly. We first give an $O(n\sum_{j=1}^{n} w_j)$ time algorithm (in Section 2.3.1) and then an $O(n\sum_{j=1}^{n} p_j)$ time algorithm (in Section 2.3.2), using dynamic programming, to solve $1| \,|(\sum_S w_jC_j + \sum_{\bar{S}} e_j)$. The first runs in polynomial time when the weights are polynomially bounded and the processing times are arbitrary, while the second runs in polynomial time when the processing times are polynomially bounded and the weights are arbitrary. We then generalize our dynamic programs in Section 2.3.3 to any fixed number of uniform parallel machines and solve $Qm| |(\sum_S w_jC_j + \sum_{\bar{S}} e_j)$. In Section 2.4, we show how to modify the dynamic program of Section 2.3.2 to obtain an FPTAS for $1| |(\sum_S w_jC_j + \sum_{\bar{S}} e_j)$. We also generalize this FPTAS to any fixed number of uniform parallel machines and solve $Qm| |(\sum_S w_jC_j + \sum_{\bar{S}} e_j)$.

2.3.1 Dynamic Programming on the Weights w_j.

To solve our problem, we set up a dynamic program for a harder problem: namely, to find the schedule that minimizes the objective function when the total weight of the scheduled jobs is given. We number the jobs in non-decreasing order of p_j/w_j. This is because for any given set S of scheduled jobs, the ordering given by Smith's rule minimizes $\sum_{j\in S} w_jC_j$. Let $\phi_{w,j}$ denote the optimal value of the objective function when the jobs in consideration are $j, j+1, \ldots, n$, and the total weight of the scheduled jobs is w. Note that

$$\phi_{w,n} = \begin{cases} w_np_n & \text{if } w = w_n \\ e_n & \text{if } w = 0 \\ \infty & \text{otherwise} \end{cases} \tag{2.1}$$

This forms the boundary conditions for the dynamic program.

Now, consider any optimal schedule for the jobs $j, j+1, \ldots, n$ in which the total weight of the scheduled jobs is w. In any such schedule, there are two possible cases — either job j is rejected or job j is scheduled.

Case 1: Job j is rejected. Then, the optimal value of the objective function is clearly $\phi_{w,j+1} + e_j$, since the total weight of the scheduled jobs among $j+1, \ldots, n$ must be w.

Case 2: Job j is scheduled. This is possible only if $w \geq w_j$. Otherwise, there is no feasible schedule in which the sum of the weights of the scheduled jobs is w and job j is scheduled, in which case only Case 1 applies. Hence, assume that

$w \geq w_j$. In this case, the total weight of the scheduled jobs among $j+1, \ldots, n$ must be $w - w_j$. Also, when job j is scheduled before all jobs in the optimal schedule for jobs $j+1, j+2, \ldots, n$, the completion time of every scheduled job among $j+1, j+2, \ldots, n$ is increased by p_j. Then, the optimal value of the objective function is clearly $\phi_{w-w_j,j+1} + wp_j$.

Combining the above two cases, we have:

$$\phi_{w,j} = \begin{cases} \phi_{w,j+1} + e_j & \text{if } w < w_j \\ \min(\phi_{w,j+1} + e_j, \phi_{w-w_j,j+1} + wp_j) & \text{otherwise} \end{cases} \tag{2.2}$$

Now, observe that the weight of the scheduled jobs can be at most $\sum_{j=1}^{n} w_j$, and the answer to our original problem is $\min\{\phi_{w,1} \mid 0 \leq w \leq \sum_{j=1}^{n} w_j\}$. Thus, we need to compute at most $n \sum_{j=1}^{n} w_j$ values $\phi_{w,j}$. Computation of each such value takes $O(1)$ time, so that the running time of the algorithm $O(n \sum_{j=1}^{n} w_j)$.

Theorem 2.3.1 *Dynamic programming yields an $O(n \sum_{j=1}^{n} w_j)$ time algorithm for exactly solving $1| |(\sum_S w_j C_j + \sum_{\bar{S}} e_j)$.*

2.3.2 Dynamic Programming on the Processing Times p_j.

In this section, we give another dynamic program that solves $1| |(\sum_S w_j C_j + \sum_{\bar{S}} e_j)$ in $O(n \sum_{j=1}^{n} p_j)$ time.

As before, we set up a dynamic program for a harder problem: namely, to find the schedule that minimizes the objective function when the total processing time of the scheduled jobs, i.e., the makespan of the schedule is given. We number the jobs in non-decreasing order of p_j/w_j (as in Section 2.3.1). Let $\phi_{p,j}$ denote the optimal value of the objective function when the jobs in consideration are $1, 2, \ldots, j$, and the makespan of the schedule is p. Observe that in this case, we are considering the jobs $1, 2, \ldots, j$, which is in contrast to the dynamic program of the previous section where we considered the jobs $j, j+1, \ldots, n$. The boundary conditions of this dynamic program are given by

$$\phi_{p,1} = \begin{cases} w_1 p_1 & \text{if } p = p_1 \\ e_1 & \text{if } p = 0 \\ \infty & \text{otherwise} \end{cases} \tag{2.3}$$

Now, consider any optimal schedule for the jobs $1, 2, \ldots, j$ in which the total processing time of the scheduled jobs is p. In any such schedule, there are two possible cases — either job j is rejected or job j is scheduled.

Case 1: Job j is rejected. Then, the optimal value of the objective function is clearly $\phi_{p,j-1} + e_j$, since the total processing time of the scheduled jobs among $1, 2, \ldots, j-1$ must be p.

Case 2: Job j is scheduled. This is possible only if $p \geq p_j$. Otherwise, there is no feasible schedule in which the makespan is p and job j is scheduled. Hence, assume that $p \geq p_j$. In this case, the completion time of job j is p, and the total processing time of the scheduled jobs among $1, 2, \ldots, j-1$ must be $p - p_j$. Then, the optimal value of the objective function is clearly $\phi_{p-p_j,j-1} + w_j p$.

Combining the above two cases, we have:

$$\phi_{p,j} = \begin{cases} \phi_{p,j-1} + e_j & \text{if } p < p_j \\ \min(\phi_{p,j-1} + e_j, \phi_{p-p_j,j-1} + w_j p) & \text{otherwise} \end{cases} \tag{2.4}$$

Now, observe that the total processing time of the scheduled jobs can be at most $\sum_{j=1}^{n} p_j$, and the answer to our original problem is $\min\{\phi_{p,n} \mid 0 \leq p \leq \sum_{j=1}^{n} p_j\}$. Thus, we need to compute at most $n \sum_{j=1}^{n} p_j$ values $\phi_{p,j}$. Computation of each such value takes $O(1)$ time, so that the running time of the algorithm $O(n \sum_{j=1}^{n} p_j)$.

Theorem 2.3.2 *Dynamic programming yields an $O(n \sum_{j=1}^{n} p_j)$ time algorithm for exactly solving $1| |(\sum_S w_j C_j + \sum_{\bar{S}} e_j)$.*

2.3.3 Generalization to any fixed number of Uniform Parallel Machines.

In this section, we generalize the dynamic program of Section 2.3.1 to any fixed number m of uniform parallel machines and solve $Qm|\ |(\sum_S w_jC_j + \sum_{\bar{S}} e_j)$. The dynamic program of Section 2.3.2 can also be generalized in a similar manner. For uniform parallel machines, the processing time p_{ij} of job i on machine j is given by $p_{ij} = p_i/s_j$, where p_i is the size of job i and s_j is the speed of machine j.

We set up a dynamic program for the problem of finding the schedule that minimizes the objective function when the total weight of the scheduled jobs on *each machine* is given. We number the jobs in non-decreasing order of p_j/w_j (as in Section 2.3.1). Note that for any given machine k, this orders the jobs in increasing order of p_{jk}/w_j. This is useful because given the set of jobs scheduled on each machine, Smith's rule still applies to the jobs scheduled on each machine as far as minimizing $\sum_j w_jC_j$ is concerned. This is where we make use of the fact that the parallel machines are uniform.

Let $\phi_{\omega_1,\omega_2,\ldots,\omega_m,j}$ denote the optimal value of the objective function when the jobs in consideration are $j, j+1, \ldots, n$, and the total weight of the scheduled jobs on machine k is ω_k for all $1 \leq k \leq m$. Note that

$$\phi_{\omega_1,\omega_2,\ldots,\omega_m,n} = \begin{cases} w_n p_{nk} & \text{if } \exists\ k \text{ such that } \omega_k = w_n \text{ and } \omega_k = 0 \text{ for } i \neq k \\ e_n & \text{if } \omega_i = 0 \text{ for all } i \\ \infty & \text{otherwise} \end{cases} \tag{2.5}$$

This forms the boundary conditions for the dynamic program.

Now, consider any optimal schedule for the jobs $j, j+1, \ldots, n$ in which the total weight of the scheduled jobs on machine k is ω_k for all $1 \leq k \leq m$. In any such schedule, there are two possible cases — either job j is rejected or job j is scheduled.

Case 1: Job j is rejected. Then, the optimal value of the objective function is clearly $\phi_{\omega_1,\omega_2,\ldots,\omega_m,j+1}+e_j$, since the total weight of the scheduled jobs among $j+1,\ldots,n$ must be ω_k on machine k.

Case 2: Job j is scheduled. Suppose job j is scheduled on machine k. This is possible only if $\omega_k \geq w_j$. Otherwise, there is no feasible solution in which the sum of the weights of the jobs scheduled on machine k is ω_k and job j is scheduled on machine k. Hence, assume that $\omega_k \geq w_j$. In this case, the total weight of the scheduled jobs among $j+1,\ldots,n$ must be ω_i on machine i for $i \neq k$ and $\omega_k - w_j$ on machine k. Also, when job j is scheduled on machine k before all jobs in the optimal schedule on that machine, the completion time of every scheduled job on that machine is increased by p_{jk}. Then, the optimal value of the objective function is clearly $\phi_{\omega_1,\ldots,\omega_k-w_j,\ldots,\omega_m,j+1} + \omega_k p_{jk}$.

Combining the above two cases, we have:

$$\begin{aligned}\phi_{\omega_1,\omega_2,\ldots,\omega_m,j} = \min(&\phi_{\omega_1,\omega_2,\ldots,\omega_m,j-1} + e_j,\\ &\min\{(\phi_{\omega_1,\ldots,\omega_k-w_j,\ldots,\omega_m,j-1} + \omega_k p_{jk}) \mid \omega_k \geq w_j \text{ and } 1 \leq k \leq m\})\end{aligned}$$

Now, observe that the total weight of the scheduled jobs on any machine can be at most $\sum_{j=1}^{n} w_j$, and the answer to our original problem is $\min\{\phi_{\omega_1,\omega_2,\ldots,\omega_m,1} \mid 0 \leq \omega_i \leq \sum_{j=1}^{n} w_j \text{ and } 1 \leq i \leq m\}$. Thus, we need to compute at most $n(\sum_{j=1}^{n} w_j)^m$ values $\phi_{\omega_1,\omega_2,\ldots,\omega_m,j}$. Computation of each such value takes $O(m)$ time, so that the running time of the algorithm is $O(nm(\sum_{j=1}^{n} w_j)^m)$.

Theorem 2.3.3 *Dynamic programming yields an $O(nm(\sum_{j=1}^{n} w_j)^m)$ time algorithm for exactly solving $Qm| \,|(\sum_S w_jC_j + \sum_{\bar{S}} e_j)$.*

The dynamic program of Section 2.3.2 can be generalized to any fixed number m of uniform parallel machines in a similar manner. The result is summarized in the following theorem.

Theorem 2.3.4 *Dynamic programming yields an $O(nm\prod_{j=1}^{m}(\sum_{i=1}^{n} p_i/s_j))$ time algorithm for exactly solving $Qm|\ |(\sum_S w_jC_j + \sum_{\bar{S}} e_j)$.*

2.4 Fully Polynomial Time Approximation Scheme.

In this section, we describe a fully polynomial time approximation scheme (FPTAS) for sum of weighted completion times with rejection. We first introduce the concept of *aligned schedules* in Section 2.4.1. In Section 2.4.2, we develop an FPTAS for $1|\ |$-$(\sum_S w_jC_j + \sum_{\bar{S}} e_j)$, using the idea of an aligned schedule. The algorithm runs in time polynomial in n, $\frac{1}{\epsilon}$, and the size (number of bits) of the processing times of the jobs. We then generalize this FPTAS to any fixed number of uniform parallel machines in Section 2.4.3 and solve $Qm|\ |(\sum_S w_jC_j + \sum_{\bar{S}} e_j)$.

2.4.1 Aligned Schedules.

We "trim" the state space of the dynamic program of Section 2.3.2 by fusing states that are "close" to each other. This fusion of "close" states is achieved by constraining the jobs in any schedule to finish at times of the form $\tau_i = (1+\epsilon')^i$, for $i \geq 0$, and $\epsilon' > 0$. We call such schedules *ϵ'-aligned schedules.* Note that an ϵ'-aligned schedule may contain idle time. We will handle the zero completion time of the empty schedule by defining $\tau_{-1} = 0$.

We transform any given schedule Γ to an ϵ'-aligned schedule by sliding the scheduled time of each job (starting from the first scheduled job and proceeding in order) forward in time until its completion time coincides with the next time instant of the form τ_i. The job is then said to be *ϵ'-aligned.* Note that when we slide the scheduled time of job i, the scheduled times of later jobs also get shifted forward in time by the same amount as for job i. When the time comes to ϵ'-align job j, its completion time has already moved forward in time by an amount equal to the sum of the amounts moved by the completion times of the jobs scheduled earlier than itself. This identification of a schedule with an ϵ'-aligned schedule gives us the notion of "closeness" of two schedules, i.e., two schedules are "close" if the same jobs are scheduled in both, and they have the same "closest" ϵ'-aligned schedule. We use the term *geometric rounding* to refer to this technique of transforming any schedule to an ϵ'-aligned schedule because the τ_i's form a geometric progression and the completion time of every scheduled job is, in some sense, rounded to one of the τ_i's.

The following lemma establishes an upper bound on the increase in the optimal objective function value when we restrict our attention to ϵ'-aligned schedules only. Without any loss of generality, we can assume that the smallest processing time is at least 1. Otherwise, we can divide each processing time p_j and rejection penalty e_j by the smallest processing time $p < 1$. This increases the size of each processing time and rejection penalty by $\log \frac{1}{p} = -\log p$ bits, and hence the running time of the algorithm (see Theorem 2.4.3) by at most a polynomial additive factor of $\frac{n^2}{\epsilon} \log \frac{1}{p}$.

Lemma 2.4.1 *For any given schedule Γ with n jobs and any $\epsilon' > 0$, the completion time of each job in Γ increases by a factor of at most $(1+\epsilon')^n$ when we ϵ'-align schedule Γ.*

Proof. We establish, by induction on i, that in any given schedule, the completion time C_i of the i^{th} scheduled job increases by a factor of at most $(1+\epsilon')^i$ after the schedule is ϵ'-aligned. Note that if a job finishes at time $t \in (\tau_{i-1}, \tau_i]$ after all the jobs before it have been ϵ'-aligned, its completion time after being ϵ'-aligned is $\tau_i < (1+\epsilon')t$. Clearly, C_1 increases by a factor of at most $(1+\epsilon')$, since it is made to coincide with the right end-point of the interval $(\tau_{\ell-1}, \tau_\ell]$ it is initially contained in. Note that the smallest τ_i is $\tau_0 = 1$, so that we made use here of the assumption that the smallest processing time is at least 1.

Now, assume by induction, that C_i increases by a factor of at most $(1+\epsilon')^i$. When the turn comes to ϵ'-align the $(i+1)^{th}$ scheduled job, its completion time C_{i+1} has already increased by an amount equal to the amount moved by C_i, which is at most $[(1+\epsilon')^i - 1]C_i$. When it is ϵ'-aligned, C_{i+1} *further* increases by a factor of at most

$(1+\epsilon')$. Thus, the final value of C_{i+1} is at most $(1+\epsilon')([(1+\epsilon')^i - 1]C_i + C_{i+1})$. Since $C_i < C_{i+1}$, this is at most $(1+\epsilon')^{i+1} C_{i+1}$.

Since the number of scheduled jobs is at most n, the result follows. ■

Setting $\epsilon' = \frac{\epsilon}{2n}$ gives us an $(1+\epsilon)$-factor increase in the optimal objective function value, as stated in the next lemma.

Lemma 2.4.2 *For* $1|\,|(\sum_S w_j C_j + \sum_{\bar{S}} e_j)$ *and* $\epsilon' = \epsilon/2n$ *for any* $\epsilon > 0$*, the optimal objective function value increases by a factor of at most* $(1+\epsilon)$ *when we restrict our attention to* ϵ'*-aligned schedules only.*

Proof. We first establish the inequality $(1 + \frac{x}{m})^m \leq 1 + 2x$, which holds for any $0 \leq x \leq 1$, and any real $m \geq 1$. The left-hand side of the inequality is a convex function in x, and the right-hand side is a linear function in x. Moreover, the inequality holds at $x = 0$ and $x = 1$. Hence, it holds for any $0 \leq x \leq 1$. Setting $x = \frac{\epsilon}{2}$ and $m = n$, we have

$$(1 + \frac{\epsilon}{2n})^n \leq (1+\epsilon)$$

Using Lemma 2.4.1 and the above inequality, we conclude that in any given schedule, the completion time of a job increases by a factor of at most $(1+\epsilon)$ after the schedule is ϵ'-aligned. Thus, the sum of weighted completion times increases by a factor of at most $(1+\epsilon)$ after the schedule is ϵ'-aligned.

Now, consider an optimal schedule Γ in which the set of scheduled jobs is S. The quantity $\sum_{j \in S} w_j C_j$ for the schedule Γ increases by a factor of at most $(1+\epsilon)$ after the schedule Γ is ϵ'-aligned. Also, the total rejection penalty of the jobs in $\bar{S}$ trivially remains unchanged. Hence, the objective function value for schedule Γ increases by a factor of at most $(1+\epsilon)$ when Γ is ϵ'-aligned. This implies that the optimal objective function value increases by a factor of at most $(1+\epsilon)$ when we restrict our attention to ϵ'-aligned schedules only. ■

2.4.2 The Algorithm.

Let $\epsilon' = \frac{\epsilon}{2n}$. For our FPTAS, we set up a dynamic program for a harder problem: namely, to find the ϵ'-aligned schedule that minimizes the objective function when the completion time of the latest scheduled job is τ_i, for a given i. We number the jobs in ascending order of p_j/w_j (as in Section 2.3.2). Let $\phi_{i,j}$ denote the optimal value of the objective function when the jobs in consideration are $1, 2, \ldots, j$, and the latest scheduled job (if any) in an ϵ'-aligned schedule completes at time τ_i for $i \geq 0$. Note that

$$\phi_{i,1} = \begin{cases} w_1 \tau_i & \text{if } p_1 \in (\tau_{i-1}, \tau_i] \\ e_1 & \text{if } i = -1 \\ \infty & \text{otherwise} \end{cases} \tag{2.6}$$

This forms the boundary conditions for the dynamic program.

Now, consider any optimal ϵ'-aligned schedule for the jobs $1, 2, \ldots, j, j+1$ in which the latest scheduled job completes at time τ_i. In any such schedule, there are two possible cases — either job $j+1$ is rejected or job $j+1$ is scheduled.

