New Utilization Criteria for Online Scheduling

Dissertation

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

der Universität Dortmund

am Fachbereich Informatik

von

Mohamed Hussein

Dortmund

2005

Tag der mündlichen Prüfung: 18.07.2005

Dekan: Prof. Dr. Bernhard Steffen

Gutachter: Prof. Dr.-Ing. Uwe Schwiegelshohn
Prof. Dr. Ingo Wegener

Abstract

In the classical scheduling problems, it has been assumed that complete knowledge of the problem was available when it was to be solved. However, scheduling problems in the real world face the possibility of the lack of the knowledge. Uncertainties frequently encountered in scheduling environments include the appearance of new jobs and unknown processing times. In this work, we take into account these realistic issues.

This thesis deals with the problem of non-preemptive scheduling independent jobs on m identical parallel machines. In our online model, the jobs are submitted over time non-clairvoyantly. Therefore, the processing times of the jobs are unknown until they complete. Further, we assume that the ratio of weight to processing time is equal for all jobs, that is, all jobs have the same priorities. The jobs are assigned to the machines in a nondelay fashion. Our main scheduling objective is to maximize the utilization of the system.

We show that the commonly used makespan criterion usually cannot reflect the true utilization of this kind of online scheduling problems. For this reason, it is very important to find another criterion capable of evaluating system utilization. Therefore, we introduce two new alternative criteria that more accurately capture the utilization of the machines. Moreover, we derive competitive factors for both criteria. Those competitive factors are tight for one criterion and almost tight for the other. Finally, we present an experimental investigation to evaluate the performance of the nondelay online algorithm with respect to our criteria. The experimental results show the confirmation of our theoretical results.

keywords: Nonclairvoyant, Online scheduling, Scheduling criteria.

Contents

1 Introduction **1**

1.1 Motivation and Model 5

1.2 How to read this thesis 9

1.3 Classification of Scheduling Problems 10

1.4 Online Paradigms 17

1.4.1 Jobs arriving one by one 18

1.4.2 Jobs arriving over time 19

1.4.3 Scheduling with rejection (Interval Scheduling) 21

1.5 Competitive Analysis 22

1.6 Alternative techniques for analyzing online algorithms 24

1.7 History and List Scheduling 26

1.8 Practical Examples of online models 29

2 Criteria for system utilization **33**

2.1 Is makespan suitable for utilization? 34

2.2 New criteria for machine utilization 37

2.3 Basic Job Systems 44

2.4 Transformation into Basic job system 47

3 Online scheduling to maximize utilization **55**

3.1 introduction 55

3.2 Scheduling jobs online with unknown size 57

3.3 Productive interval of machines 59

3.4 An upper bound for the utilization 61

4 Online scheduling to minimize equal priority completion time 77

4.1 introduction 77
4.2 Related Results 79
4.3 Jobs with Arbitrary Priority 83
4.4 The upper bound of the off-line problem 86
4.5 An upper bound of equal priority completion time 94
4.6 The applicability of equal priority flow time 100

5 Experimental Study 105

5.1 Experimental Design 105
5.1.1 Computing Environment 105
5.1.2 Benchmark Instances 106
5.2 An approach for optimal solution 108
5.3 Analysis of the Results 110

6 Conclusion 123

A Near-Optimal Algorithm 127

B Additional Experimental Results 129

Bibliography 133

List of Figures

1.1 *Example Gantt-chart for three parallel machines* 3

1.2 *Possible transmission route for messages between computers A and B in a network* 31

2.1 *Comparison of Makespan and Utilization for 2 Schedules* 35

2.2 *Effect on Future Job Submissions on 2 Schedules* 35

2.3 *Nondelay schedule S_1 (left) and the optimal schedule (right) for τ_1* 40

2.4 *Nondelay schedule S_2 (left) and the optimal schedule (right) for τ_2* 43

2.5 *Basic job system and its basic nondelay schedule S* 46

3.1 *Optimal Schedule σ of τ (left) and new optimal schedule σ' of τ' (right) when $p_{j_2'} > r - r_{j_2}$.* 69