Case 1: Job $j+1$ is rejected. Then, the optimal value of the objective function is clearly $\phi_{i,j} + e_{j+1}$, since the last of the scheduled jobs among $1, 2, \ldots, j$ must finish at time τ_i.

Case 2: Job $j+1$ is scheduled. This is possible only if $p_{j+1} \leq \tau_i$. Otherwise, there is no feasible ϵ'-aligned schedule whose completion time is τ_i and in which job $j+1$ is scheduled, in which case only Case 1 applies. Hence, assume that $p_{j+1} \leq \tau_i$. In this case, if there was a job scheduled before job $j+1$, it must have completed at time $\tau_{i'}$, where i' is the greatest value of ℓ satisfying $(\tau_i - \tau_\ell) \geq p_{j+1}$. Then, the optimal value of the objective function is clearly $\phi_{i',j} + w_{j+1}\tau_i$.

Combining the above two cases, we have:

$$\phi_{i,j+1} = \begin{cases} \phi_{i,j} + e_{j+1} & \text{if } p_{j+1} > \tau_i \\ \min(\phi_{i,j} + e_{j+1}, \phi_{i',j} + w_{j+1}\tau_i) & \text{otherwise} \end{cases} \tag{2.7}$$

Now, observe that for finding an ϵ'-aligned schedule with the optimum objective function value, it is sufficient to assume that the completion time of the latest scheduled job is at most $(1+\epsilon')^n \sum_{j=1}^n p_j$. The answer to our original problem is $\min\{\phi_{i,n} \mid -1 \leq i \leq L\}$, where L is the smallest integer such that $\tau_L \geq (1+\epsilon')^n \sum_{j=1}^n p_j$. Thus, L is the smallest integer greater than or equal to $\frac{\log \sum_{j=1}^n p_j}{\log(1+\epsilon')} + n$, whence $L = O(\frac{n}{\epsilon} \log \sum_{j=1}^n p_j)$. Thus, we need to compute at most $n(L+2)$ values $\phi_{i,j}$. Computation of each such value takes $O(1)$ time, so that the overall time for the dynamic program (FPTAS) is $O(nL) = O(\frac{n^2}{\epsilon} \log \sum_{j=1}^n p_j)$. This is polynomial in the input size, since we need $\sum_{j=1}^n \log p_j$ bits to represent the processing times. We also note that dividing each processing time and rejection penalty by the smallest processing time $p < 1$ increases the running time of the algorithm by at most a polynomial additive factor of $\frac{n^2}{\epsilon} \log \frac{1}{p}$.

Theorem 2.4.3 *Dynamic programming yields an $(1+\epsilon)$-factor fully polynomial time approximation scheme (FPTAS) for $1| |(\sum_S w_j C_j + \sum_{\bar{S}} e_j)$, which runs in $O(\frac{n^2}{\epsilon} \log \sum_{j=1}^n p_j)$ time.*

2.4.3 Generalization to any fixed number of Uniform Parallel Machines.

In this section, we generalize the dynamic program of Section 2.4.2 to any fixed number m of uniform parallel machines and obtain an FPTAS for $Qm| |(\sum_S w_j C_j + \sum_{\bar{S}} e_j)$. Let p_j denote the size of job j and s_i the speed of machine i for $1 \leq j \leq n$ and $1 \leq i \leq m$ in the uniform parallel machine model. Then, the processing time of job j on machine i is $p_{ji} = p_j / s_i$.

For our generalized FPTAS, we set up a dynamic program for the following problem: namely, to find the ϵ'-aligned schedule that minimizes the objective function when the completion time of the latest scheduled job on machine k is τ_{i_k} for $1 \leq k \leq m$. We number the jobs in ascending order of p_j/w_j (as in Section 2.3.2). Note that in the uniform parallel machine model, this numbers the jobs in increasing order of p_{jk}/w_j on any machine k, so that the jobs still get scheduled according to Smith's rule on any given machine. This is where we make use of the fact that the parallel machines are uniform.

Let $\phi_{i_1,i_2,\ldots,i_m,j}$ denote the optimal value of the objective function when the jobs in consideration are $1, 2, \ldots, j$, and the latest scheduled job (if any) in an ϵ'-aligned schedule on machine k completes at time τ_{i_k} for $1 \leq k \leq m$. Note that

$$\phi_{i_1,i_2,\ldots,i_m,1} = \begin{cases} w_1\tau_{i_k} & \text{if } \exists\ k \text{ such that } p_{1k} \in (\tau_{i_k-1}, \tau_{i_k}] \text{ and } i_\ell = -1\ \forall\ \ell \neq k \\ e_1 & \text{if } i_\ell = -1\ \forall\ \ell \\ \infty & \text{otherwise} \end{cases} \tag{2.8}$$

This forms the boundary conditions for the dynamic program.

Now, consider any optimal ϵ'-aligned schedule for the jobs $1, 2, \ldots, j, j+1$ in which the latest scheduled job on machine k completes at time τ_{i_k} for $1 \leq k \leq m$. In any such schedule, there are two possible cases — either job $j+1$ is rejected or job $j+1$ is scheduled.

Case 1: Job $j+1$ is rejected. Then, the optimal value of the objective function is clearly $\phi_{i_1,i_2,\ldots,i_m,j} + e_{j+1}$, since the last of the jobs among $1, 2, \ldots, j$ scheduled on machine k must finish at time τ_{i_k}.

Case 2: Job $j+1$ is scheduled. Suppose job $j+1$ is scheduled on machine k. This is possible only if $p_{j+1,k} \leq \tau_{i_k}$. Otherwise, there is no feasible ϵ'-aligned schedule whose completion time on machine k is τ_{i_k} and in which job $j+1$ is scheduled on machine k. Hence, assume that $p_{j+1,k} \leq \tau_i$. In this case, if there was a job scheduled before job $j+1$ on machine k, it must have completed at time $\tau_{i'_k}$, where i'_k is the greatest value of ℓ satisfying $(\tau_{i_k} - \tau_\ell) \geq p_{j+1,k}$. Then, the optimal value of the objective function is clearly $\phi_{i_1,\ldots,i'_k,\ldots,i_m,j} + w_{j+1}\tau_{i_k}$.

Combining the above two cases, we have:

$$\phi_{i_1,i_2,\ldots,i_m,j+1} = \min(\phi_{i_1,i_2,\ldots,i_m,j} + e_{j+1},$$
$$\min\{(\phi_{i_1,\ldots,i'_k,\ldots,i_m,j} + w_{j+1}\tau_{i_k}) \mid p_{j+1,k} \le \tau_{i_k} \text{ and } 1 \le k \le m\})$$

Now, observe that for finding an ϵ'-aligned schedule with the optimum objective function value, it is sufficient to assume that the completion time of the latest scheduled job on machine i is at most $(1+\epsilon')^n \sum_{j=1}^n p_{ji} = (1+\epsilon')^n \sum_{j=1}^n p_j/s_i$ for $1 \le i \le m$. Hence, the largest value of i_k, $1 \le k \le m$, for which we need to compute $\phi_{i_1,i_2,\ldots,i_m,j}$ is L_k, where L_k is the smallest integer such that $\tau_{L_k} \ge (1+\epsilon')^n \sum_{j=1}^n p_j/s_k$. Thus, L_k is the smallest integer greater than or equal to $\frac{\log(\sum_{j=1}^n p_j/s_k)}{\log(1+\epsilon')} + n$, whence $L_k = O(\frac{n}{\epsilon}\log(\sum_{j=1}^n p_j/s_k))$.

The answer to our original problem is

$$\min\{\phi_{i_1,i_2,\ldots,i_m,n} \mid -1 \le i_k \le L_k \text{ and } 1 \le k \le m\}$$

Thus, we need to compute at most $n(L_1+2)(L_2+2)\cdots(L_m+2)$ values $\phi_{i_1,i_2,\ldots,i_m,j}$. Computation of each such value takes $O(m)$ time, so that the overall time for the dynamic program (FPTAS) is $O(nmL_1L_2\cdots L_m) = O(\frac{n^{m+1}m}{\epsilon^m}\prod_{i=1}^m(\log(\sum_{j=1}^n p_j/s_i)))$. This is polynomial in the input size for a fixed m, since we need $\sum_{j=1}^n \log p_j$ bits to represent the job sizes.

Theorem 2.4.4 *Dynamic programming yields an $(1+\epsilon)$-factor fully polynomial time approximation scheme (FPTAS) for $Qm| |(\sum_S w_jC_j + \sum_{\bar{S}} e_j)$, which runs in $O(\frac{n^{m+1}m}{\epsilon^m}\prod_{i=1}^m(\log(\sum_{j=1}^n p_j/s_i)))$ time.*

2.5 Special Cases.

In this section, we consider two special cases for which simple greedy algorithms exist to solve $1| |(\sum_S w_jC_j + \sum_{\bar{S}} e_j)$. In Section 2.5.1, we give an $O(n^2)$ time algorithm for the case when all weights are equal, i.e., for $1|w_j = w|(\sum_S w_jC_j + \sum_{\bar{S}} e_j)$. This algorithm also works for the case when all processing times are equal, i.e., for $1|p_j = p|(\sum_S w_jC_j+\sum_{\bar{S}} e_j)$. In Section 2.5.2, we define the concept of compatible processing times, weights, and rejection costs and then give an $O(n \log n)$ time algorithm for this case.

2.5.1 Equal Weights or Equal Processing Times.

In this section, we give a simple and efficient $O(n^2)$ time algorithm for exactly solving $1| |(\sum_S w_jC_j + \sum_{\bar{S}} e_j)$ when all weights or all processing times are equal.

Our greedy algorithm, which we call SCHREJ, is as follows. We start with all jobs scheduled, i.e., the set of scheduled jobs is $S = \{1, 2, \ldots, n\}$. Note that we can optimally schedule the jobs in any subset S using Smith's ordering. We then reject jobs greedily until we arrive at an optimal schedule. Observe that when we reject a previously scheduled job j, there is a change (positive or negative) in the objective function. We determine a job k that causes the maximum decrease in the objective function. If there is no such job, (i.e., each job, on being rejected, increases the value of the objective function), then (as we will argue below) we have reached an optimal solution. Otherwise, we remove job k from S (i.e., reject it), and iterate on the remaining jobs in S. We describe the algorithm formally below.

```
Algorithm SCHREJ:

Sort the jobs in increasing order of $p_j/w_j$ ;
Label them in sorted order from 1 to $n$ ;
$S \leftarrow \{1, 2, \ldots, n\}$ ;
$\bar{S} \leftarrow \emptyset$ ;
for $j = 1$ to $n$ do
    $\Delta_j \leftarrow -[w_j \sum_{1 \leq i \leq j} p_i + p_j \sum_{j < i \leq n} w_i] + e_j$ ;
repeat {
    $\Delta_k \leftarrow \min\{\Delta_j : j \in S\}$ ; /* this defines k */
    if $(\Delta_k < 0)$ then {
            /* reject job $k$ and solve the problem for jobs in $S - \{k\}$ */
            $S \leftarrow S - \{k\}$ ;
            $\bar{S} \leftarrow \bar{S} \cup \{k\}$ ;
            /* update the $\Delta_j$ values */
            for $j \in S$ do
                if $(j < k)$ then
                    $\Delta_j \leftarrow \Delta_j + p_j w_k$ ;
                else
                    $\Delta_j \leftarrow \Delta_j + w_j p_k$ ;
    }
} until $(\Delta_k \geq 0)$ or $(S = \emptyset)$ ;
Output the schedule in which jobs in $S$ are scheduled in increasing
order of processing times $p_j$, and jobs in $\bar{S}$ are rejected ;
```

As this algorithms runs, we maintain the set S of currently scheduled jobs. During an iteration of the repeat-loop, we consider only the jobs in the set S. The quantity Δ_j is the change in the objective function value (or, the cost of our schedule) when we (pay to) reject job j from the schedule in which all jobs in S are scheduled. For notational convenience, we will denote this by $\Delta_j(S)$. Note that Δ_j as used in the algorithm is the same as $\Delta_j(S)$. Let $k \in S$ be such that $\Delta_k(S)$ is minimum. We first derive an expression for $\Delta_j(S)$.

Lemma 2.5.1 *During any iteration of* SCHREJ, *for any $j \in S$, we have*

$$\Delta_j(S) = -[w_j \sum_{i \leq j, i \in S} p_i \; + \; p_j \sum_{i > j, i \in S} w_i] + e_j$$

Proof. The first term on the right hand side is equal to $-w_j C_j$, where C_j is the compeltion time of job j in the optimal schedule when all jobs in S are scheduled. Note that when we reject job j, the completion time of each job scheduled after job j is reduced by p_j. Since the completion times are weighted by the weights of the jobs, there is a further decrease of $p_j \sum_{i>j, i \in S} w_i$ in the objective function value. Finally, there is an increase equal to the penalty e_j in rejecting job j. ∎

Note that we initialize the Δ_j's in accordance with this expression before entering the repeat-loop. When we reject a job k inside the repeat-loop i.e. remove it from S, this expression allows us to update the $\Delta_j(S)$ to $\Delta_j(S - \{k\})$ in $O(1)$ time for each $j \in S - \{k\}$. That is,

$$\Delta_j(S - \{k\}) = \begin{cases} \Delta_j(S) + p_j w_k & \text{if } j < k \\ \Delta_j(S) + w_j p_k & \text{otherwise} \end{cases} \tag{2.9}$$

Since every iteration of the repeat-loop rejects one job (except, possibly, the last iteration) and there are n jobs initially in S, the repeat-loop runs for at most n iterations. Each iteration takes $O(n)$ time to compute k and update the Δ_j's. Hence, the running time of the entire loop is $O(n^2)$. Also, the initial sorting of the jobs takes $O(n \log n)$ time and the initialization of the Δ_j's takes $O(n^2)$ time, so that algorithm SCHREJ has running time $O(n^2)$.

For the proof, we label the jobs in increasing order of p_j/w_j. For the case in which the w_j's are equal, this labels the processing times in increasing order, so that $p_1 \leq p_2 \leq \cdots \leq p_n$.

The following lemma is an immediate consequence of Lemma 2.5.1 and the update rule for the $\Delta_j(S)$'s.

Lemma 2.5.2 *The value of $\Delta_j(S)$ increases across every iteration of* SCHREJ*, so long as job j remains in S.*

The following lemma argues that the algorithm can terminate when $\Delta_k(S) \geq 0$.

Lemma 2.5.3 *For a set S of jobs, if $\Delta_j(S)$ is non-negative for each $j \in S$, then there is an optimal schedule in which all the jobs in S are scheduled.*

Proof. Consider any non-empty set of jobs $R \subset S$. Let us start with the schedule in which all jobs in S are scheduled, and start rejecting the jobs in R one by one. Clearly, the first rejection does not decrease the objective function value, since $\Delta_j(S)$ is non-negative for each $j \in S$. Also, on rejecting a job from S, the value of the Δ_j's only increase, so that the same is true upon rejecting further jobs in R. Thus, the objective function value does not decrease upon rejecting the jobs in R. This proves the lemma. ∎

For the above lemmas, we have not used the fact that the weights w_j are equal. Hence, Lemma 2.5.3 serves as a test of termination of the algorithm even when the weights are arbitrary. The fact that the w_j's are equal is used in the next lemma, which proves that we are justified in moving job k from S to $\bar{S}$ (i.e. rejecting job k) when $\Delta_k(S) < 0$.

Lemma 2.5.4 *For a set S of jobs with equal weights, if $\Delta_k(S)$ (the minimum of the $\Delta_j(S)$ for $j \in S$, as computed in* SCHREJ*) is negative, then there is an optimal schedule for the set of jobs in S in which job k is rejected.*

Proof. The proof is by contradiction. Suppose that in every optimal schedule for S, job k is scheduled. Consider any such schedule Γ in which the set of rejected jobs is R. Clearly, R is non-empty, otherwise, since $\Delta_k < 0$, we can get a better schedule by rejecting job k. We show that we can improve the schedule Γ by rejecting job k and instead scheduling one of the jobs in R (the one immediately preceding or following job k according to the ordering given by Smith's rule) to obtain a schedule Γ'. We will compare the objective function value for the schedules Γ and Γ' by starting from a schedule in which all jobs are scheduled, and then rejecting the set of rejected jobs for each schedule in a particular order. Let R_1 be the set of jobs in R which precede job k in job index order, and let R_2 be the set of jobs in R which follow job k in job index order. We will consider two cases, depending on whether the set R_2 is empty or not.

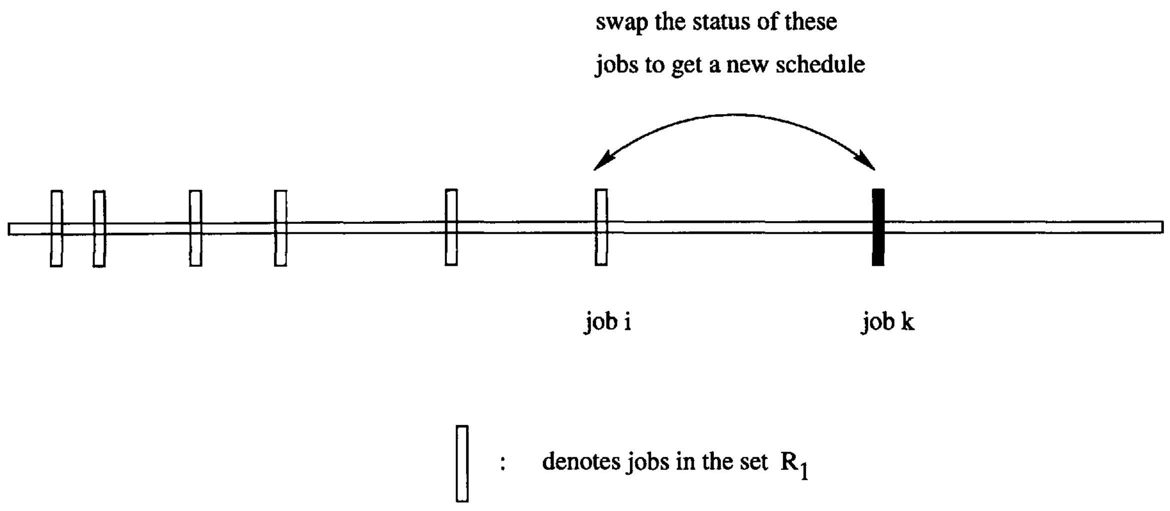

Figure 2-1: Case 1 for Lemma 2.5.4

Case 1: Set R_2 is empty (see Figure 2-1). Let the last job in job index order in R_1 be i ($i < k$). Note that $\Delta_i(S) \geq \Delta_k(S)$. Consider another schedule Γ' in which job k is rejected, job i is scheduled, and the status of other jobs remains unchanged, i.e., we swap the status of jobs k and i in the optimal schedule Γ to get this new schedule Γ'. We compare the objective function value for these two schedules by first rejecting the jobs in $R_1 - \{i\}$ for both schedules, and finally rejecting job i for Γ and job k for Γ'. Note that for the set of jobs $R_1 - \{i\}$ rejected for each schedule, the decrease in objective function value is the same for both Γ' and Γ. At this point, the new value for Δ_i is $\Delta_i' = \Delta_i(S) + w_i \sum_{j \in R_1 - \{i\}} p_j$ and the new value for Δ_k is $\Delta_k' = \Delta_k(S) + w_k \sum_{j \in R_1 - \{i\}} p_j$. Since $w_i = w_k$, we still have $\Delta_i{}' \geq \Delta_k{}'$ (actually, it suffices to have $w_i \geq w_k$ here). Hence, on now rejecting job i for Γ and job k for Γ', the decrease (positive or negative) in the objective function value is greater (or same) for Γ' than Γ. Thus, the final objective function value for schedule Γ' is less than or equal to the final objective function value for schedule Γ, and job k is rejected in schedule Γ'. This is a contradiction.

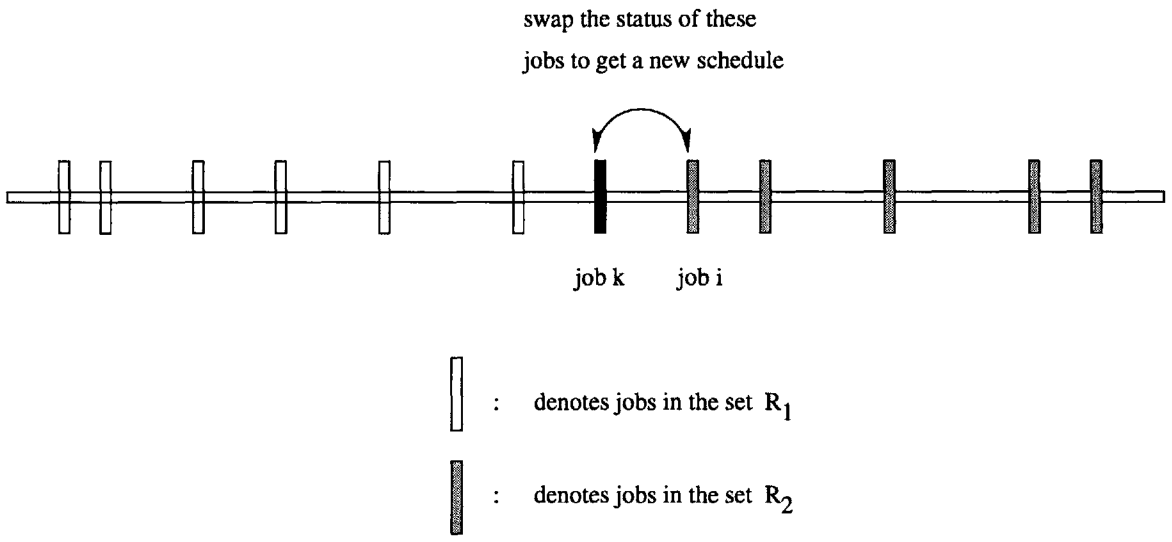
swap the status of these jobs to get a new schedule
job k
job i
: denotes jobs in the set R_1
: denotes jobs in the set R_2

Figure 2-2: Case 2 for Lemma 2.5.4

Case 2: Set R_2 is non-empty (see Figure 2-2). Let the first job in job index order in R_2 be i ($i > k$). Note that $\Delta_i(S) \geq \Delta_k(S)$. Consider another schedule Γ' in which job k is rejected, job i is scheduled, and the status of other jobs remains unchanged, i.e., we swap the status of jobs k and i in the optimal schedule Γ to get this new schedule Γ'. We compare the objective function value for these two schedules by first rejecting the jobs in R_1 for both schedules, then the jobs in $R_2 - \{i\}$ for both schedules, and finally rejecting job i for Γ and job k for Γ'. Note that for the sets of jobs R_1 and $R_2 - \{i\}$ rejected for each schedule, the decrease in objective function value is the same for both Γ' and Γ. At this point, the new value for Δ_i is $\Delta_i{}' = \Delta_i(S) + w_i \sum_{j \in R_1} p_j + p_i \sum_{j \in R_2 - \{i\}} w_j$ and the new value for Δ_k is $\Delta_k{}' = \Delta_k(S) + w_k \sum_{j \in R_1} p_j + p_k \sum_{j \in R_2 - \{i\}} w_j$. Since $w_i = w_k$

and $p_i \geq p_k$, we still have $\Delta_i{}' \geq \Delta_k{}'$ (actually, it suffices to have $w_k \leq w_i$ here). Hence, on now rejecting job i for Γ and job k for Γ', the decrease (positive or negative) in the objective function value is greater (or same) for Γ' than Γ. Thus, the final objective function value for schedule Γ' is less than or equal to the final objective function value for schedule Γ, and job k is rejected in schedule Γ'. This is a contradiction.

Note that we required the w_i's to be in non-increasing order in Case 1 and in non-decreasing order in Case 2 for the proof to carry through. These two together require the w_i's to be equal. ∎

We finally prove the correctness of algorithm SCHREJ.

Theorem 2.5.5 *Algorithm* SCHREJ *outputs an optimal schedule for* $1|w_j = w|(\sum_S w_j C_j + \sum_{\bar{S}} e_j)$ *in* $O(n^2)$ *time.*

Proof. The proof is by induction on the number of jobs. The basic idea of the proof is that algorithm SCHREJ rejects only jobs which are safe to reject, and terminates when all remaining jobs must be in the optimal schedule. Let J denote the set of jobs input to the algorithm, and let $f(J)$ be the optimum objective function value for the jobs in J. By Lemma 2.5.3, the algorithm terminates correctly when $\Delta_j(J) \geq 0$ for each $j \in J$. Now, consider the case when $\Delta_k(J) < 0$, and we reject job k during the first iteration of the repeat loop. By induction on the size of J, assume that the algorithm finds an optimal schedule Γ for $J - \{k\}$. By Lemma 2.5.4, there exists an optimal schedule Γ_{opt} for J in which job k is rejected. Clearly, the set of scheduled jobs in Γ_{opt} is also a valid schedule for $J - \{k\}$, so that $f(J - \{k\}) \leq f(J) - e_k$. Also, the set of scheduled jobs for Γ is a valid schedule for J, so that $f(J) \leq f(J - \{k\}) + e_k$. The above two inequalities imply that $f(J) = f(J - \{k\}) + e_k$. This proves the correctness of algorithm SCHREJ.

It was shown earlier that the running time of the algorithm is $O(n^2)$. ∎

Algorithm SCHREJ also works for the case when all the processing times are equal, i.e., for $1|p_j = p|(\sum_S w_j C_j + \sum_{\bar{S}} e_j)$. The structure of the proof is essentially the same. The only change is in the proof of Lemma 2.5.4. We restate and prove the version of this lemma when all the processing times are equal.

Lemma 2.5.6 *For a set S of jobs with equal processing times, if $\Delta_k(S)$ (the minimum of the $\Delta_j(S)$ for $j \in S$, as computed in* SCHREJ*) is negative, then there is an optimal schedule for the set of jobs in S in which job k is rejected.*

Proof. The proof is by contradiction. Suppose that in every optimal schedule for S, job k is scheduled. Consider any such schedule Γ in which the set of rejected jobs is R. Clearly, R is non-empty, otherwise, since $\Delta_k < 0$, we can get a better schedule by rejecting job k. We show that we can improve the schedule Γ by rejecting job k and instead scheduling one of the jobs in R (the one immediately preceding or following job k according to the ordering given by Smith's rule) to obtain a schedule Γ'. We will compare the objective function value for the schedules Γ and Γ' by starting from a schedule in which all jobs are scheduled, and then rejecting the set of rejected jobs for each schedule in a particular order. Let R_1 be the set of jobs in R which precede job k in job index order, and let R_2 be the set of jobs in R which follow job k in job index order. We will consider two cases, depending on whether the set R_1 is empty or not.

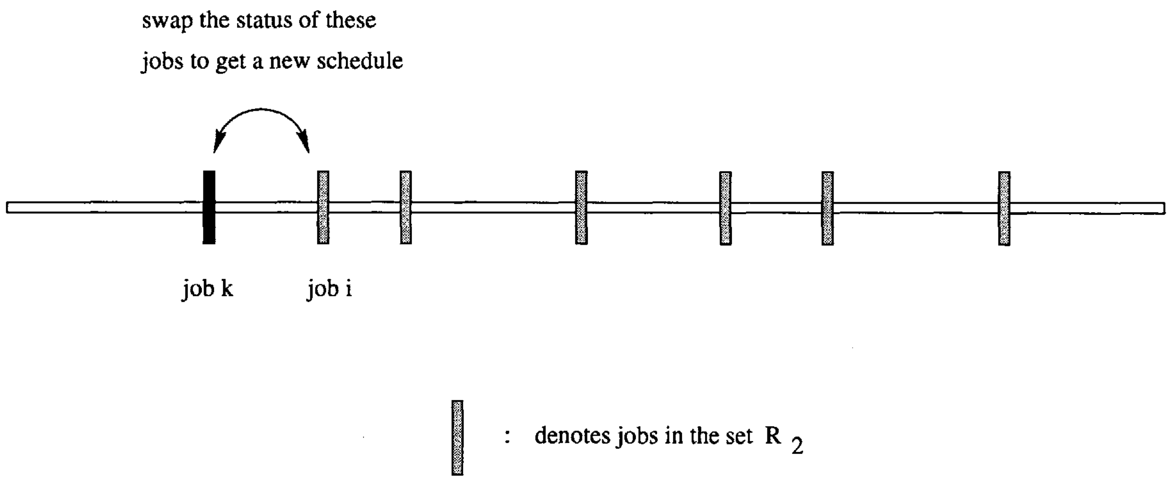

Figure 2-3: Case 1 for Lemma 2.5.6

Case 1: Set R_1 is empty (see Figure 2-3). Let the first job in job index order in R_2 be i $(i < k)$. Note that $\Delta_i(S) \geq \Delta_k(S)$. Consider another schedule Γ' in which job k is rejected, job i is scheduled, and the status of other jobs remains unchanged, i.e., we swap the status of jobs k and i in the optimal schedule Γ to get this new schedule Γ'. We compare the objective function value for these two schedules by first rejecting the jobs in $R_2 - \{i\}$ for both schedules, and finally rejecting job i for Γ and job k for Γ'. Note that for the set of jobs $R_2 - \{i\}$ rejected for each schedule, the decrease in objective function value is the same for both Γ' and Γ. At this point, the new value for Δ_i is $\Delta_i{}' = \Delta_i(S) + p_i \sum_{j \in R_2 - \{i\}} w_j$ and the new value for Δ_k is $\Delta_k{}' = \Delta_k(S) + p_k \sum_{j \in R_2 - \{i\}} w_j$. Since $p_i = p_k$, we still have $\Delta_i{}' \geq \Delta_k{}'$ (actually, it suffices to have $p_i \geq p_k$ here). Hence, on now rejecting job i for Γ and job k for Γ', the decrease (positive or negative) in the objective function value is greater (or same) for Γ' than Γ. Thus, the final objective function value for schedule Γ' is less than or equal to the final objective function value for schedule Γ, and job k is rejected in schedule Γ'. This is a contradiction.

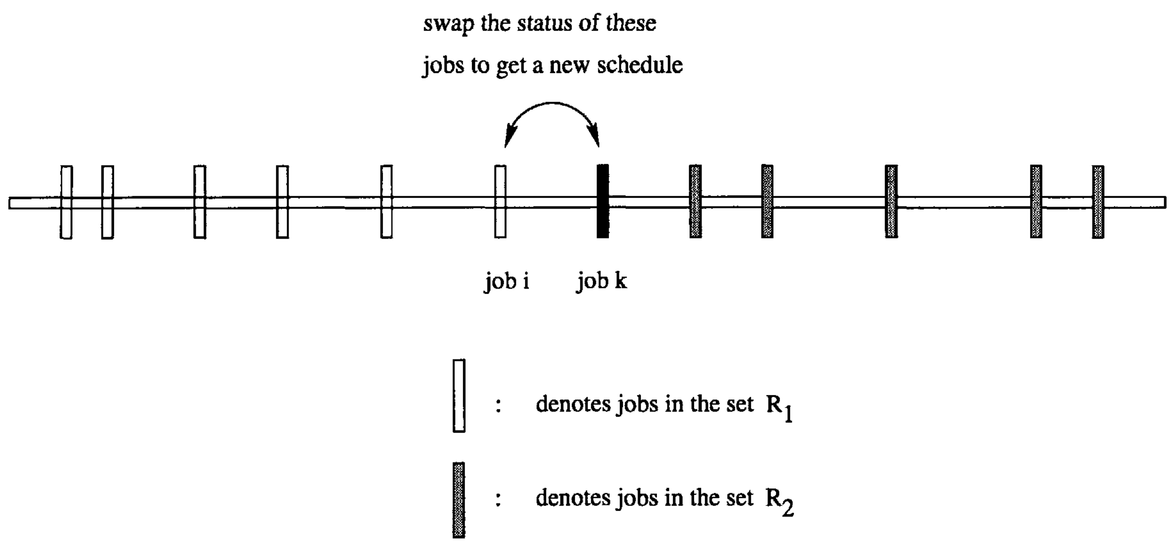
swap the status of these
jobs to get a new schedule
job i
job k
: denotes jobs in the set R1
: denotes jobs in the set R2

Figure 2-4: Case 2 for Lemma 2.5.6

Case 2: Set R_1 is non-empty (see Figure 2-4). Let the last job in job index order in

R_1 be i ($i > k$). Note that $\Delta_i(S) \geq \Delta_k(S)$. Consider another schedule Γ' in which job k is rejected, job i is scheduled, and the status of other jobs remains unchanged, i.e., we swap the status of jobs k and i in the optimal schedule Γ to get this new schedule Γ'. We compare the objective function value for these two schedules by first rejecting the jobs in $R_1 - \{i\}$ for both schedules, then the jobs in R_2 for both schedules, and finally rejecting job i for Γ and job k for Γ'. Note that for the sets of jobs $R_1 - \{i\}$ and R_2 rejected for each schedule, the decrease in objective function value is the same for both Γ' and Γ. At this point, the new value for Δ_i is $\Delta_i{}' = \Delta_i(S) + w_i \sum_{j \in R_1 - \{i\}} p_j + p_i \sum_{j \in R_2} w_j$ and the new value for Δ_k is $\Delta_k{}' = \Delta_k(S) + w_k \sum_{j \in R_1 - \{i\}} p_j + p_k \sum_{j \in R_2} w_j$. Since $p_i = p_k$ and $w_i \geq w_k$, we still have $\Delta_i{}' \geq \Delta_k{}'$ (actually, it suffices to have $p_i \geq p_k$ here). Hence, on now rejecting job i for Γ and job k for Γ', the decrease (positive or negative) in the objective function value is greater (or same) for Γ' than Γ. Thus, the final objective function value for schedule Γ' is less than or equal to the final objective function value for schedule Γ, and job k is rejected in schedule Γ'. This is a contradiction.

Note that we required the p_i's to be in non-decreasing order in Case 1 and in non-increasing order in Case 2 for the proof to carry through. These two together require the p_i's to be equal. ∎

We finally restate Theorem 2.5.5 for the case when all the processing times are equal.

Theorem 2.5.7 *Algorithm* SCHREJ *outputs an optimal schedule for* $1|p_j = p|(\sum_S w_j C_j + \sum_{\bar{S}} e_j)$ *in* $O(n^2)$ *time.*

2.5.2 Compatible Processing Times, Weights, and Rejection Costs.

In this section, we given an $O(n \log n)$ time algorithm for $1||(\sum_S w_j C_j + \sum_{\bar{S}} e_j)$ when the processing times, weights, and the rejection costs are *compatible.* We first define what we mean by the term "compatible".

A set of n jobs is said to have compatible processing times, weights, and rejection costs when an ordering of the jobs according to Smith's rule also orders the jobs in non-decreasing order of both processing times and weights, and in non-increasing order of rejection costs. That is, the jobs can be indexed such that

1. $p_1/w_1 \leq p_2/w_2 \leq \cdots p_n/w_n$,
2. $p_1 \leq p_2 \leq \cdots \leq p_n$,
3. $w_1 \leq w_2 \leq \cdots \leq w_n$, and
4. $e_1 \geq e_2 \geq \cdots \geq e_n$.

We prove that the following simple greedy algorithm determines an optimal schedule in polynomial time. We call the algorithm COMPAT. It first labels the jobs in compatible order. Job i is assigned a completion time of $C_i = \sum_{j=1}^{i} p_j$. The algorithm schedules job i if and only if $w_iC_i \leq e_i$. This makes intuitive sense because job i makes a contribution of w_iC_i to the objective function when it is scheduled and a contribution of e_i to the objective function when it is rejected. The algorithm stops when it has to reject a job.

```
Algorithm COMPAT:

Sort the jobs in compatible order ;
Label them in sorted order from 1 to $n$ ;
$S \leftarrow \emptyset$ ;
for $j = 1$ to $n$ do {
    if ($j = 1$) then
        $C_1 \leftarrow p_1$ ;
    else
        $C_j \leftarrow C_{j-1} + p_j$ ;
    if ($w_j C_j \leq e_j$) then
        /* schedule job $j$ */
        $S \leftarrow S \cup \{j\}$ ;
    else
        /* break out of for loop */
        break ;
}
Output the schedule in which jobs in $S$ are scheduled
in compatible order and all other jobs are rejected ;
```

It is easy to analyze the running time of algorithm COMPAT. Since every iteration of the for-loop takes $O(1)$ time and the for-loop runs for at most n iterations, the total running for the for-loop is $O(n)$. The initial sorting into compatible order takes $O(n \log n)$ time, so that the overall running time for algorithm COMPAT is $O(n \log n)$.

For the rest of this section, we assume that the jobs are labelled in compatible order. Our proof of correctness shows why it makes sense to use $C_i = \sum_{j=1}^{i} p_i$ as the completion time of job i if it is scheduled. This does not seem correct at first glance since it is possible for some jobs among $\{1, 2, \ldots, i-1\}$ to get rejected in the optimal schedule. However, we show that there exists an optimal schedule in which, for some $1 \leq k \leq n$, all the jobs $\{1, 2, \ldots, k\}$ are scheduled and all the jobs $\{k+1, k+2, \ldots, n\}$

are rejected.

Lemma 2.5.8 *Let the jobs be labelled in compatible order. Consider a schedule* Γ *in which job* k *is rejected but job* $k+1$ *is scheduled for some* $1 \leq k < n$. *Let the schedule* Γ' *be obtained from schedule* Γ *by rejecting job* $k+1$, *scheduling job* k, *and keeping the status of all other jobs unchanged. Then, the objective function value for schedule* Γ' *is at most the objective function value for schedule* Γ.

Proof. Let S_1 be the set of scheduled jobs which are less than $k+1$ in compatible order and let Let S_2 be the set of scheduled jobs which are greater than $k+1$ in compatible order. Because the jobs are also labelled according to Smith's rule, we can assume that they are scheduled in compatible order in schedule Γ.

When we reject job $k+1$ from schedule Γ, the change in the objective function value is

$$e_{k+1} - w_{k+1} \sum_{j \in S_1} p_j - w_{k+1}p_{k+1} - p_{k+1} \sum_{j \in S_2} w_j$$

When we now schedule job k to obtain schedule Γ', the change in the objective function value is

$$-e_k + w_k \sum_{j \in S_1} p_j + w_k p_k + p_k \sum_{j \in S_2} w_j$$

Thus, the total difference in the objective function value for schedule Γ' over that for schedule Γ is the sum of the above two quantities and is equal to

$$(e_{k+1} - e_k) + [(w_k - w_{k+1}) \sum_{j \in S_1} p_j] + (w_k p_k - w_{k+1} p_{k+1}) + [(p_k - p_{k+1}) \sum_{j \in S_2} w_j]$$

We have bracketed the above quantity into four parts. From the definition of compatible order, it is easy to see that each bracketed term is non-positive. This proves the lemma. ∎

We use the above lemma to show that there exists an optimal schedule in which a contiguous sequence of jobs starting from the first one is scheduled and the rest are rejected.