3.2 *Basic Schedule S of τ (left) and new basic S' of τ' (right) when $p_{j_2'} > r - r_{j_2}$* . 69

3.3 *Optimal Schedule σ of τ (left) and new optimal schedule σ' of τ' (right) when $p_{j_1'} = 0$.* 70

3.4 *Basic Schedule S of τ (left) and new basic schedule S' of τ' (right) when $p_{j_1'} = 0$.* 70

3.5 *The transformation process when $m_S \leq m_r$. Left: Basic schedule S of job system τ. Right: A resulting basic nondelay schedule S' of the new job system τ'.* 73

3.6 *Transformation process from the optimal schedule σ (left) of job system τ into the new optimal schedule σ' (right) of job system τ' when $m_S \leq m_r$.* 73

3.7 *An illustration of the transformation process when $m_S > m_r$. Left: Basic schedule S of job system τ. Right: A new basic nondelay schedule S' of job system τ'.* 75

3.8 *Transformation process from the optimal schedule σ (left) of job system τ into the new optimal schedule σ' (right) of job system τ' in case $m_S > m_r$.* 75

4.1 *The transformation of τ to rectangular job system $\bar{\tau}(m_t)$.* 89
4.2 *The transformation of τ with a large number of short jobs.* 92
4.3 *Illustration of the primary transformation process from schedule S (left) of τ into schedule S' (right)of τ'* 97
4.4 *Illustration of the primary transformation process from the optimal schedule σ of τ into the optimal schedule σ' of τ'.* 97
4.5 *Illustration of the generation process of schedule S'' (right) of the new job system τ'' from schedule S' (left) of job system τ'.* 99
4.6 *Illustration of the generation process of the optimal schedule σ'' (right) of the new job system τ'' from the optimal schedule σ' (left) of job system τ'.* 99

5.1 *The competitive ratio of the utilization and equal priority completion time with different number of jobs when $m = 3$.* 112
5.2 *The competitive ratio of the utilization and equal priority completion time with different number of jobs when $m = 5$.* 113
5.3 *The competitive ratio of the utilization and equal priority completion time with different number of jobs when $m = 10$.* 114
5.4 *The competitive ratio of the utilization and equal priority completion time with different number of jobs when $m = 15$.* 115
5.5 *The competitive ratio of the utilization and equal priority completion time with different number of jobs when $m = 20$.* 116

B.1 *The ratios between the competitive ratio of the utilization and the competitive ratio of the equal priority completion time when $m = 3$.* 129
B.2 *The ratios between the competitive ratio of the utilization and the competitive ratio of the equal priority completion time when $m = 5$ (top) and $m = 10$ (bottom).* 130

B.3 *The ratios between the competitive ratio of the utilization and the competitive ratio of the equal priority completion time when $m = 15$ (top) and $m = 20$ (bottom).* 131

List of Tables

2.1 *Comparison between Utilization, Equal Priority Completion Time, and Makespan for Job Systems τ_1 and τ_2.* 44

4.1 Comparison between Utilization, Equal Priority Completion Time, and Equal Priority Flow Time for Selected Schedules 102

5.1 *The competitive ratios of the utilization criterion for exponentially distributed job processing times.* 118
5.2 *The competitive ratios of the utilization criterion for job instances generated by chi-square distribution.* 119
5.3 *The competitive ratios of the utilization criterion for job instances generated by using log-normal distribution.* 119
5.4 *The competitive ratios of the equal priority completion time criterion under exponentially distributed job processing times.* 120
5.5 *The competitive ratios of the equal priority completion time criterion for job instances generated by chi-square distribution.* 121
5.6 *The competitive ratios of the equal priority completion time criterion for job instances generated by using log-normal distribution.* 121

Acknowledgments

I owe a huge debt of gratitude to my supervisor Prof. Dr. Uwe Schwiegelshohn for teaching me a lot about scheduling. I wish to express my deepest appreciation to him for his infinite guidance, insights throughout and continued dedication to this thesis. I am extremely grateful to him for sharing his invaluable ideas and his limitless enthusiasm for research. I will be indebted to him for all the ways in which I have grown as a researcher and especially for teaching me how to think. Clearly, this thesis would not be possible without him.