Lemma 2.5.9 *Assume that the jobs are labelled in compatible order. Then, there exists an optimal schedule in which, for some $1 \leq k \leq n$, the first k jobs are scheduled and the rest are rejected, i.e., a contiguous sequence of jobs starting from the first one is scheduled and the rest are rejected.*

Proof. The proof is by contradiction. Suppose that there is no optimal schedule with the structure stated above. Consider any optimal schedule Γ. Then, by applying the transformation described in Lemma 2.5.8 repeatedly, we can obtain a schedule Γ' in which, for some $1 \leq k \leq n$, the first k jobs are scheduled and the rest are rejected. The schedule Γ' has an objective function value which is less than or equal to the objective function value for the optimal schedule Γ. This contradicts our assumption at the beginning of the proof. ∎

We finally prove the correctness of algorithm COMPAT.

Theorem 2.5.10 *Algorithm* COMPAT *outputs an optimal schedule for* $1||(\sum_S w_jC_j + \sum_{\bar{S}} e_j)$ *in* $O(n \log n)$ *time when the processing times, weights, and rejection costs are compatible.*

Proof. Because of Lemma 2.5.9, we can assume that the optimal schedule, for some $1 \leq k \leq n$, schedules the first k jobs in compatible order and rejects the rest. Thus, if job i is scheduled in the optimal schedule, its completion time C_i must be $\sum_{j=1}^{i} p_i$, and it makes a contribution of w_iC_i to the objective function value. If job i is rejected in the optimal schedule, it makes a contribution of e_i to the optimal schedule. To minimize the objective function value, we must clearly take the smaller contribution for each job.

Also, it is easy to see that the algorithm can stop as soon as the first job is rejected. Suppose $w_iC_i > e_i$ and consider any job $j > i$. Because of the labelling of the jobs in compatible order, we have $w_j \geq w_i$ and $e_j \leq e_i$. Hence, it follows that $w_jC_j > e_j$. Thus, if job i is rejected, then job j must also be rejected.

It was shown earlier that the running time of algorithm `COMPAT` is $O(n \log n)$. ■

Chapter 3

Maximum Lateness/Tardiness with Rejection

3.1 Introduction.

In this chapter, we consider the problem of maximum tardiness/lateness with rejection for which the objective function is the sum of the maximum lateness/tardiness of the scheduled jobs and the total rejection penalty of the rejected jobs. The one machine versions of these two problems are denoted as $1| |(L_{max}(S)+\sum_{\bar{S}} e_j)$ and $1| |$-$(T_{max}(S) + \sum_{\bar{S}} e_j)$ respectively. In Section 3.2, we motivate the maximum tardiness with rejection problem by pointing out applications to hardware/software codesign for real-time systems. If rejection is not considered, the problems are solvable in polynomial time using the *Earliest Due Date (EDD)* rule: schedule the jobs in non-decreasing order of due dates d_j. In Section 3.3, we show that adding the option of rejection makes the problems $\mathcal{NP}$-complete. This result reflects joint work with James Orlin. In Section 3.4, we give two pseudo-polynomial time algorithms, based on dynamic programming, for $1| |(L_{max}(S) + \sum_{\bar{S}} e_j)$ and $1| |(T_{max}(S) + \sum_{\bar{S}} e_j)$. The first runs in $O(n \sum_{j=1}^{n} e_j)$ time and the second runs in $O(n \sum_{j=1}^{n} p_j)$ time. We generalize the second algorithm in Section 3.4.3 to any fixed number of unrelated parallel machines and solve $Rm|\ |(L_{max}(S) + \sum_{\bar{S}} e_j)$ and $Rm|\ |(T_{max}(S) + \sum_{\bar{S}} e_j)$.

We also develop an FPTAS for $1|\ |(T_{max}(S) + \sum_{\bar{S}} e_j)$ in Section 3.5. The FPTAS uses a geometric rounding technique on the total rejection penalty and works with what we call the *inflated rejection penalty.* In our FPTAS, we constrain the total rejection penalty for each set of rejected jobs to be of the form $(1 + \epsilon/2n)^i$, where $(1 + \epsilon)$ is the factor of approximation achieved by the algorithm.

Observe that the notion of an approximation algorithm (in the usual sense) does not hold much meaning for $1|\ |(L_{max}(S) + \sum_{\bar{S}} e_j)$ because the optimal objective function value could be negative. In such a case, it makes sense to consider *inverse approximation* algorithms for the problem. We introduce and motivate the notion of an inverse approximation algorithm in Section 3.6. We then give an inverse approximation scheme for $1|\ |(L_{max}(S) + \sum_{\bar{S}} e_j)$ in Section 3.6.2 and a fully polynomial time inverse approximation scheme (IFPTAS) for the problem $1|\ |(L_{max}(S) + \prod_{\bar{S}} e_j)$ in Section 3.6.3, where the total rejection penalty is the *product* (and not the sum) of the rejection costs of the rejected jobs.

3.2 Application to Hardware/Software Codesign for Real-time Systems.

Consider a real-time system where every job has to be scheduled with a (soft) deadline constraint. The functionality of the job can be implemented either in hardware or in software. Implementing a job in hardware is expensive but its speed of execution is several orders of magnitude higher than when it is implemented in software. On the other hand, implementing a job in software is much more economical, but due to the much slower speed of execution in software, implementing most jobs in software may cause too many jobs to complete after their deadline. Any hardware/software codesign for a real-time system must make trade-offs between the following two conflicting requirements:

- minimize the maximum tardiness of the the jobs implemented in software (this is not a problem for the jobs implemented in hardware because of their much higher speeds of execution), and
- minimize the cost of hardware used for implementing the jobs in hardware.

We can look at a job implemented in hardware as a "rejected" job as far as software implementation is concerned. The cost of implementing a job in hardware can thus be modelled as a rejection cost e_j for each job j. The efficient algorithms that we come up with in this chapter can be used to come up with an optimal partition of the jobs into the above two categories, i.e., implemention in hardware or software. Our techniques can be used as a decision support and analysis tool in hardware/software codesign for real-time systems.

3.3 Complexity of Maximum Lateness/Tardiness with Rejection.

In this section, we show that the problems $Pm| \;|(L_{max}(S) + \sum_{\bar{S}} e_j)$ and $Pm| \;|$-$(T_{max}(S) + \sum_{\bar{S}} e_j)$ are $\mathcal{NP}$-complete for any $m \geq 1$. Both the problems are solvable on one machine in polynomial time using the Earliest Due date First (EDD) rule *when rejection is not considered.* We show that adding the rejection option to even the *single* machine problem in each case makes it $\mathcal{NP}$-complete. In the discussion below, we will work with the problem $1| \;|(L_{max}(S) + \sum_{\bar{S}} e_j)$. The $\mathcal{NP}$-completeness proof for $1| \;|(T_{max}(S) + \sum_{\bar{S}} e_j)$ is exactly the same, since every job in our reduction has a non-negative lateness which is, hence, equal to its tardiness.

The decision problem formulation of $1| \;|(L_{max}(S) + \sum_{\bar{S}} e_j)$ is defined as follows:

Given a set of n independent jobs, $N = \{J_1, \ldots, J_n\}$, with processing times p_j, $\forall\, 1 \leq j \leq n$, due dates d_j, $\forall\, 1 \leq j \leq n$, and rejection penalties e_j, $\forall\, 1 \leq j \leq n$, a single machine, and a number K, is there a schedule of a subset of jobs $S \subseteq N$ on the machine such that $L_{max}(S) + \sum_{j \in \bar{S}=N-S} e_j \leq K$?

We reduce the Partition Problem [5] to this problem, thus proving that even on one machine, maximum lateness with rejection is $\mathcal{NP}$-complete.

Theorem 3.3.1 *$1|\ |(L_{max}(S) + \sum_{\bar{S}} e_j)$ is $\mathcal{NP}$-complete.*

Proof. $1|\ |(L_{max}(S) + \sum_{\bar{S}} e_j)$ is clearly in $\mathcal{NP}$. To prove that it is also $\mathcal{NP}$-hard, we reduce the Partition Problem [5] to $1|\ |(L_{max}(S) + \sum_{\bar{S}} e_j)$. The Partition Problem is defined as follows:

> Given a set $A = \{a_1, a_2, \ldots, a_n\}$ of n numbers such that $\sum_{i=1}^{n} a_i = 2b$, is there a subset A' of A such that $\sum_{a_i \in A'} a_i = b$?

Given an instance $A = \{a_1, \ldots, a_n\}$ of the partition problem, we create an instance of $1|\ |(L_{max}(S) + \sum_{\bar{S}} e_j)$ with $n+1$ jobs, $J_0, , J_1, \ldots, J_n$. For $i = 1, 2, \ldots, n$, each of the n elements a_i in the Partition Problem corresponds to a job J_i in $1|\ |(L_{max}(S) + \sum_{\bar{S}} e_j)$ with processing time $p_i = a_i$, due date $d_i = b$, and rejection cost $e_i = a_i/2$, where $b = \frac{1}{2}\sum_{i=1}^{n} a_i$. The special job J_0 has processing time equal to b, due date equal to 0, and rejection cost equal to ∞.

Consider any optimal schedule for $1|\ |(L_{max}(S) + \sum_{\bar{S}} e_j)$. Since J_0 has rejection cost of ∞ and the smallest due date, it must be scheduled first. Let S and $\bar{S}$ be the set of indices of the scheduled and rejected jobs respectively among $J_1, J_2, \ldots, J_n$ and let $x = \sum_{i \in S} p_i = \sum_{i \in S} a_i$. Observe that the makespan of the set of jobs in S is $x + b$, and since every job in S has the same due date b, the maximum lateness of this set of jobs is x. Also, the total rejection penalty of the rejected jobs is $\sum_{i \in \bar{S}} e_i = \sum_{i \in \bar{S}} a_i/2 = (2b - x)/2 = b - x/2$. Then, the value of the objective function for this schedule is

$$\max(x, b) + (b - x/2)$$

This function has a unique minimum of $\frac{3}{2}b$ at $x = b$ (see Figure 3-1). Hence, the best possible solution has $\sum_{i \in S} p_i = b$, and is optimum if it exists. Therefore, if the optimum solution to $1|\ |(L_{max}(S) + \sum_{\bar{S}} e_j)$ is equal to $\frac{3}{2}b$, then there exists a subset $A' = S$ of A such that $\sum_{i \in A'} a_i = b$, i.e., the answer to the Partition Problem is 'Yes,' and S is a witness. If the optimum solution to $1|\ |(L_{max}(S) + \sum_{\bar{S}} e_j)$ is greater than $\frac{3}{2}b$, then there does not exist any partition A' of A such that $\sum_{i \in A'} a_i = b$, i.e., the answer to the Partition Problem is 'No.' Conversely, if the answer to the

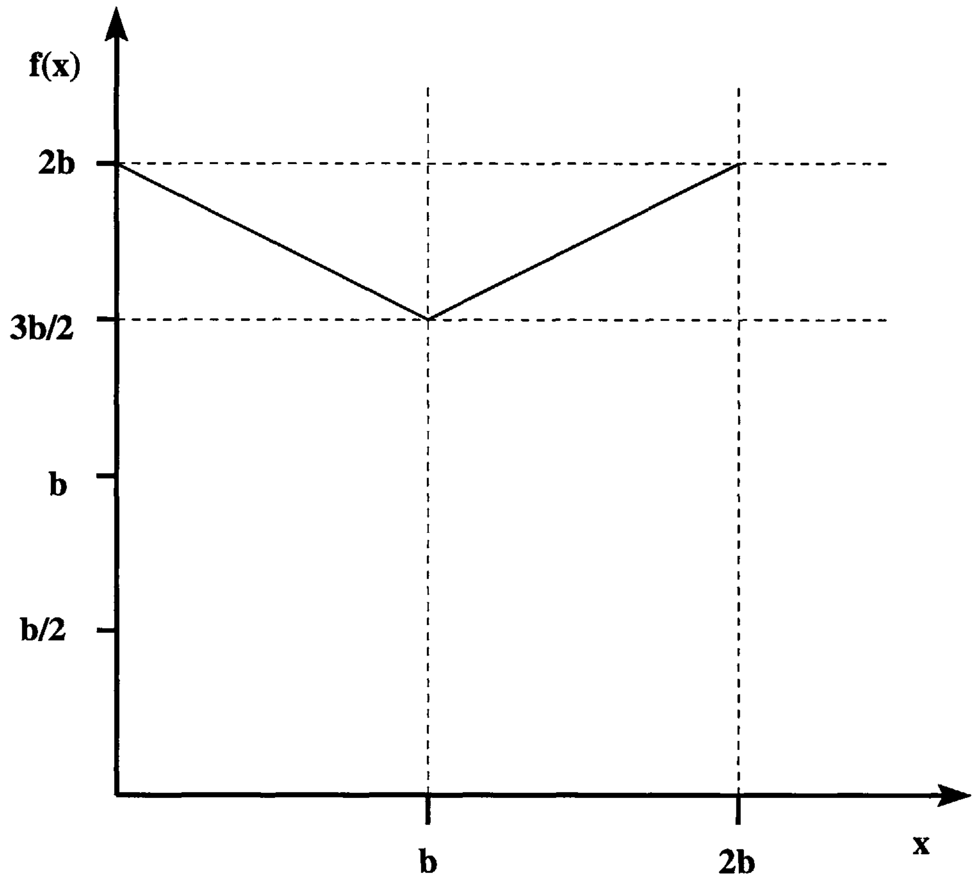

Figure 3-1: Graph for $f(x) = \max(x, b) + (b - x/2)$ in the range $0 \le x \le b$

Partition Problem is 'Yes,' the optimum solution to $1|\ |(L_{max}(S) + \sum_{\bar{S}} e_j)$ is clearly equal to $\frac{3}{2}b$. If the answer to the Partition Problem is 'No,' the optimum solution to $1|\ |(\sum_S w_j C_j + \sum_{\bar{S}} e_j)$ is clearly greater than $\frac{3}{2}b$. ∎

As mentioned before, the above proof also works for $1|\ |(T_{max}(S) + \sum_{\bar{S}} e_j)$, thus proving the following theorem.

Theorem 3.3.2 $1| \ |(T_{max}(S) + \sum_{\bar{S}} e_j)$ *is $\mathcal{NP}$-complete.*

As a corollary, it follows that $Pm| \ |(L_{max}(S)+\sum_{\bar{S}} e_j)$ and $Pm| \ |(T_{max}(S)+\sum_{\bar{S}} e_j)$ are both $\mathcal{NP}$-complete for any $m \geq 1$.

Corollary 3.3.3 *$Pm| \ |(L_{max}(S) + \sum_{\bar{S}} e_j)$ and $Pm| \ |(T_{max}(S) + \sum_{\bar{S}} e_j)$ are both $\mathcal{NP}$-complete for any $m \geq 1$.*

3.4 Pseudo-polynomial Time Algorithms.

In this section, we give pseudo-polynomial time algorithms for solving $1| \ |(L_{max}(S) + \sum_{\bar{S}} e_j)$ and $1| \ |(T_{max}(S) + \sum_{\bar{S}} e_j)$ exactly. We first give an $O(n \sum_{j=1}^{n} e_j)$ time algorithm (in section 3.4.1) and then an $O(n \sum_{j=1}^{n} p_j)$ time algorithm (in Section 3.4.2), using dynamic programming, to solve $1| \ |(L_{max}(S)+\sum_{\bar{S}} e_j)$ and $1| \ |(T_{max}(S)+\sum_{\bar{S}} e_j)$. The first runs in polynomial time when the rejection costs are polynomially bounded and the processing times are arbitrary, while the second runs in polynomial time when the processing times are polynomially bounded and the rejection costs are arbitrary. We then generalize our second dynamic program in Section 3.4.3 to a fixed number of unrelated parallel machines and solve $Rm| \ |(L_{max}(S) + \sum_{\bar{S}} e_j)$ and $Rm| \ |(T_{max}(S)+\sum_{\bar{S}} e_j)$. In Section 3.5, we show how to modify the dynamic program of Section 3.4.1 to obtain an FPTAS for $1| \ |(T_{max}(S) + \sum_{\bar{S}} e_j)$.

3.4.1 Dynamic Programming on the Rejection Costs e_j.

To solve our problem, we set up a dynamic program for the following problem: to find the schedule that minimizes the maximum lateness when the total rejection penalty of the rejected jobs is given. We number the jobs in non-decreasing order of due dates d_j. This is because the Earliest Due Date (EDD) rule minimizes the maximum lateness for any given set of scheduled jobs. Let $\phi_{e,j}$ denote the minimum value of the maximum lateness when the jobs in consideration are $j, j+1, \ldots, n$, and the total rejection penalty of the rejected jobs is e. Note that

$$\phi_{e,n} = \begin{cases} -\infty & \text{if } e = e_n \\ p_n - d_n & \text{if } e = 0 \\ \infty & \text{otherwise} \end{cases} \tag{3.1}$$

This forms the boundary conditions for the dynamic program. Since $\max(-\infty, x) = x$ for any x, the symbol $-\infty$ is the identity element for the max operator. Hence, we have assigned a lateness of $-\infty$ to the empty schedule. This should be treated as a special symbol rather than a value of negative infinity, since otherwise, the empty schedule will be the optimal schedule with an objective function value of negative infinity.

Now, consider any schedule for the jobs $j, j+1, \ldots, n$ that minimizes the maximum lateness when the total rejection penalty of the rejected jobs is e. We will refer to this as the optimal schedule in the discussion below. In any such schedule, there are two possible cases — either job j is rejected or job j is scheduled.

Case 1: Job j is rejected. This is possible only if $e \geq e_j$. Otherwise, there is no feasible solution with rejection penalty e and job j rejected, in which case only Case 2 applies. Hence, assume that $e \geq e_j$. Then, the value of the maximum lateness for the optimal schedule is clearly $\phi_{e-e_j,j+1}$, since the total rejection penalty of the rejected jobs among $j+1, \ldots, n$ must be $e - e_j$.

Case 2: Job j is scheduled. In this case, the total rejection penalty of the rejected jobs among $j+1, \ldots, n$ must be e. Also, when job j is scheduled before all jobs in the optimal schedule for jobs $j+1, j+2, \ldots, n$, the lateness of every scheduled job among $j+1, j+2, \ldots, n$ is increased by p_j and the lateness of job j is exactly $p_j - d_j$. Then, the value of the maximum lateness for the optimal schedule is clearly $\max(\phi_{e,j+1} + p_j, p_j - d_j)$.

Combining the above two cases, we have:

$$\phi_{e,j} = \begin{cases} \max(\phi_{e,j+1} + p_j, p_j - d_j) & \text{if } e < e_j \\ \min[\phi_{e-e_j,j+1}, \max(\phi_{e,j+1} + p_j, p_j - d_j)] & \text{otherwise} \end{cases} \tag{3.2}$$

Now, observe that the total rejection penalty of the rejected jobs can be at most $\sum_{j=1}^{n} e_j$, and the answers to our original problems are

- $\min\{(\phi_{e,1} + e) \mid 0 \leq e \leq \sum_{j=1}^{n} e_j\}$ for the problem $1| \;|(L_{max}(S) + \sum_{\bar{S}} e_j)$, and
- $\min\{(\max(0, \phi_{e,1}) + e) \mid 0 \leq e \leq \sum_{j=1}^{n} e_j\}$ for the problem $1| \;|(T_{max}(S) + \sum_{\bar{S}} e_j)$.

Thus, we need to compute at most $n \sum_{j=1}^{n} e_j$ table entries $\phi_{e,j}$. Computation of each such entry takes $O(1)$ time, so that the running time of the algorithm is $O(n \sum_{j=1}^{n} e_j)$.

Theorem 3.4.1 *Dynamic programming yields an $O(n \sum_{j=1}^{n} e_j)$ time algorithm for exactly solving $1| \;|(L_{max}(S) + \sum_{\bar{S}} e_j)$ and $1| \;|(T_{max}(S) + \sum_{\bar{S}} e_j)$.*

3.4.2 Dynamic Programming on the Lateness of the jobs.

In this section, we give another dynamic program that solves $1| \;|(L_{max}(S) + \sum_{\bar{S}} e_j)$ and $1| \;|(T_{max}(S) + \sum_{\bar{S}} e_j)$ in $O(n \sum_{j=1}^{n} p_j)$-time.

As before, we set up a dynamic program for a slightly different problem: namely, to find the schedule that minimizes the total rejection penalty of the rejected jobs when an upper bound on the maximum lateness of the scheduled jobs is given. We number the jobs in non-decreasing order of due dates d_j (as in Section 3.4.1). Let $\phi_{\ell,j}$ denote the minimum value of the total rejection penalty of the rejected jobs when the jobs in consideration are $j, j+1, \ldots, n$, and the maximum lateness of the scheduled jobs is at most ℓ. The boundary conditions of this dynamic program are given by

$$\phi_{\ell,n} = \begin{cases} e_n & \text{if } \ell = -\infty \\ 0 & \text{if } \ell \geq p_n - d_n \\ \infty & \text{otherwise} \end{cases} \tag{3.3}$$

Now, consider any schedule for the jobs $j, j+1, \ldots, n$ that minimizes the total rejection penalty of the rejected jobs when the maximum lateness of the scheduled jobs is at most ℓ. We will refer to this as the optimal schedule in the discussion below. In any such schedule, there are two possible cases — either job j is rejected or job j is scheduled.

Case 1: Job j is rejected. Then, the value of the total rejection penalty of the rejected jobs for the optimal schedule is clearly $\phi_{\ell,j+1} + e_j$, since the maximum lateness of the scheduled jobs among $j+1, \ldots, n$ is at most ℓ.

Case 2: Job j is scheduled. In this case, the lateness of job j is $p_j - d_j$. Hence, if the value of ℓ is smaller than $p_j - d_j$, there is no feasible solution with maximum lateness ℓ and job j scheduled, in which case only Case 1 applies. Therefore, assume that $\ell \geq p_j - d_j$. Now, when job j is scheduled before all jobs in the schedule for jobs $j+1, j+2, \ldots, n$, the lateness of every scheduled job among $j+1, j+2, \ldots, n$ is increased by p_j. Thus, the maximum lateness of the scheduled jobs among $j+1, \ldots, n$ can be at most $\ell - p_j$. Then, the value of the total rejection penalty of the rejected jobs for the optimal schedule is $\phi_{\ell-p_j,j+1}$.

Combining the above two cases, we have:

$$\phi_{\ell,j} = \begin{cases} \phi_{\ell,j+1} + e_j & \text{if } \ell < p_j - d_j \\ \min(\phi_{\ell,j+1} + e_j, \phi_{\ell-p_j,j+1}) & \text{otherwise} \end{cases} \tag{3.4}$$

We now obtain lower and upper bounds ℓ_{min} and ℓ_{max} respectively on the maximum lateness of any schedule. Since the maximum completion time of any job is at most $\sum_{j=1}^{n} p_j$ and the due dates are non-negative, we have $\ell_{max} \leq \sum_{j=1}^{n} p_j$. Also, observe that a scheduled job j has the minimum possible lateness when it is the first job to be scheduled. Thus, $\ell_{min} \geq \min_j(p_j - d_j) \geq \min_j(-d_j) = -\max_j d_j \geq -\sum_{j=1}^n p_j$, since we can assume, without any loss of generality, that the maximum due date is at most $\sum_{j=1}^n p_j$. Thus, the possible number of *finite* values of the maximum lateness ℓ for any schedule is at most $\ell_{max} - \ell_{min} \leq 2\sum_{j=1}^n p_j$. Note that in addition to this, the value of ℓ can also be $-\infty$ (for the empty schedule).

We can now see that the answers to our original problems are

- $\min\{(\ell + \phi_{\ell,1}) \mid \ell_{min} \leq \ell \leq \ell_{max} \text{ or } \ell = -\infty\}$ for the problem $1| \,|(L_{max}(S) + \sum_{\bar{S}} e_j)$, and
- $\min\{(\max(0,\ell) + \phi_{\ell,1}) \mid \ell_{min} \leq \ell \leq \ell_{max} \text{ or } \ell = -\infty\}$ for the problem $1| \,|(T_{max}(S) + \sum_{\bar{S}} e_j)$.

Thus, we need to compute at most $n(2\sum_{j=1}^n p_j)$ table entries $\phi_{\ell,j}$. Computation of each such entry takes $O(1)$ time, so that the running time of the algorithm is $O(n\sum_{j=1}^n p_j)$.

Theorem 3.4.2 *Dynamic programming yields an* $O(n\sum_{j=1}^n p_j)$ *time algorithm for exactly solving* $1| \,|(L_{max}(S) + \sum_{\bar{S}} e_j)$ *and* $1| \,|(T_{max}(S) + \sum_{\bar{S}} e_j)$.

3.4.3 Generalization to any fixed number of Unrelated Parallel Machines.

In this section, we generalize the dynamic program of Section 3.4.2 to any fixed number m of unrelated parallel machines and solve $Rm| \,|(L_{max}(S) + \sum_{\bar{S}} e_j)$ and $Rm| \,|(T_{max}(S) + \sum_{\bar{S}} e_j)$. Let p_{ij} denote the processing time of job i on machine j for $1 \leq i \leq n$ and $1 \leq j \leq m$ in the unrelated parallel machine model.

We set up a dynamic program for finding the schedule that minimizes the total rejection penalty of the rejected jobs when the maximum lateness of the jobs scheduled on each machine is given. We number the jobs in non-decreasing order of due dates d_j (as in Section 3.4.1). Note that given the set of jobs scheduled on a particular machine, the EDD rule still applies as far as minimizing the maximum lateness on that machine is concerned.

Let $\phi_{\ell_1,\ell_2,\ldots,\ell_m,j}$ denote the minimum value of the total rejection penalty of the rejected jobs when the jobs in consideration are $j, j+1, \ldots, n$, and the maximum lateness of the jobs scheduled on machine k is at most ℓ_k for all $1 \leq k \leq m$. The boundary conditions of this dynamic program are given by

$$\phi_{\ell_1,\ell_2,\ldots,\ell_m,n} = \begin{cases} e_n & \text{if } \ell_i = -\infty \ \forall\, i \\ 0 & \text{if } \exists\ k \text{ such that } \ell_k \geq p_{nk} - d_n \text{ and } \ell_i = -\infty \ \forall\, i \neq k \\ \infty & \text{otherwise} \end{cases} \tag{3.5}$$

Now, consider any schedule for the jobs $j, j+1, \ldots, n$ that minimizes the total rejection penalty of the rejected jobs when the maximum lateness of the jobs scheduled on machine k is at most ℓ_k for all $1 \leq k \leq m$. We will refer to this as the optimal schedule in the discussion below. In any such schedule, there are two possible cases — either job j is rejected or job j is scheduled.

Case 1: Job j is rejected. Then, the value of the total rejection penalty of the rejected jobs for the optimal schedule is clearly $\phi_{\ell_1,\ell_2,\ldots,\ell_m,j+1} + e_j$, since the maximum lateness of the jobs scheduled on machine k among $j+1, \ldots, n$ is at most ℓ_k.

Case 2: Job j is scheduled. Suppose job j is scheduled on machine k. This is possible only if p_{jk} is finite. In this case, the lateness of job j is $p_{jk} - d_j$. Hence, if the value of ℓ_k is smaller than $p_{jk} - d_j$, there is no feasible solution in which the maximum lateness on machine k is ℓ_k and job j is scheduled on machine k. Therefore, assume that $\ell_k \geq p_{jk} - d_j$. Now, when job j is scheduled on machine k before all jobs among $j+1, j+2, \ldots, n$ scheduled on machine k, the lateness of every scheduled job among $j+1, j+2, \ldots, n$ on machine k is increased by p_{jk}. Thus, the maximum lateness of the jobs scheduled on machine k among

$j+1,\ldots,n$ can be at most $\ell_k - p_{jk}$. Then, the value of the total rejection penalty of the rejected jobs for the optimal schedule is $\phi_{\ell_1,\ldots,\ell_k-p_{jk},\ldots,\ell_m,j+1}$.

Combining the above two cases, we have:

$$\phi_{\ell_1,\ell_2,\ldots,\ell_m,j} = \min(\phi_{\ell_1,\ell_2,\ldots,\ell_m,j+1} + e_j,$$
$$\min\{\phi_{\ell_1,\ldots,\ell_k-p_{jk},\ldots,\ell_m,j+1} \mid \ell_k \geq p_{jk} - d_j \text{ and } 1 \leq k \leq m\})$$

Let $\ell_{min,k}$ and $\ell_{max,k}$ denote the lower and upper bounds respectively on the maximum lateness of any schedule on machine k. By repeating the argument in the last section for machine k, we have $\ell_{max,k} \leq \sum_{j=1}^{n} p_{jk}$ and $\ell_{min,k} \geq -\sum_{j=1}^{n} p_{jk}$. We restrict the summation in each case to those jobs j that have a finite processing time on machine k. Thus, the possible number of *finite* values of the maximum lateness ℓ_k for any schedule on machine k is at most $\ell_{max,k} - \ell_{min,k} \leq 2\sum_{j=1}^{n} p_{jk}$. Note that in addition to this, the value of ℓ_k can also be $-\infty$ (for the empty schedule on machine k).

We can now see that the answer to our original problems are

- $\min\{(\max(\ell_1,\ell_2,\ldots,\ell_m) + \phi_{\ell_1,\ell_2,\ldots,\ell_m,1}) \mid \ell_{min,k} \leq \ell_k \leq \ell_{max,k}$ or $\ell_k = -\infty, 1 \leq k \leq m\}$ for the problem $Rm|\ |(L_{max}(S) + \sum_{\bar{S}} e_j)$, and
- $\min\{(\max(0,\ell_1,\ell_2,\ldots,\ell_m) + \phi_{\ell_1,\ell_2,\ldots,\ell_m,1}) \mid \ell_{min,k} \leq \ell_k \leq \ell_{max,k}$ or $\ell_k = -\infty, 1 \leq k \leq m\}$ for the problem $Rm|\ |(T_{max}(S) + \sum_{\bar{S}} e_j)$.

Thus, we need to compute at most $n\prod_{i=1}^{m}(2\sum_{j=1}^{n} p_{ji}) = n2^m \prod_{i=1}^{m}(\sum_{j=1}^{n} p_{ji})$ table entries $\phi_{\ell_1,\ell_2,\ldots,\ell_m,j}$. Computation of each such entry takes $O(m)$ time, so that the running time of the algorithm is $O(nm2^m \prod_{i=1}^{m}(\sum_{j=1}^{n} p_{ji}))$, which is pseudo-polynomial when the number of machines m is fixed.

Theorem 3.4.3 *Dynamic programming yields an $O(nm2^m \prod_{i=1}^{m}(\sum_{j=1}^{n} p_{ji}))$ time algorithm for exactly solving $Rm|\ |(L_{max}(S) + \sum_{\bar{S}} e_j)$ and $Rm|\ |(T_{max}(S) + \sum_{\bar{S}} e_j)$.*

3.5 Fully Polynomial Time Approximation Scheme for Maximum Tardiness with Rejection.

In this section, we describe a fully polynomial time approximation scheme (FPTAS) for $1||(T_{max}(S) + \sum_{\bar{S}} e_j)$. The algorithm runs in time polynomial in n, $\frac{1}{\epsilon}$, and the size (number of bits) of the rejection costs of the jobs.

We "trim" the state space of the dynamic program of Section 3.4.1 by fusing states that are "close" to each other. This fusion of "close" states is achieved by considering the *inflated rejection penalty* instead of the actual rejection penalty for a set of rejected jobs. We introduce this concept in the next section.

3.5.1 Inflated Rejection Penalty.

The actual rejection penalty for a set R of rejected jobs is $\sum_{i \in R} e_i$. The definition of *inflated rejection penalty* involves a *geometric rounding* technique which we state first. For any $\epsilon' > 0$ and $x \geq 1$, the quantities $\lceil x \rceil_{\epsilon'}$ and $\lfloor x \rfloor_{\epsilon'}$ denote x rounded up and rounded down respectively to the nearest power of $(1+\epsilon')$. Thus, if $(1+\epsilon')^{k-1} < x < (1+\epsilon')^k$, then $\lceil x \rceil_{\epsilon'} = (1+\epsilon')^k$ and $\lfloor x \rfloor_{\epsilon'} = (1+\epsilon')^{k-1}$. If x is an exact power of $(1+\epsilon')$, then $\lceil x \rceil_{\epsilon'} = \lfloor x \rfloor_{\epsilon'} = x$. Note that $\lceil x \rceil_{\epsilon'} \leq (1+\epsilon')x$ for any $x \geq 1$. We will use this property in Lemma 3.5.1.

Let $R = \{i_1, i_2, \ldots, i_k\}$, where $i_1 < i_2 < \cdots < i_k$ and $k \geq 0$. We define the *ϵ'-inflated rejection penalty* $f_{\epsilon'}(R)$ of the set R of jobs with respect to any $\epsilon' > 0$ as

$$f_{\epsilon'}(R) = \begin{cases} \lceil e_{i_1} + f_{\epsilon'}(R - \{i_1\}) \rceil_{\epsilon'} & \text{if } k \geq 1 \\ 0 & \text{if } R \text{ is empty} \end{cases} \tag{3.6}$$

As an illustrative example, let $R = \{1, 2, 5\}$. Then, $f_{\epsilon'}(R) = \lceil e_1 + \lceil e_2 + \lceil e_5 \rceil_{\epsilon'} \rceil_{\epsilon'} \rceil_{\epsilon'}$. Note how we start with the largest indexed job in the set R and consider the jobs in decreasing order of job index. At every step, we add the rejection cost of the next job and then round up. We will see later why this particular order of rounding is useful.

Since we are rounding up at each stage, it is easy to see that $f_{\epsilon'}(R) \geq \sum_{j \in R} e_j$ for any set R of jobs and any $\epsilon' > 0$. Hence, the reason for the term "inflated". We now prove a lemma which establishes an upper bound on the inflated rejection penalty in terms of the actual rejection penalty.

Lemma 3.5.1 *For any set R of jobs and any $\epsilon' > 0$,*

$$f_{\epsilon'}(R) \leq (1+\epsilon')^{|R|} \sum_{j \in R} e_j$$

Proof. The proof is by induction on the size of R. If R is empty, then both sides of the inequality are equal to zero, and the result is trivially true. For the inductive step, assume that $|R| \geq 1$, and let i be the smallest index job in R. By the induction hypothesis, we know that $f_{\epsilon'}(R - \{i\}) \leq (1+\epsilon')^{|R|-1} \sum_{j \in R - \{i\}} e_j$. Hence,

$$\begin{aligned}
f_{\epsilon'}(R) &= \lceil e_i + f_{\epsilon'}(R - \{i\}) \rceil_{\epsilon'} \\
&\leq (1+\epsilon')(e_i + f_{\epsilon'}(R - \{i\})) \quad \text{since } e_i \geq 1 \\
&\leq (1+\epsilon')(e_i + (1+\epsilon')^{|R|-1} \sum_{j \in R-\{i\}} e_j) \quad \text{by induction hypothesis} \\
&= (1+\epsilon')e_i + (1+\epsilon')^{|R|} \sum_{j \in R-\{i\}} e_j \\
&\leq (1+\epsilon')^{|R|}e_i + (1+\epsilon')^{|R|} \sum_{j \in R-\{i\}} e_j \\
&= (1+\epsilon')^{|R|} \sum_{j \in R} e_j
\end{aligned}$$

This completes the proof. ∎

Now, let $\epsilon' = \epsilon/2n$, where $(1+\epsilon)$ is the desired factor of approximation for the FPTAS. Since R has at most n jobs, we have $f_{\epsilon'}(R) \leq (1+\epsilon/2n)^n \sum_{j \in R} e_j \leq (1+\epsilon) \sum_{j \in R} e_j$. We put this down in the following lemma.

Lemma 3.5.2 *For any $\epsilon > 0$, let $\epsilon' = \frac{\epsilon}{2n}$. Then, for any set of jobs R,*

$$f_{\epsilon'}(R) \leq (1+\epsilon)\sum_{j \in R} e_j$$

This implies that if we work with the inflated rejection penalty instead of the actual rejection penalty, we will overestimate the rejection penalty by a factor of at most $(1+\epsilon)$. Working with the inflated rejection penalty has the following advantage. Since the inflated rejection penalty for any set of jobs is of the form $(1+\epsilon')^k$, we can store the exponent k instead of the actual value in the state of the dynamic program of Section 3.4.1. This reduces the number of states of the dynamic program so much so that we get an FPTAS out of it. We elaborate on this in the next section.

3.5.2 The Algorithm.

In this section, we arrive at an FPTAS for $1| \,|(T_{max}(S) + \sum_{\bar{S}} e_j)$ by setting up a dynamic program for the following problem: to find the schedule that minimizes the maximum lateness when the *inflated rejection penalty* of the rejected jobs is given. As before, we number the jobs in ascending order of due date d_j. Let $\phi_{k,j}$ denote the minimum value of the maximum lateness when the jobs in consideration are $j, j+1, \ldots, n$, and the inflated rejection penalty of the rejected jobs is $\tau_k = (1+\epsilon')^k$, where $\epsilon' = \epsilon/2n$. We will accommodate the zero inflated rejection cost (for the case when all the jobs are scheduled) by having $\tau_{-1} = 0$ for this case.

Note that

$$\phi_{k,n} = \begin{cases} -\infty & \text{if } \tau_k = \lceil e_n \rceil_{\epsilon'} \\ p_n - d_n & \text{if } k = -1 \\ \infty & \text{otherwise} \end{cases} \tag{3.7}$$

This forms the boundary conditions for the dynamic program.

Now, consider any schedule for the jobs $j, j+1, \ldots, n$ that minimizes the maximum lateness when the inflated rejection penalty of the rejected jobs is $\tau_k = (1+\epsilon')^k$. We will refer to this as the optimal schedule in the discussion below. In any such schedule, there are two possible cases — either job j is rejected or job j is scheduled.