I take this opportunity to thank all colleagues at the Computer Engineering Institute in Dortmund university for creating a warm and friendly atmosphere in which I could work at this thesis.

Last but not least I would like to thank my wife, Amal, for her unlimited patience, endless love, and ongoing encouragement.

Author's Contribution

The work of this dissertation has been completely done by myself. However, the mathematical proofs of Theorem 3.1 in Chapter 3 and Theorem 4.1 in Chapter 4 have been performed in collaboration with my advisor Prof. Dr. Uwe Schwiegelshohn. The author equally contributed in both proofs. Information derived from the published and unpublished work of others has been acknowledge in the text, and a list of references is given.

Mohamed Hussein

Publications

- M. Hussein, U. Schwiegelshohn, "*Nonclairvoyant Online Scheduling of Jobs with Equal Priority*", submitted to the Journal of Theoretical Computer Science, 20 August 2004.
- M. Hussein, U. Schwiegelshohn, "*On an Online Scheduling Problems for Jobs with Equal Priority*", Technical Report No. 0203, Dortmund university, Faculty of electrical engineering and information technology, ISSN 0941-4169, Sep. 2003.

Chapter 1

Introduction

Scheduling is known as a decision-making process of allocating limited resources over time in order to perform a collection of competing activities for the purpose of optimizing certain objective functions, BAKER [5]. More precisely, the home of the majority of scheduling problems is the area of *combinatorial optimization*, and in fact, there are numerous combinatorial optimization problems that can be equivalently re-formulated as scheduling problems. Hence, there are at least two reasons to study scheduling problems: the great diversity of existing applications on the one hand, and the mathematical interest in the corresponding models on the other hand. Scheduling problems have been studied by researchers in various communities such as operations research, algorithms, and queueing theory. In fact, scheduling decisions occur only whenever there are more outstanding requests (activities) than the number of available resources or the activities have to be done on different types of resources. The output of this decision process will be the set of *task/resource/time* assignments.

Scheduling plays a crucial role in a wide variety of environments such as in most manufacturing and production systems as well as in most information-processing environments. In the current competitive environment, effective scheduling has become a necessity for survival in the marketplace. System owners have to schedule activities in such a way as to use the resources available in an efficient manner to get the highest possible of the resources utilization. The previous definition of BAKER [5] for scheduling is very general since the concept of resources and activities may take many different

forms because resources and activities may vary a lot. One can think of resources as being machines in a production environment and the activities being the operations that have to be performed on these machines; or the resources may be processors in a computing environment and the activities will be executions of computer programs. Also, the resources may be runways and activities may be take-offs and landings at an airport. Another application area of scheduling can be found in telecommunications systems. From a functional point of view, wireless links in the telecommunication network can equally be regarded as resources. This is a good example of resources which are not physical in the usual sense of the world. In these systems, activities are information or messages that have to be transmitted through the communication channels. Other examples can be found in the areas of personnel scheduling, transportation, maintenance scheduling, and other types of service industries.

Although the resources and activities have various shapes depending on the scenario, the resources of any scheduling problem can be modelled as *machines* and activities as *jobs* that have to be executed by these machines. There are two issues commonly associated with scheduling problems: how to allocate jobs to machines and how to sequence jobs on each machine. In other words, there are *allocation* decisions and *sequencing* decisions. Therefore, it is worth noting the distinction between a single machine schedule and a schedule for multi-machines. A single-machine schedule corresponds usually to a permutation of the job system or the order in which jobs are to be processed on a given single machine. While, a multi-machine schedule refers to an allocation of jobs within a more complicated setting of machines. This schedule is a description of when and on which machine to process the job satisfying the constraints. It is often visualized using a GANTT-CHART, an example of which can be found in Figure 1.1. Each column in the GANTT-chart represents a machine and each box represents a job. The jobs have been labelled with their processing requirements and the timetable information of each job can be read on the time-axis. Schedules are often categorized in the following classes:

- A valid schedule, in which no job can be finished earlier without changing the

processing order on any one of the machines, is termed *semi-active*,

- a valid schedule, in which no job can be completed earlier without delaying at least one other job, is termed *active*,
- a valid schedule, in which no machine is ever idle if a job, is ready to be processed on it is, called *non-delay*.

For more details of the relations between those schedules, we refer the reader to PINEDO [66]. The research in this thesis will focus on the analysis of non-delay schedules.

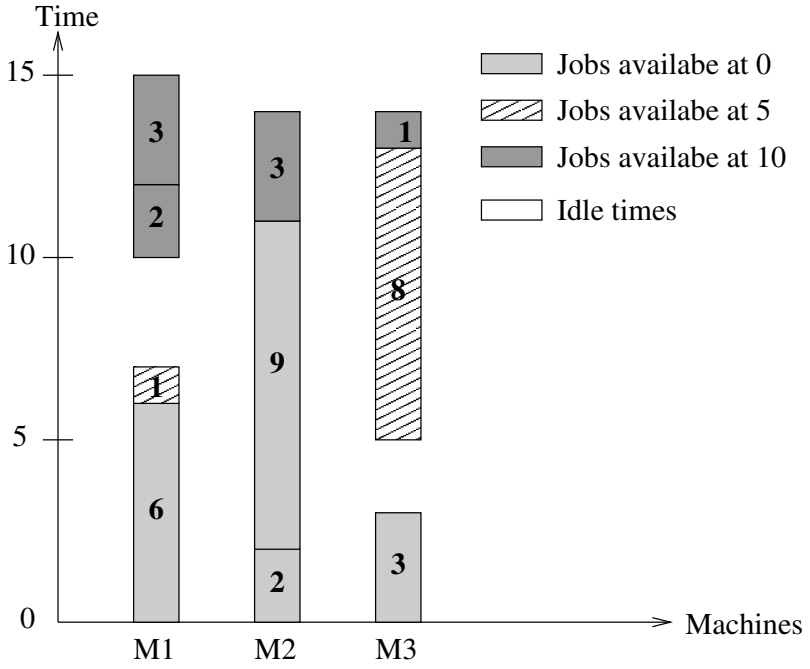

Figure 1.1: *Example Gantt-chart for three parallel machines*

The scheduling literature is full of very diverse scheduling problems [19, 66, 12]. The work in this thesis falls under the category of *online* scheduling problems. In online scheduling, the scheduler receives jobs that arrive over time, and generally must schedule the jobs without any knowledge about future submissions of the new jobs. Figure 1.1 shows such online scenario in which jobs have different arrival (available)

times. The lack of knowledge of the future generally prevents the scheduler from guaranteeing optimal schedules. Thus much research has been focused on finding scheduling algorithms that guarantee schedules that are in some way not too far from optimal. The online scenario may occur in situations that arise within the omnipresent *customer-server* setting. In a customer-server system, there are many customers and a few servers. Customers submit requests for service to the servers over time. Further, the servers are not aware of the arrival times of these requests in advance. In the language of scheduling, a server is a machine (processor), and a request is a job. In this thesis, we are interested to consider online models in which the scheduler is not aware of the future submission times. Moreover, our model has an additive hard restriction beside such online feature. In our scenario, we assume that the scheduler does not have any information about the processing requirements of the existing job (which is already accessed by the system) until the completion. As the main focus of this thesis goes to online scheduling problems with the uncertainty of the processing requirements, we will provide more details of such kind of problems in Section 1.4.

Why are most Scheduling Problems Difficult?

A scheduling problem may not be hard to formulate, but solving it is entirely another matter. It is well known that the difficulty of most scheduling problems is due to their computationally hard nature. In fact, the computational requirements for obtaining an optimal solution grows rapidly beyond reasonable bounds as the size of the problem increases. Formally speaking, most scheduling problems are notoriously $\mathcal{NP}$-hard or $\mathcal{NP}$-complete [20]. An $\mathcal{NP}$ *problem* (non-deterministically polynomial) is one that, in the worst case, requires time polynomial in the length of the input for solution by a non-deterministic algorithm [20, 61]. Non-deterministic algorithms are theoretical, idealized programs that somehow manage to guess the right answer and then show that it is correct. An $\mathcal{NP}$–complete problem is $\mathcal{NP}$ and at least as hard as every other $\mathcal{NP}$ problem. An $\mathcal{NP}$–hard problem is $\mathcal{NP}$-complete or harder than $\mathcal{NP}$. In the seminal work of KARP [42], the pervasive nature of $\mathcal{NP}$–completeness has been established. The author has shown that decision versions of several naturally occurring problems

in combinatorial optimization are $\mathcal{NP}$–complete, and thus are unlikely to have efficient (polynomial time) algorithms. The proofs of the $\mathcal{NP}$–completeness are available for a number of simple scheduling problems, and realistic problems tend to be even more complex. From that time onwards, much research concentrated on the classification of scheduling problems according to their computational complexity. The aim was to 'delineate' as closely as possible, the boundary between those machine scheduling problems which are *easy* (solvable in polynomial time) and those which are $\mathcal{NP}$–hard (see LAWLER [49]).

The practical implication of $\mathcal{NP}$-hardness is that the time required for the computation to find the optimal solution grows at least exponentially with the size of the problem [51, 61], *e.g.* the time increases exponentially with the number of jobs and may with the number of machines. Due to the $\mathcal{NP}$-hard nature of most scheduling problems, it is usually very difficult to find an optimal schedule. A very fruitful approach has been to relax the notion of optimality and settle for *near–optimal* solutions. A near–optimal solution is one whose objective function value is within some small *multiplicative* factor of the optimal value. The near-optimal scheduling algorithms spend time on improving the schedule quality but do not continue until the optimum is found. This led to the idea of the approximation algorithms that are heuristics and provide *provably good* guarantees on the quality of the solutions they return. This approach was pioneered by the influential paper of JOHNSON [35]. In this paper, the author showed the existence of good approximation algorithms for several $\mathcal{NP}$–hard optimization problems. HOCHBAUM's book [32] on approximation algorithms gives a good glimpse of the knowledge on that subject.

1.1 Motivation and Model

Our central motivating question is:

What is a good scheduling criterion for system utilization?

Clearly, the answer would depend on the particular scenario which we have. In parallel-machine models, makespan is usually used to evaluate system utilization. However,

we are interested to find alternative criteria that more accurately capture the utilization of the system for online scheduling models.

In the traditional theory of scheduling, it is often assumed that the scheduler has full information on the job as soon as the job accesses the system (*clairvoyant scheduling*). Recently, a more realistic theoretical model has been emerged (*nonclairvoyant online scheduling*) to consider the scheduling problems in which a job's data are not completely known until the job has completed. For example, no one knows the exact number of phone calls that are going to reach a switch-board during a certain period, nor do we know the exact length of each individual call. Similarly, we do not know the exact number of tasks that are going to be executed on a time-shared multi-user computer system. The notion of non-clairvoyant algorithm is intended to formalize the realistic scenario where the algorithm does not have the access to the whole input instance, unlike the clairvoyant algorithm. Instead, it learns the input piece by piece, and has to react to the new requests with only a partial knowledge of the input. The work in this thesis is devoted to consider such nonclairvoyant online scheduling problems so as to attain the highest resource utilization.