Case 1: Job j is rejected. This is possible only if $\tau_k \geq \lceil e_j \rceil_{\epsilon'}$. Otherwise, there is no feasible solution with inflated rejection penalty τ_k and job j rejected, in which case only Case 2 applies. Hence, assume that $\tau_k \geq \lceil e_j \rceil_{\epsilon'}$. Then, the value of the maximum lateness for the optimal schedule is $\phi_{k',j+1}$, where $(1+\epsilon')^{k'}$ is the inflated rejection penalty of the rejected jobs among $j+1, \ldots, n$. From the definition of inflated rejection penalty, the possible values of k' must be such that $\lceil e_j + (1+\epsilon')^{k'} \rceil_{\epsilon'} = (1+\epsilon')^k$. Thus, the largest value of k' (call it $\tilde{k}$) is given by $(1+\epsilon')^{\tilde{k}} = \lfloor (1+\epsilon')^k - e_j \rfloor_{\epsilon'}$. But, k' may also take values smaller than $\tilde{k}$. Hence, the value of the maximum lateness for the optimal schedule is $\min_{-1 \leq k' \leq \tilde{k}} \phi_{k',j+1}$.

Case 2: Job j is scheduled. In this case, the inflated rejection penalty of the rejected jobs among $j+1, \ldots, n$ must be $(1+\epsilon')^k$. Also, when job j is scheduled before all jobs in the optimal schedule for jobs $j+1, j+2, \ldots, n$, the lateness of every scheduled job among $j+1, j+2, \ldots, n$ is increased by p_j and the lateness of job j is exactly $p_j - d_j$. Then, the value of the maximum lateness for the optimal schedule is clearly $\max(\phi_{k,j+1} + p_j, p_j - d_j)$.

Combining the above two cases, we have:

$$\phi_{k,j} = \begin{cases} \max(\phi_{k,j+1} + p_j, p_j - d_j) & \text{if } \tau_k < \lceil e_j \rceil_{\epsilon'} \\ \min[\min_{-1 \leq k' \leq \tilde{k}} \phi_{k',j+1}, \max(\phi_{k,j+1} + p_j, p_j - d_j)] & \text{otherwise} \end{cases} \quad (3.8)$$

Now, observe that the inflated rejection penalty of the rejected jobs is the largest when all the jobs are rejected. Hence, the inflated rejection penalty is at most

$f_{\epsilon'}(\{1,2,\ldots,n\}) \leq (1+\epsilon')^n \sum_{j=1}^n e_j$ (using Lemma 3.5.1). Thus, the largest value of k for which we need to compute $\phi_{k,j}$ is L, where L is the smallest integer such that $(1+\epsilon')^L \geq (1+\epsilon')^n \sum_{j=1}^n e_j$. Thus, L is the smallest integer greater than or equal to $\frac{\log \sum_{j=1}^n e_j}{\log(1+\epsilon')} + n$, whence $L = O(\frac{n}{\epsilon} \log \sum_{j=1}^n e_j)$.

When we consider the inflated rejection penalty instead of the actual rejection penalty, our problem becomes $1| \ |(T_{max}(S) + f_{\epsilon'}(\bar{S}))$. The answer to this problem is given by

$$\min\{\phi_{k,1} + \tau_k \mid -1 \leq k \leq L\}$$

Thus, we need to compute exactly $n(L+2)$ values $\phi_{k,j}$. Computation of each such value takes $O(L)$ time, so that the overall time for the dynamic program (FPTAS) is $O(nL^2) = O(\frac{n^3}{\epsilon^2} \log^2 \sum_{j=1}^n e_j)$. This is polynomial in the input size, since we need $\sum_{j=1}^n \log e_j$ bits to represent the rejection costs.

We now relate the optimal objective function values for the problems $1| \ |(T_{max}(S) + \sum_{\bar{S}} e_j)$ and $1| \ |(T_{max}(S) + f_{\epsilon'}(\bar{S}))$ through the following theorem.

Theorem 3.5.3 *For $\epsilon' = \epsilon/2n$, the optimal objective function value for $1| \ |(T_{max}(S) + f_{\epsilon'}(\bar{S}))$ is at most a factor of $(1+\epsilon)$ times the optimal objective function value for $1| \ |(T_{max}(S) + \sum_{\bar{S}} e_j)$.*

Proof. Consider any optimal schedule for $1| \ |(T_{max}(S) + \sum_{\bar{S}} e_j)$ in which the set of scheduled jobs is S. When we consider the inflated rejection penalty instead of the actual rejection penalty, the rejection penalty of the jobs in $\bar{S}$ increases by a factor of at most $(1+\epsilon)$ (using Lemma 3.5.2). Also, the maximum tardiness of the jobs in S remains unchanged. Hence, the objective function value for this schedule increases by a factor of at most $(1+\epsilon)$ when we consider the inflated rejection penalty instead of the actual rejection penalty. Thus, the optimal objective function value for $1| \ |(T_{max}(S) + f_{\epsilon'}(\bar{S}))$, with $\epsilon' = \epsilon/2n$, is at most $(1+\epsilon)$ times the optimal objective function value for $1| \ |(T_{max}(S) + \sum_{\bar{S}} e_j)$. ∎

This implies that the above dynamic program, which solves $1| \,|(T_{max}(S) + f_{\epsilon'}(\bar{S}))$ *exactly*, also gives a $(1+\epsilon)$-factor approximation for $1| \,|(T_{max}(S) + \sum_{\bar{S}} e_j)$.

Theorem 3.5.4 *There exists $(1+\epsilon)$-factor fully polynomial time approximation scheme (FPTAS) for $1| \,|(T_{max}(S) + \sum_{\bar{S}} e_j)$ which runs in $O(\frac{n^3}{\epsilon^2} \log^2 \sum_{j=1}^{n} e_j)$ time.*

3.6 Inverse Approximation for Maximum Lateness with Rejection.

As mentioned before, the notion of an approximation algorithm (in the usual sense) does not hold much meaning for $1| \,|(L_{max}(S) + \sum_{\bar{S}} e_j)$ because the optimal objective function value could be negative. In such a case, it makes sense to consider *inverse approximation* algorithms for the problem. In section 3.6.1, we introduce the notion of inverse approximation and discuss some of its advantages over the usual notion of approximation. We then give an inverse approximation scheme for $1| \,|(L_{max}(S) + \sum_{\bar{S}} e_j)$ in section 3.6.2 and a fully polynomial time inverse approximation scheme (IFPTAS) for the problem $1| \,|(L_{max}(S) + \prod_{\bar{S}} e_j)$ in section 3.6.3, where the total rejection penalty is the *product* (and not the sum) of the rejection costs of the rejected jobs.

3.6.1 Introduction to Inverse Approximation.

Any approximation algorithm must use some notion of distance from the optimal solution in order to measure the quality of the approximate solution that it produces. The most commonly used notion in the literature is that of *worst-case relative error* – a worst-case factor by which the objective function value of the output solution differs from the optimal objective function value. Although widely accepted, this way of measuring the quality of an approximate solution faces the following drawbacks:

- In the case when the optimal objective function value is non-positive, the relative error is an inappropriate measure of performance.
- A translation of variables has a dramatic impact on the relative error. That is, replacing x by $y = x - a$ will lead to very different measures of the relative error.
- The cost coefficients may themselves be uncertain, and the optimal objective function value may be very sensitive to small changes in the cost coefficients.

The possibility of having a negative objective function value for $1| \, |(L_{max}(S) + \sum_{\bar{S}} e_j)$ precludes the existence of an approximation algorithm for it in the usual sense. However, it does make sense to talk about *inverse approximation algorithms* (as defined below) for this problem. Inverse approximation, and more generally, *inverse optimization*, is a relatively new area of research and the concept was first introduced by Bitran, Chandru, Sempolinski, and Shapiro [31]. The work of Ahuja and Orlin [25, 26, 27] is a very good introduction to the area. We now define the notion of inverse approximation and then discuss some of the advantages of using it as a measure of the quality of performance.

A feasible solution x^* for an optimization problem with input costs (parameters) c_j is said to have an *inverse relative error* of at most ϵ if x^* is optimal for a problem with perturbed costs c'_j satisfying the following conditions: $c_j/(1+\epsilon) \leq c'_j \leq c_j(1+\epsilon)$ for all $c_j \geq 0$, and $c_j(1+\epsilon) \leq c'_j \leq c_j/(1+\epsilon)$ for all $c_j < 0$. An *ϵ-inverse approximation algorithm* (for a fixed $\epsilon > 0$) returns a feasible solution with an inverse relative error of at most ϵ. An *ϵ-inverse approximation scheme* returns a feasible solution with an inverse relative error of at most ϵ for any $\epsilon > 0$.

Some of the advantages of using the inverse relative error notion of approximation are as follows:

- Inverse approximation algorithms are invariant under the following operations:
  - subtraction of a constant from the objective function

  - changing the unit of the cost coefficients, that is, multiplying each cost coefficient by a common constant.

- Inverse approximation algorithms are properly defined even if the objective function takes on negative costs.

- Inverse approximation algorithms take into account that the cost coefficients are quite commonly known only with a limited degree of precision.

- One may consider weighted inverse relative errors, taking into account the fact that some data is known with higher precision than other data.

3.6.2 Inverse Approximation Scheme for $1| |(L_{max}(S)+\sum_{\bar{S}} e_j)$.

In this section, we give an inverse approximation scheme for $1| |(L_{max}(S) + \sum_{\bar{S}} e_j)$. Our approach consists of first rounding up the rejection costs e_j to $e'_j = \lceil e_j \rceil_\epsilon$, and then finding an optimal solution for $1| |(L_{max}(S) + \sum_{\bar{S}} e_j)$ with the modified costs e'_j. Note that $e'_j \leq (1+\epsilon)e_j$ for all j. Hence, by the definition of inverse approximation, it is clear that this optimal solution has an inverse relative error of at most ϵ.

To find the optimal solution to the modified problem, we run the dynamic program of Section 3.4.1. Observe that due to the modified rejection costs, the total rejection penalty of any set of jobs is of the form $\sum_{i=0}^{L} a_i(1+\epsilon)^i$ with $a_i \geq 0$ for all i. Here, L is such that $(1+\epsilon)^L$ is the maximum rounded rejection cost. Thus, if e_{max} is the maximum rejection cost, then L is the smallest integer such that $(1+\epsilon)^L \geq e_{max}$, i.e., $L = O(\frac{1}{\epsilon}\log e_{max})$. Note that it is possible for a_i to be greater than 1, since two rounded rejection costs could have the same value $(1+\epsilon)^i$.

Hence, instead of storing the actual rejection penalty $e = \sum_{i=0}^{L} a_i(1+\epsilon)^i$ (which is no longer an integer) in the state of the dynamic program, we can store the $(L+1)$-tuple $(a_0, a_1, \ldots, a_L)$, which denotes the rejection penalty of $\sum_{i=0}^{L} a_i(1+\epsilon)^i$. Note that $a_i \leq n$, and hence, the total number of such tuples is $n^{L+1} = n^{O(\log e_{max}/\epsilon)}$. Thus, we need to compute at most $n * n^{O(\log e_{max}/\epsilon)}$ entries $\phi_{(a_0,a_1,\ldots,a_L),j}$. Computation of each such entry takes $O(1)$ time, so that the running time of the algorithm is $O(n^{1+O(\log e_{max}/\epsilon)}) = O(n^{O(\log e_{max}/\epsilon)})$.

Theorem 3.6.1 *Dynamic programming yields an ϵ-inverse approximation scheme for $1| \ |(L_{max}(S) + \sum_{\bar{S}} e_j)$, which runs in $O(n^{O(\log e_{max}/\epsilon)})$ time.*

3.6.3 Fully Polynomial Time Inverse Approximation Scheme for $1| \ |(L_{max}(S) + \prod_{\bar{S}} e_j)$.

In this section, we describe a full polynomial time inverse approximation scheme (IFPTAS) for $1| \ |(L_{max}(S) + \prod_{\bar{S}} e_j)$. The algorithm runs in time polynomial in n, $\frac{1}{\epsilon}$, and the size (number of bits) of the rejection costs of the jobs. Note that for this problem, the total rejection penalty is the *product* and not the sum of the rejection costs of the rejected jobs.

As in the previous section, we first round up the rejection costs e_j to $e'_j = \lceil e_j \rceil_\epsilon$, and then find an optimal solution for $1| \ |(L_{max}(S)+\prod_{\bar{S}} e_j)$ with the modified costs e'_j. Note that $e'_j \leq (1+\epsilon)e_j$ for all j. Hence, by the definition of inverse approximation, it is clear that this optimal solution has an inverse relative error of at most ϵ. To find the optimal solution to the modified problem, we give a dynamic program which is very similar to that of Section 3.4.1. Observe that due to the modified rejection costs, the total rejection penalty of any set of jobs is of the form $(1+\epsilon)^k$, i.e., a power of $(1+\epsilon)$. Hence, instead of storing the actual rejection penalty $e = (1+\epsilon)^k$ (which is no longer an integer) in the state of the dynamic program, we can store the exponent of the rejection penalty, i.e., the value k will denote a rejection penalty of $\tau_k = (1+\epsilon)^k$ for $k > 0$. We explain below why $k = 0$ is a special case and how we handle it.

Note that since the total rejection penalty is the product of the rejection costs of the rejected jobs, jobs with a rejection cost of 1 do not increase the rejection penalty when they get rejected. In order to avoid this anomaly, we will assume that $e_j > 1$ for all j. Then, the exponent of $k = 0$ in the rejection penalty will be indicative of the fact that none of the jobs are rejected, and we will make the rejection penalty zero in this case by defining $\tau_0 = 0$.

We set up a dynamic program for the following problem: to find the schedule that minimizes the maximum lateness when the total rejection penalty (product form) of the rejected jobs is given. As in Section 3.4.1, we number the jobs in ascending order of due date d_j. Let $\phi_{k,j}$ denote the minimum value of the maximum lateness when the jobs in consideration are $j, j+1, \dots, n$, and the total rejection penalty of the rejected jobs is τ_k, where $\tau_k = (1+\epsilon)^k$ for $k > 0$ and $\tau_0 = 0$. Let L_j denote the exponent of e'_j, i.e., $e'_j = (1+\epsilon)^{L_j}$.

Note that

$$\phi_{k,n} = \begin{cases} -\infty & \text{if } k = L_n \\ p_n - d_n & \text{if } k = 0 \\ \infty & \text{otherwise} \end{cases} \tag{3.9}$$

This forms the boundary conditions for the dynamic program.

Now, consider any schedule for the jobs $j, j+1, \dots, n$ that minimizes the maximum lateness when the total rejection penalty of the rejected jobs is $(1+\epsilon)^k$. We will refer to this as the optimal schedule in the discussion below. In any such schedule, there are two possible cases — either job j is rejected or job j is scheduled.

Case 1: Job j is rejected. This is possible only if $(1+\epsilon)^k \geq e'_j$, i.e., $k \geq L_j$. Otherwise, there is no feasible solution with total rejection penalty $(1+\epsilon)^k$ in which job j (with rejection cost e'_j) is rejected, in which case only Case 2 applies. Hence, assume that $k \geq L_j$. Then, the value of the maximum lateness for the optimal schedule is clearly $\phi_{k-L_j,j+1}$, since the total rejection penalty of the rejected jobs among $j+1, \dots, n$ must be $(1+\epsilon)^k / e'_j = (1+\epsilon)^{(k-L_j)}$.

Case 2: Job j is scheduled. In this case, the total rejection penalty of the rejected jobs among $j+1, \dots, n$ must be $(1+\epsilon)^k$. Also, when job j is scheduled before all jobs in the optimal schedule for jobs $j+1, j+2, \dots, n$, the lateness of every scheduled job among $j+1, j+2, \dots, n$ is increased by p_j and the lateness of job

j is exactly $p_j - d_j$. Then, the value of the maximum lateness for the optimal schedule is clearly $\max(\phi_{k,j+1} + p_j, p_j - d_j)$.

Combining the above two cases, we have:

$$\phi_{k,j} = \begin{cases} \max(\phi_{k,j+1} + p_j, p_j - d_j) & \text{if } k < L_j \\ \min[\phi_{k-L_j,j+1}, \max(\phi_{k,j+1} + p_j, p_j - d_j)] & \text{otherwise} \end{cases} \tag{3.10}$$

Now, observe that the total rejection penalty of the rejected jobs is at most $\prod_{j=1}^{n} e'_j = \prod_{j=1}^{n}(1+\epsilon)^{L_j} = (1+\epsilon)^{\sum_{j=1}^{n} L_j}$. From the definition of the L_j's, it follows that L_j is the smallest integer such that $(1+\epsilon')^{L_j} \geq e_j$, i.e., $L_j = O(\frac{1}{\epsilon}\log e_j)$. Hence, the maximum exponent of the total rejection penalty is $\sum_{j=1}^{n} L_j = O(\frac{1}{\epsilon}\sum_{j=1}^{n}\log e_j)$.

The answer to our problem $1| |(L_{max}(S) + \prod_{\bar{S}} e_j)$ with modified rejection costs e'_j is given by

$$\min\{\phi_{k,1} + \tau_k \mid 0 \leq k \leq \sum_{j=1}^{n} L_j\}$$

Thus, we need to compute at most $n\sum_{j=1}^{n} L_j$ values $\phi_{k,j}$. Computation of each such value takes $O(1)$ time, so that the overall running time for the dynamic program (IFPTAS) is $O(n\sum_{j=1}^{n} L_j) = O(\frac{n}{\epsilon}\sum_{j=1}^{n}\log e_j)$. This is polynomial in the input size, since we need $\sum_{j=1}^{n}\log e_j$ bits to represent the rejection costs.

Theorem 3.6.2 *Dynamic programming yields an ϵ-inverse fully polynomial time approximation scheme (IFPTAS) for $1| |(L_{max}(S) + \prod_{\bar{S}} e_j)$, which runs in $O(\frac{n}{\epsilon}\sum_{j=1}^{n}\log e_j)$ time.*

Chapter 4

Makespan with Rejection

4.1 Introduction.

In this chapter, we consider the problem of makespan with rejection for which the objective function is the sum of the makespan of the scheduled jobs and the total rejection penalty of the rejected jobs. The one machine version of this problem is denoted as $1| |(C_{max}(S) + \sum_{\bar{S}} e_j)$. If rejection is not considered, the problem is trivial on one machine and $\mathcal{NP}$-complete on more than one machine. In Section 4.2, we give a simple $O(n)$ time algorithm for $1| |(C_{max}(S) + \sum_{\bar{S}} e_j)$. In Section 4.3, we give a pseudo-polynomial time algorithm, based on dynamic programming, for the problem on any fixed number of unrelated parallel machines, i.e., $Rm| |(C_{max}(S) + \sum_{\bar{S}} e_j)$.

We also develop a fully polynomial time approximation scheme (FPTAS) for $Rm| |(C_{max}(S) + \sum_{\bar{S}} e_j)$ in Section 4.4. The FPTAS uses the geometric rounding technique on the job completion times and works with aligned schedules (as already introduced in Section 2.4.1). In our FPTAS, we constrain each job to finish at times of the form $(1 + \epsilon/2n)^i$, where $(1 + \epsilon)$ is the factor of approximation achieved by the algorithm.

4.2 Complexity of Makespan with Rejection.

For any fixed number of machines $m > 1$, $Pm| \,|(C_{max}(S) + \sum_{\bar{S}} e_j)$ is trivially seen to be $\mathcal{NP}$-complete by restricting the problem to $Pm| \,| \sum w_j C_j$, a known $\mathcal{NP}$-complete problem [5].

Theorem 4.2.1 *$Pm| \,|(C_{max}(S) + \sum_{\bar{S}} e_j)$ is $\mathcal{NP}$-complete for any $m > 1$.*

For the one machine case, we give a simple $O(n)$ time algorithm for this problem. Note that if S is the set of scheduled jobs, then $C_{max}(S) = \sum_{j \in S} p_j$ on one machine. Hence, the objective function reduces to

$$\sum_{j \in S} p_j + \sum_{j \in \bar{S}} e_j$$

Hence, if a job j is scheduled, it contributes p_j to the sum, and if it is rejected, it contributes e_j to the sum. It is easy to see now that the following strategy gives the optimal objective function value: if $p_j \le e_j$, then schedule job j, otherwise reject job j. The running time is clearly $O(n)$. We put this down in the following theorem.

Theorem 4.2.2 *There exists an $O(n)$ time algorithm for exactly solving $1| \,|(C_{max}(S) + \sum_{\bar{S}} e_j)$.*

4.3 Pseudo-polynomial Time Algorithm.

In this section, we give a pseudo-polynomial time algorithm, based on dynamic programming, to solve the makespan with rejection problem on any fixed number m of unrelated parallel machines, i.e., $Rm| \,|(C_{max}(S) + \sum_{\bar{S}} e_j)$. Let p_{ij} denote the processing time of job i on machine j for $1 \le i \le n$ and $1 \le j \le m$ in the unrelated parallel machine model. In Section 4.4, we show how to modify this dynamic program to obtain an FPTAS for $Rm| \,|(C_{max}(S) + \sum_{\bar{S}} e_j)$.

We set up a dynamic program for finding the schedule that minimizes the total rejection penalty of the rejected jobs when the makespan of the jobs scheduled on each machine is given. We number the jobs in arbitrary order, since the makespan on any machine does not depend on the order in which a given set of jobs is scheduled on that machine. We will assume that the jobs are scheduled in increasing order of job index on any given machine.

Let $\phi_{s_1,s_2,\ldots,s_m,j}$ denote the minimum value of the total rejection penalty of the rejected jobs when the jobs in consideration are $1, 2, \ldots, j$, and the makespan of the jobs scheduled on machine k is s_k for all $1 \leq k \leq m$. The boundary conditions for this dynamic program are given by

$$\phi_{s_1,s_2,\ldots,s_m,1} = \begin{cases} e_1 & \text{if } s_i = 0 \ \forall \ i \\ 0 & \text{if } \exists \ k \text{ such that } s_k = p_{1k} \text{ and } s_i = 0 \ \forall \ i \neq k \\ \infty & \text{otherwise} \end{cases} \tag{4.1}$$

Now, consider any schedule for the jobs $1, 2, \ldots, j$ that minimizes the total rejection penalty of the rejected jobs when the makespan of the jobs scheduled on machine k is s_k for $1 \leq k \leq m$. We will refer to this as the optimal schedule in the discussion below. In any such schedule, there are two possible cases — either job j is rejected or job j is scheduled.

Case 1: Job j is rejected. Then, the value of the total rejection penalty of the rejected jobs for the optimal schedule is clearly $\phi_{s_1,s_2,\ldots,s_m,j-1} + e_j$, since the makespan of the jobs among $1, 2, \ldots, j-1$ scheduled on machine k is s_k.

Case 2: Job j is scheduled. Suppose job j is scheduled on machine k. This is possible only if p_{jk} is finite. In this case, the makespan on machine k is at least p_{jk}. Hence, if $s_k < p_{jk}$, there is no feasible solution in which the makespan on machine k is s_k and job j is scheduled on machine k. Therefore, assume that $s_k \geq p_{jk}$. In this case, if there was a job among $1, 2, \ldots, j-1$ scheduled on machine k before job j, it must have completed at time $s_k - p_{jk}$. Then, the value of the total rejection penalty of the rejected jobs for the optimal schedule is $\phi_{s_1,\ldots,s_k-p_{jk},\ldots,s_m,j-1}$.

Combining the above two cases, we have:

$$\begin{aligned}\phi_{s_1,s_2,\ldots,s_m,j} = \min(&\phi_{s_1,s_2,\ldots,s_m,j-1} + e_j,\\ &\min\{\phi_{s_1,\ldots,s_k-p_{jk},\ldots,s_m,j-1} \mid s_k \geq p_{jk} \text{ and } 1 \leq k \leq m\})\end{aligned}$$

Note that the value of the makespan on machine k is at most $\sum_{j=1}^{n} p_{jk}$. We restrict this summation to only those values of k for which p_{jk} is finite. We can now see that the answer to our original problem is

$$\min\{(\max(s_1, s_2, \ldots, s_m) + \phi_{s_1,s_2,\ldots,s_m,n}) \mid 0 \leq s_k \leq \sum_{j=1}^{n} p_{jk} \text{ and } 1 \leq k \leq m\}$$

Thus, we need to compute at most $n \prod_{i=1}^{m} (\sum_{j=1}^{n} p_{ji})$ table entries $\phi_{s_1,s_2,\ldots,s_m,j}$. Computation of each such entry takes $O(m)$ time, so that the running time of the algorithm is $O(nm \prod_{i=1}^{m} (\sum_{j=1}^{n} p_{ji}))$, which is polynomial in the values of the processing times.

Theorem 4.3.1 *Dynamic programming yields an $O(nm \prod_{i=1}^{m} (\sum_{j=1}^{n} p_{ji}))$ time algorithm for exactly solving $Rm| \,|(C_{max}(S) + \sum_{\bar{S}} e_j)$.*

4.4 Fully Polynomial Time Approximation Scheme.

In this section, we describe a fully polynomial time approximation scheme (FPTAS) for $Rm| \,|(C_{max}(S) + \sum_{\bar{S}} e_j)$. The algorithm runs in time polynomial in n, $\frac{1}{\epsilon}$, and the size (number of bits) of the processing times of the jobs.

We "trim" the state space of the dynamic program of Section 4.3 by fusing states that are "close" to each other. This fusion of "close" states is achieved by transforming any schedule to an ϵ'-aligned schedule, as described in Section 2.4.1. Recall that in an ϵ'-aligned schedule, every job finishes at a time of the form $\tau_i = (1+\epsilon')^i$, for $i \geq 0$, and $\epsilon' > 0$. As before, we will handle the zero completion time of the empty schedule by defining $\tau_{-1} = 0$.

The following lemma establishes an upper bound on the increase in the optimal objective function value for $Rm|\ |(C_{max}(S) + \sum_{\bar{S}} e_j)$ when we restrict our attention to ϵ'-aligned schedules only.

Lemma 4.4.1 *For $Rm|\ |(C_{max}(S) + \sum_{\bar{S}} e_j)$, the optimal objective function value increases by a factor of at most $(1+\epsilon')^n$ for any $\epsilon' > 0$, when we restrict our attention to ϵ'-aligned schedules only.*

Proof. The proof of Lemma 4.4.1 tells us that in any given schedule, the completion time C_i of the i^{th} scheduled job on any machine increases by a factor of at most $(1+\epsilon')^i$ after the schedule on that machine is ϵ'-aligned. Thus, the makespan on any machine increases by a factor of at most $(1+\epsilon')^n$ after the schedule on that machine is ϵ'-aligned.

Consider an optimal schedule Γ for $Rm|\ |(C_{max}(S) + \sum_{\bar{S}} e_j)$ in which the set of scheduled jobs is S. Since the makespan of the entire schedule is the maximum of the makespan on each machine, we conclude that the makespan of Γ increases by a factor of at most $(1+\epsilon')^n$ after the schedule Γ is ϵ'-aligned. Also, the rejection cost of the jobs in $\bar{S}$ trivially remains unchanged. Hence, the objective function value for schedule Γ increases by a factor of at most $(1+\epsilon')^n$ when Γ is ϵ'-aligned. This implies that the optimal objective function value increases by a factor of at most $(1+\epsilon')^n$ for any $\epsilon' > 0$, when we restrict our attention to ϵ'-aligned schedules only. ∎

Setting $\epsilon' = \frac{\epsilon}{2n}$ gives us an $(1+\epsilon)$-factor increase in the optimal objective function value, as stated in the next lemma. The proof is similar to Lemma 2.4.2.

Lemma 4.4.2 *For $Rm|\ |(C_{max}(S)+\sum_{\bar{S}} e_j)$ and $\epsilon' = \epsilon/2n$ for any $\epsilon > 0$, the optimal objective function value increases by a factor of at most $(1+\epsilon)$ for any $\epsilon > 0$, when we restrict our attention to ϵ'-aligned schedules only.*

Let $\epsilon' = \frac{\epsilon}{2n}$. For our FPTAS, we set up a dynamic program for a harder problem: namely, to find the ϵ'-aligned schedule that minimizes the total rejection penalty of the rejected jobs when the makespan of the jobs scheduled on each machine is given. We number the jobs in arbitrary order, since the makespan on any machine does not depend on the order in which a given set of jobs is scheduled on that machine. We will assume that the jobs are scheduled in increasing order of job index on any given machine.

Let $\phi_{i_1,i_2,\ldots,i_m,j}$ denote the minimum value of the total rejection penalty of the rejected jobs when the jobs in consideration are $1, 2, \ldots, j$, and the makespan of the ϵ'-aligned schedule on machine k is τ_{i_k} for $1 \leq k \leq m$. The boundary conditions for this dynamic program are given by

$$\phi_{i_1,i_2,\ldots,i_m,1} = \begin{cases} e_1 & \text{if } i_k = -1 \ \forall\ k \\ 0 & \text{if } \exists\ k \text{ such that } p_{1k} \in (\tau_{i_k-1}, \tau_{i_k}] \text{ and } i_\ell = 0 \ \forall\ \ell \neq k \\ \infty & \text{otherwise} \end{cases} \tag{4.2}$$

Now, consider any ϵ'-aligned schedule for the jobs $1, 2, \ldots, j$ that minimizes the total rejection penalty of the rejected jobs when the makespan of the jobs scheduled on machine k is τ_{i_k} for $1 \leq k \leq m$. We will refer to this as the optimal schedule in the discussion below. In any such schedule, there are two possible cases — either job j is rejected or job j is scheduled.

Case 1: Job j is rejected. Then, the value of the total rejection penalty of the rejected jobs for the optimal schedule is clearly $\phi_{i_1,i_2,\ldots,i_m,j-1} + e_j$, since the makespan of the jobs among $1, 2, \ldots, j-1$ scheduled on machine k is τ_{i_k}.

Case 2: Job j is scheduled. Suppose job j is scheduled on machine k. This is possible only if p_{jk} is finite. In this case, the makespan on machine k is at least p_{jk}. Hence, if $\tau_{i_k} < p_{jk}$, there is no feasible solution in which the makespan on machine k is τ_{i_k} and job j is scheduled on machine k. Therefore, assume that $\tau_{i_k} \geq p_{jk}$. In this case, if there was a job among $1, 2, \ldots, j-1$ scheduled on machine k before job j, it must have completed at time $\tau_{i'_k}$, where i'_k is the largest value of ℓ satisfying $\tau_\ell + p_{jk} \leq \tau_{i_k}$. Then, the value of the total rejection penalty of the rejected jobs for the optimal schedule is $\phi_{i_1,\ldots,i'_k,\ldots,i_m,j-1}$.

Combining the above two cases, we have:

$$\phi_{i_1,i_2,\ldots,i_m,j} = \min(\phi_{i_1,i_2,\ldots,i_m,j-1} + e_j, \min\{\phi_{i_1,\ldots,i'_k,\ldots,i_m,j-1} \mid \tau_{i_k} \geq p_{jk} \text{ and } 1 \leq k \leq m\})$$

Now, observe that for finding an ϵ'-aligned schedule with the optimum objective function value, it is sufficient to assume that the completion time of the latest scheduled job on any machine k is at most $(1+\epsilon')^n \sum_{j=1}^n p_{jk}$ for all $1 \leq k \leq m$. Thus, the largest value of i_k, $1 \leq k \leq m$, for which we need to compute $\phi_{i_1,i_2,\ldots,i_m,j}$ is L_k, where L_k is the smallest integer such that $\tau_{L_k} \geq (1+\epsilon')^n \sum_{j=1}^n p_{jk}$. Thus, L_k is the smallest integer greater than or equal to $\frac{\log \sum_{j=1}^n p_{jk}}{\log(1+\epsilon')} + n$, whence $L_k = O(\frac{n}{\epsilon} \log \sum_{j=1}^n p_{jk})$.

The answer to our problem $Rm| \, |(C_{max}(S) + \sum_{\bar{S}} e_j)$ when we consider ϵ'-aligned schedules only is

$$\min\{(\tau_{\max(i_1,i_2,\ldots,i_m)} + \phi_{i_1,i_2,\ldots,i_m,n}) \mid -1 \leq i_k \leq L_k \text{ and } 1 \leq k \leq m\}$$

Thus, we need to compute at most $n(L_1+2)(L_2+2)\cdots(L_m+2)$ values $\phi_{i_1,i_2,\ldots,i_m,j}$. Computation of each such value takes $O(m)$ time, so that the overall running time for the dynamic program (FPTAS) is $O(nm \prod_{i=1}^m L_i) = O(\frac{n^{m+1}m}{\epsilon^m} \prod_{k=1}^m (\log \sum_{j=1}^n p_{jk}))$. This is polynomial in the input size when the number of machines m is fixed, since we need $\sum_{j=1}^n \log p_{jk}$ bits to represent the processing times on machine k.

Theorem 4.4.3 *Dynamic programming yields a $(1+\epsilon)$-factor fully polynomial time approximation scheme (FPTAS) for $Rm| \, |(C_{max}(S) + \sum_{\bar{S}} e_j)$, which runs in $O(\frac{n^{m+1}m}{\epsilon^m} \prod_{k=1}^m (\log \sum_{j=1}^n p_{jk}))$ time.*

Chapter 5

Fixed-precision and Logarithmic-precision Models and Strongly Polynomial Time Algorithms

5.1 Introduction.

In this chapter, we introduce a new model for algorithm design which we call the *L-bit precision model.* In this model, the input numbers (for the problem) are of the form $c * 2^t$, where $t \geq 0$ is arbitrary and $c < c^* = 2^L$. Thus, the input numbers have a precision of L bits. The L-bit precision model is realistic because it incorporates the format in which large numbers are stored in computers today. One such format which is becoming increasingly popular in the computer industry is the *IEEE Standard for Binary Floating-point Arithmetic* [28, 29, 30].

In this L-bit precision model, we define a *polynomial time algorithm* to have running time polynomial in $c^* = 2^L$ and n, where n is the size of the problem instance. Depending on the value of L, we have two different models of precision which we describe below.

Fixed-precision Model: In this model, L is a constant, so that a polynomial time algorithm has running time polynomial in n under this model, and is, hence, a *strongly polynomial time algorithm*, i.e., the running time does not depend on the sizes of the input numbers.

Logarithmic-precision Model: In this model, L is $O(\log n)$, where n is the size of the problem instance. This implies that $c^* = 2^L$ is polynomial in n. Hence, a polynomial time algorithm is also a *strongly polynomial time algorithm* under this model.

We focus on designing algorithms for $\mathcal{NP}$-complete problems under the above models. In particular, we give strongly polynomial time algorithms for the following $\mathcal{NP}$-complete problems:

- the *knapsack* problem (Section 5.2),
- the *k-partition* problem for a fixed k (Section 5.3), and
- *scheduling to minimize makespan* on a fixed number of identical parallel machines (Section 5.4).

This is joint work with James Orlin. Our results show that it is possible for $\mathcal{NP}$-complete problems to have strongly polynomial time algorithms under the above models.

5.2 The Knapsack Problem.

The *knapsack* problem is defined as follows:

Given sets $A = \{a_1, a_2, \ldots, a_n\}$ and $C = \{c_1, c_2, \ldots, c_n\}$ of n numbers each and a number b, find a set $S \subseteq \{1, 2, \ldots, n\}$ which maximizes $\sum_{j \in S} c_j$ subject to the condition that $\sum_{j \in S} a_j \leq b$.

We can interpret the above definition as follows. Suppose there are n items, with item i having weight c_i and volume a_i. The items are to be placed in a knapsack with (volume) capacity b. We want to find the subset S of items which can be put into the knapsack to maximize the total weight of items in the knapsack. This problem is $\mathcal{NP}$-complete [5].

We first discuss a dynamic program to solve this problem when the c_i's are arbitrary. We then show how the running time of this dynamic program can be improved when the c_i's have L bits of precision. We will use the indicator variable x_j to denote whether item j is placed or not placed in the knapsack. Recall that S is the set of items placed in the knapsack. That is,

$$x_j = \begin{cases} 1 & \text{if } j \in S \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

Thus, we want to maximize $\sum_{j=1}^{n} c_j x_j$ subject to the condition that $\sum_{j=1}^{n} a_j x_j \le b$.

Let $\phi_{s,k}$ denote the minimum value of $\sum_{j=1}^{k} a_j x_j$ subject to the restriction that $\sum_{j=1}^{k} c_j x_j \ge s$, i.e., the minimum total volume of items chosen such that the sum of their weights is at least s. The boundary conditions for this dynamic program are given by

$$\phi_{s,1} = \begin{cases} 0 & \text{if } s = 0 \\ a_1 & \text{if } 0 < s \le c_1 \\ \infty & \text{otherwise} \end{cases} \tag{5.2}$$

We now work out the dynamic programming recursion for $\phi_{s,k}$. Depending on the value of x_k, there are two possible cases:

Case 1: $x_k = 0$, i.e., item k is not chosen. In this case, it is easy to see that $\phi_{s,k} = \phi_{s,k-1}$.

Case 2: $x_k = 1$, i.e., item k is chosen. This is possible only if $s \geq c_k$. Otherwise, there is no feasible solution in which $\sum_{j=1}^{k} c_j x_j \geq s$ and $x_k = 1$, in which case only Case 1 applies. Hence, assume that $s \geq c_k$. In this case, the sum of the weights of the items chosen from $\{1, 2, \ldots, k-1\}$ must be at least $s - c_k$. Hence, it follows that $\phi_{s,k} = \phi_{s-c_k,k-1} + a_k$.

Combining the above two cases, we have

$$\phi_{s,k} = \begin{cases} \phi_{s,k-1} & \text{if } s < c_k \\ \min(\phi_{s,k-1}, \phi_{s-c_k,k-1} + a_k) & \text{otherwise} \end{cases} \tag{5.3}$$

The answer to our original knapsack problem is:

$$\max\{s \mid \phi_{s,n} \leq b\}$$

Note that for a fixed k, we need to compute $\phi_{s,k}$ for at most $\sum_{j=1}^{k} c_j$ values of s. Thus, we need to compute at most $n \sum_{j=1}^{n} c_j$ entries $\phi_{s,k}$. Computation of each such entry takes $O(1)$ time, so that the running time of the algorithm is $O(n \sum_{j=1}^{n} c_j)$, which is not polynomial in the size of the input.

We now show how the computation associated with the above dynamic program can be reduced to polynomial time under the L-bit precision model. Assume that each c_j has a representation with a precision of L bits, i.e., $c_j = d_j 2^{t_j}$, with $d_j < c^* = 2^L$ for all $1 \leq j \leq n$.

We first sort the c_j's in non-increasing order of t_j's, so that $t_1 \geq t_2 \geq \cdots t_n$. Let $s_i = \sum_{j=i}^{n} c_j$. That is, s_i is the sum of the weights of the items $i, i+1, \ldots, n$. Before we proceed further, let us derive an upper bound on $s_i / 2^{t_i}$.

$$\begin{aligned} s_i &= \sum_{j=i}^{n} d_j 2^{t_j} \\ &< \sum_{j=i}^{n} c^* 2^{t_j} \quad \text{since } d_j < c^* \text{ for all } j \end{aligned}$$

$$\begin{aligned} &\leq c^* \sum_{j=i}^{n} 2^{t_i} \quad \text{since } t_j \leq t_i \text{ for all } j > i \\ &\leq c^* n 2^{t_i} \\ \Rightarrow \frac{s_i}{2^{t_i}} &< c^* n \end{aligned}$$

We compute the $\phi_{s,k}$'s in increasing order of k. For a fixed k, consider the stage when we start computation of the $\phi_{s,k}$'s for different values of s. We analyze the number of values of s for which we need to compute $\phi_{s,k}$ for this fixed k. Let s^* be the largest value of s for which $\phi_{s,k-1} \leq b$. Since we are computing the $\phi_{s,k}$'s in increasing order of k, we know s^* at this stage of the computation. We now observe the following two facts.

Fact 1: There is a feasible solution to the knapsack problem with $\sum_{j=1}^{n} c_j x_j$ equal to s^*. Hence, the value of $\sum_{j=1}^{n} c_j x_j$ for the optimal solution is at least s^*. It is also easy to see that $(s^* + s_k)$ is an upper bound on the value of $\sum_{j=1}^{n} c_j x_j$ for the optimal solution. The (possibly infeasible) solution that attains this bound is obtained by adding all the items $k, k+1, \ldots, n$ to the solution with value s^*. Thus, we are only concerned with values of s in the range $s^* \leq s \leq s^* + s_k$.

Fact 2: Because of the non-increasing order of the t_i's, the sum of any subset of the c_j's from $\{c_1, c_2, \ldots, c_k\}$ is a multiple of 2^{t_k}. Thus, each value of s that we need to consider must also be a multiple of 2^{t_k}.

Due to the above two facts, the number of different values of s for which we need to compute $\phi_{s,k}$ for a fixed k is $s_k/2^{t_k} < c^* n$. This implies that the total number of table entries $\phi_{s,k}$ that we need to compute is $c^* n^2$, and the running time of the dynamic program becomes $O(c^* n^2)$.

Theorem 5.2.1 *In the L-bit precision model, the knapsack problem can be solved in $O(c^* n^2)$ polynomial time, where $c^* = 2^L$. The running time is strongly polynomial when L is fixed (fixed-precision model) or $L = O(\log n)$ (logarithmic-precision model).*

5.3 The k-Partition Problem.

The *k-partition* problem is defined as follows:

> Given a set $A = \{a_1, a_2, \ldots, a_n\}$ of n numbers such that $\sum_{i=1}^{n} a_i = kb$, is there a partition of A into k subsets $A_1, A_2, \ldots, A_k$ such that $\sum_{a_i \in A_j} a_i = b$ for all $1 \leq j \leq k$?

This problem is $\mathcal{NP}$-complete (even for a fixed k) [5]. The standard dynamic program for solving this problem is as follows. Let $\phi_{w_1,w_2,\ldots,w_k,j}$ be 1 (and 0 otherwise) if there exist a partition of $\{a_1, a_2, \ldots, a_j\}$ into k subsets, where the sum of the numbers in the i^{th} partition is w_i for $1 \leq i \leq k$. Obviously, the answer to the k-partition problem is 'Yes' if and only if $\phi_{b,b,\ldots,b,n} = 1$. The boundary conditions for this dynamic program are given by

$$\phi_{w_1,w_2,\ldots,w_k,1} = \begin{cases} 1 & \text{if } \exists\ i \text{ such that } w_i = a_1 \text{ and } w_j = 0\ \forall\ j \neq i \\ 0 & \text{otherwise} \end{cases} \tag{5.4}$$

Consider any partition of $\{a_1, a_2, \ldots, a_j\}$ into k subsets where the sum of the numbers in the i^{th} partition is w_i for $1 \leq i \leq k$. In any such partition, there are k possible sets A_i $(1 \leq i \leq k)$ into which element a_j can be placed. Of course, element a_j can be placed in set A_i only if $w_i \geq a_j$. Otherwise, there is no feasible solution in which the sum of the elements in the set A_i is w_i and element a_j is placed in set A_i. Hence, the dynamic programming recursion is:

$$\phi_{w_1,w_2,\ldots,w_k,j} = \min\{\phi_{w_1,\ldots,w_i-a_j,\ldots,w_k,j-1} \mid w_i \geq a_j \text{ and } 1 \leq i \leq k\}$$

Thus, we need to compute at most $\begin{pmatrix} kb+k-1 \\ k-1 \end{pmatrix}$ entries $\phi_{w_1,w_2,\ldots,w_k,j}$. Computation of each such entry takes $O(k)$ time, so that the running time of the algorithm is $O\left(k \begin{pmatrix} kb+k-1 \\ k-1 \end{pmatrix}\right)$, which is not polynomial in the size of the input, even if

k is fixed.

We now show how the computation associated with the above dynamic program can be reduced to polynomial time under the L-bit precision model. Assume that each a_i has a representation with a precision of L bits, i.e., $a_i = c_i 2^{t_i}$, with $c_i < c^* = 2^L$ for all $1 \leq i \leq n$.

We first sort the numbers a_i in non-increasing order of t_i's, so that $t_1 \geq t_2 \geq \cdots t_n$. Let y_i be the sum of the numbers $a_{i+1}, a_{i+2}, \ldots, a_n$, i.e., $y_i = \sum_{j=i+1}^{n} a_j$. Before we proceed further, let us derive an upper bound on $y_i/2^{t_i}$.

$$\begin{aligned} y_i &= \sum_{j=i+1}^{n} c_j 2^{t_j} \\ &< \sum_{j=i+1}^{n} c^* 2^{t_j} \quad \text{since } c_j < c^* \text{ for all } j \\ &\leq c^* \sum_{j=i+1}^{n} 2^{t_i} \quad \text{since } t_i \geq t_j \text{ for all } j > i \\ &\leq c^* n 2^{t_i} \\ \Rightarrow \frac{y_i}{2^{t_i}} &< c^* n \end{aligned}$$

The following two facts are crucial to the development of the polynomial time algorithm for the k-partition problem under the L-bit precision model.

Fact 1: Let w be the sum of a subset of numbers from $\{a_1, a_2, \ldots, a_i\}$. Since we are interested in a subset of $\{a_1, a_2, \ldots, a_n\}$ that sums to b, we will need to consider the value w only if $w + y_i \geq b$. Otherwise, if $w + y_i < b$, we will never reach the sum of b, even if we add all the remaining elements from $\{a_{i+1}, a_{i+2}, \ldots, a_n\}$ to this subset. Thus, when we consider subsets of numbers summing to w from the set $\{a_1, a_2, \ldots, a_i\}$, we need to consider at most y_i values of w, i.e., $b - y_i \leq w \leq b$.

Fact 2: Because of the non-increasing order of the t_i's, the sum of any subset of numbers from $\{a_1, a_2, \ldots, a_i\}$ is a multiple of 2^{t_i}.

Now, let us consider the table entry $\phi_{w_1,w_2,\ldots,w_k,i}$. Due to Fact 1, we are only interested in at most y_i different values for each of $w_1, w_2, \ldots, w_k$. Also, by Fact 2, we should only consider values that are are multiples of 2^{t_i}. Hence, there at most $y_i/2^{t_i} < c^*n$ values for each of $w_1, w_2, \ldots, w_k$ that we need to consider.

Thus, the total number of table entries that we need to compute is at most $n(c^*n)^k$. Computation of each such table entry takes $O(k)$ time, so that the total running time of the algorithm is $O(kn(c^*n)^k)$ which is polynomial in c^* and n for a fixed k.

Theorem 5.3.1 *In the L-bit precision model, the k-partition problem can be solved in $O(kn(c^*n)^k)$ polynomial time for a fixed k, where $c^* = 2^L$. The running time is strongly polynomial when L is fixed (fixed-precision model) or $L = O(\log n)$ (logarithmic-precision model).*

5.4 Scheduling to Minimize Makespan on Identical Parallel Machines.

In this section, we consider the problem of scheduling n jobs on any fixed number m of identical parallel machines in order to minimize the makespan. Each job j has a processing time p_j on any machine. In scheduling notation, this problem is denoted as $Pm| \,|C_{max}$.

The decision problem formulation of $Pm| \,|C_{max}$ is defined as follows:

> Given a set of n independent jobs, $N = \{J_1, \ldots, J_n\}$, with processing times p_j, $\forall\ 1 \leq j \leq n$, m identical parallel machines, and a number b, is there a schedule of the n jobs on the m machines such that the makespan on each machine is at most b ?

This problem is $\mathcal{NP}$-complete [5]. We first discuss a dynamic program to solve this problem when the p_j's are arbitrary. We then show how the running time of this dynamic program can be improved when the p_j's have L bits of precision.

Let $\phi_{s_1,s_2,\ldots,s_k,j}$ be 1 (and 0 otherwise) if there exists a schedule of the jobs $\{1,2,\ldots,j\}$ on m machines such that the makespan on machine k is at most s_k, for all $1 \le k \le m$. Obviously, the answer to the decision version of $Pm|\ |C_{max}$ is 'Yes' if and only if $\phi_{b,b,\ldots,b,n} = 1$.

The boundary condition for this dynamic program is given by

$$\phi_{s_1,s_2,\ldots,s_m,1} = \begin{cases} 1 & \text{if } \exists\ k \text{ such that } s_k \ge p_k \\ 0 & \text{otherwise} \end{cases} \tag{5.5}$$

Consider any schedule of $\{1,2,\ldots,j\}$ on m machines in which the makespan on machine k is at most s_k for all $1 \le k \le m$. In any such partition, there are m possible machines on which job j can be scheduled. If job j is scheduled on machine k, then the total processing time of the other jobs scheduled on machine k must be at most $s_k - p_j$. Of course, job j can be scheduled on machine k only if $s_k \ge p_j$. Otherwise, there is no feasible solution in which the makespan on machine k is at most s_k and job j is scheduled on machine k. Hence, the dynamic programming recursion is:

$$\phi_{s_1,s_2,\ldots,s_m,j} = \max\{\phi_{s_1,\ldots,s_k-p_j,\ldots,s_m,j-1} \mid s_k \ge p_j \text{ and } 1 \le k \le m\}$$

Since we are only interested in values of s_k which are less than or equal to b for all $1 \le k \le m$, the number of table entries that we need to compute is at most $O(nb^m)$. Computation of each such entry takes $O(m)$ time, so that the running time of the algorithm is $O(nmb^m)$.

We now show how the computation associated with the above dynamic program can be reduced to polynomial time under the L-bit precision model. Assume that each p_j has a representation with a precision of L bits, i.e., $p_j = c_j 2^{t_j}$, with $c_j < c^* = 2^L$ for all $1 \le j \le n$.

We first sort the p_j's in non-increasing order of t_j's, so that $t_1 \ge t_2 \ge \cdots t_n$. Let $s_i = \sum_{j=i+1}^{n} p_j$. That is, s_i is the sum of the processing times of the jobs $i+1, i+$

$2, \ldots, n$. We derive an upper bound on $s_i/2^{t_i}$ below.

$$\begin{aligned} s_i &= \sum_{j=i+1}^{n} c_j 2^{t_j} \\ &< \sum_{j=i+1}^{n} c^* 2^{t_j} \quad \text{since } d_j < c^* \text{ for all } j \\ &\leq c^* \sum_{j=i+1}^{n} 2^{t_i} \quad \text{since } t_j \leq t_i \text{ for all } j > i \\ &\leq c^* n 2^{t_i} \\ \Rightarrow \frac{s_i}{2^{t_i}} &< c^* n \end{aligned}$$

Consider the table entry $\phi_{s_1,s_2,\ldots,s_m,j}$. Observe the following two facts about the possible values of the s_k's that we need to consider.

Fact 1: Since we want to achieve a makespan of at most b on every machine, it is sufficient to consider values of s_k greater than or equal to $b - y_j$ for all $1 \leq k \leq m$. This is because the sum of the processing times of the jobs among $\{j+1, \ldots, n\}$ scheduled on machine k can be at most y_j.

Fact 2: Because of the non-increasing order of the t_i's, the makespan of any subset of jobs from $\{1, 2, \ldots, j\}$ is a multiple of 2^{t_j}.

Due to Fact 1, we are only interested in at most y_j different values for each of $s_1, s_2, \ldots, s_m$. Also, by Fact 2, we need to consider only values of s_k which are multiples of 2^{t_j}. Hence, the number of different values of s_k that we need to consider for each $1 \leq k \leq m$ is at most $y_j/2^{t_j} < c^* n$.

Thus, we need to compute at most $n(c^* n)^m$ table entries $\phi_{s_1,s_2,\ldots,s_m,j}$. Computation of each such entry takes $O(m)$ time, so that the overall running time of the algorithm is $O(nm(c^* n)^m)$, which is polynomial in c^* and n for a fixed m.

Theorem 5.4.1 *In the L-bit precision model, the decision version of $Pm| |C_{max}$ can be solved in $O(nm(c^* n)^m)$ polynomial time, where $c^* = 2^L$. The running time is* *strongly polynomial when* L *is fixed (fixed-precision model) or* $L = O(\log n)$ *(logarithmic-precision model).*

Bibliography

[1] Yair Bartal, Stefano Leonardi, Alberto Marchetti-Spaccamela, Jiří Sgall, and Leen Stougie. Multiprocessor scheduling with rejection. In *Proceedings of the 7th ACM-SIAM Symposium on Discrete Algorithms*, pages 95–103, 1996.

[2] C. Chekuri, R. Motwani, B. Natarajan, and C. Stein. Approximation techniques for average completion time scheduling. In *Proceedings of the 8th ACM-SIAM Symposium on Discrete Algorithms*, pages 609–618, January 1997.

[3] D. W. Engels, D. R. Karger, S. G. Kolliopoulos, S. Sengupta, R. N. Uma, and J. Wein. Techniques for Scheduling with Rejection. In G. Bilardi, G. F. Italiano, A. Pietracaprina, and G. Pucci, editors, *Algorithms – ESA'98*, volume 1461 of *Lecture Notes in Computer Science*, pages 490 – 501. Springer, Berlin, 1998. Proceedings of the 6th Annual European Symposium on Algorithms.

[4] D. W. Engels, D. R. Karger, S. G. Kolliopoulos, S. Sengupta, R. N. Uma, and J. Wein. Techniques for Scheduling with Rejection. Submitted to the *Journal of Algorithms.*

[5] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness.* W.H. Freeman and Company, New York, 1979.

[6] M. Goemans. Improved approximation algorithms for scheduling with release dates. In *Proceedings of the 8th ACM-SIAM Symposium on Discrete Algorithms*, pages 591–598, 1997.

[7] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[8] L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein. Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Mathematics of Operations Research*, (3):513–544, August 1997.

[9] L. A. Hall, D. B. Shmoys, and J. Wein. Scheduling to minimize average completion time: Off-line and on-line algorithms. In *Proceedings of the 7th ACM-SIAM Symposium on Discrete Algorithms*, pages 142–151, January 1996.

[10] E. L. Lawler. Scheduling a single machine to minimize the number of late jobs. Preprint, Computer Science Division, Univ. of California, Berkeley, 1982.

[11] E. L. Lawler and J. M. Moore. A functional equation and its application to resource allocation and sequencing problems. In *Management Science*, volume 16, pages 77–84, 1969.

[12] E. L. Lawler and D. B. Shmoys. Chapter 5: Weighted number of late jobs (preliminary version). To appear in: J.K. Lenstra and D.B. Shmoys (eds.) Scheduling, Wiley.

[13] Maxwell. Personal communication. 1996.

[14] Alix Munier, Maurice Queyranne, and Andreas S. Schulz. Approximation bounds for a general class of precedence constrained parallel machine scheduling problems. In R. E. Bixby, E. A. Boyd, and R. Z. Ríos-Mercado, editors, *IPCO: 6th Integer Programming and Combinatorial Optimization Conference*, volume 1412 of *Lecture Notes in Computer Science*, pages 367 – 382. Springer, Berlin, 1998.

[15] I. M. Ovacik and R. Uzsoy. *Decomposition Methods for Complex Factory Scheduling Problems.* Kluwer Academic Publishers, 1997.

[16] C. Phillips, C. Stein, and J. Wein. Scheduling jobs that arrive over time. In *Proceedings of Fourth Workshop on Algorithms and Data Structures, Lecture Notes in Computer Science, 955*, pages 86–97, Berlin, 1995. Springer-Verlag. Journal version to appear in Mathematical Programming B.

[17] M. H. Rothkopf. Scheduling independent tasks on parallel processors. In *Management Science*, volume 12, pages 437–447, 1966.

[18] A. S. Schulz and M. Skutella. Random–based scheduling: New approximations and LP lower bounds. In J. Rolim, editor, *Randomization and Approximation Techniques in Computer Science*, volume 1269 of *LNCS*, pages 119 – 133. Springer, Berlin, 1997. Proceedings of the International Workshop RANDOM'97.

[19] A. S. Schulz and M. Skutella. Scheduling–LPs bear probabilities: Randomized approximations for min–sum criteria. In R. Burkard and G. Woeginger, editors, *Algorithms – ESA '97*, volume 1284 of *LNCS*, pages 416 – 429. Springer, Berlin, 1997. Proceedings of the 5th Annual European Symposium on Algorithms.

[20] Steve Seiden. More multiprocessor scheduling with rejection. Technical Report TR Woe-16, Institut für Mathematik B, TU Graz, 1997.

[21] W.E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3:59–66, 1956.

[22] E. L. Lawler. Optimal sequencing of a single machine subject to precedence constraints. *Management Science 19*, pages 544–546.

[23] Peter Brucker. *Scheduling Algorithms.* Springer-Verlag, Germany, 1998.

[24] E. L. Lawler, J. K. Lenstra, A. H. G. Rinooy Kan, and D. B. Shmoys. Sequencing and Scheduling: Algorithms and Complexity. In S. C. Graves, A. H. G. Rinooy Kan, and P. H. Zipkin, editors, *Handbooks in Operations Research and Management Science, Vol. 4, Logistics of Production and Inventory*, pages 445–522. North-Holland, 1993.

[25] R. K. Ahuja and J. B. Orlin. Inverse Optimization, Part I: Linear Programming and general problem. *Working paper 4002, MIT Sloan School of Management*, Cambridge, MA, January 1998.

[26] R. K. Ahuja and J. B. Orlin. Inverse Optimization, Part II: Network Flow Problems. *Working paper 4003, MIT Sloan School of Management*, Cambridge, MA, February 1998.

[27] R. K. Ahuja and J. B. Orlin. Combinatorial algorithms for inverse network flow problems. *Working paper 4004, Sloan School of Management*, Cambridge, MA, February 1998.

[28] IEEE. IEEE standard for binary floating-point arithmetic. *SIGPLAN Notices*, 22:2, pages 9–25.

[29] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *Computing Surveys*, 23:1, pages 5–48.

[30] W. J. Cody, J. T. Coonen, D. M. Gay, K. Hanson, D. Hough, W. Kahan, R. Karpinski, J. Palmer, F. N. Ris, and D. Stevenson. A proposed radix- and word-length-independent standard for floating-point arithmetic. *IEEE Micro*, 4:4, pages 86–100.

[31] G. R. Bitran, V. Chandru, D. E. Sempolinski, and J. F. Shapiro. Inverse Optimization: An application to the capacitated plant location problem. *Management Science*, Vol. 27, pages 1120–1141, 1981.