In real world, a following online scenario may occur in electronic commerce if no two jobs are allowed to share the same machine during execution due to security reasons. A system owner provides m identical parallel machines to his customers. These customers are independent from each other. They submit their jobs dynamically over time. A customer's job always uses its assigned machine exclusively for its whole execution time, that is, preemption is not allowed. The owner receives a fixed fee from a customer for each minute a job of this customer occupies a machine. The customers do not provide in advance the machine usage necessary to execute their jobs. Forcing the customers to provide the execution times of their jobs in advance may be a hassle to those customers and is typically of little help as experiments with users of parallel computers have shown that those estimates are very unreliable [50].

It is reasonable to assume that the system owner primarily tries to maximize system utilization. Clearly, saving a free machine, when some job is available, for a potential future request makes little sense as in our online model, the system owner has no in-

formation about any future machine request. Therefore, without additional knowledge on the jobs, no job selection strategy can guarantee a better schedule than any other. Consequently, the system owner immediately assigns a free machine to an open request, that is, he generates a nondelay schedule [66]. On the other hand, he postpones the assignment of a job to a machine until a machine becomes available. If several requests are open at the same time he may use any arbitrary policy to pick any one of them, that is, he uses a list scheduling algorithm. In the single release date case, it is shown that *Graham*'s non-idling list scheduling algorithms are appropriate for optimizing objectives that are related to maximizing machines utilization as in MUNIER *et al.* [60]. However, remember that no information about the machine occupation of any waiting job is available.

Most customers only submit jobs during core business hours. The owner of the machines usually wants to maximize utilization of his machines during core business hours and is aware that a large percentage of the machines may be idle during the rest of the day. He must decide on the best number of machines to install. If he provides only few machines he may achieve maximal utilization in the target time frame but he may also loose some revenue as potential customers will switch to a competitor rather than waiting for their jobs being completed much later during off business hours. Therefore, the owner will typically add new machines as long as his machines are not idle during those core hours. On the other hand, he may decide to cut his costs by removing some machines if there is unused capacity during this time.

However when making this decision, the owner must consider that some idleness may be produced by an unfortunate selection of jobs. Therefore, he may be interested in the ratio between the utilizations of schedules with an optimal and a worst case job selection.

In classical approach, a schedule is generated with the objective of optimizing one or more of the performance measures. Recently, KEMPF *et al.* [45] describe a number of different considerations that must be taken into account when assessing the quality of a schedule. The authors propose a different approach: *segmentation* and then aggregation of the metrics that are used to measure the performance of a schedule. *Segmentation* is

specifying classes of scheduling objects that form a meaningful unit (*e.g.* all drilling machines in a plant can form a segment). Once the segmentation of the scheduling objects has been specified (*i.e.* the scheduling system is divided into several segments), the metrics for each segment (*e.g.* utilization of machines in that segment) can be *aggregated* into one segment-wide metric. The authors propose to use different metrics for each segment, rather than using a single metric for the whole schedule. This is equivalent to apply a single metric for only a part of the schedule. For example, in production and manufacturing scheduling, a schedule that maximize utilization of the machines throughout the whole system may be of poor quality, because this schedule would increase work-in-progress inventory levels. Instead, a good schedule should maximize utilization of only the bottleneck machines (where high utilization is needed). For that reason, we consider the system utilization in certain target time interval (core business hours).

Formally, there is a job system τ consisting of a collection of independent jobs which arrive dynamically over time. Those jobs must be scheduled on a system with m identical parallel machines without preemption. The time when a job $j \in \tau$ arrives is its *release date* and is denoted by $r_j \geq 0$. Each job has a processing requirement, also known as its *size* and denoted by $p_j > 0$. Further, a job is not known before it is released and its processing time is only determined once the job has completed (*nonclairvoyant scheduling*). All jobs are weighted according to their importance, the weight of a job j is denoted by w_j. The scheduling objective is to attain the maximum utilization of those machines during a specific time interval.

No assignment of a waiting job to a machine need be made before the machine is actually available. We denote the completion time of job j in a schedule S by $C_j(S)$. Schedule S is legal if no machine is executing more than a single job at any time, no job starts before its release date r_j, and each job executes without interruption. Therefore, job j starts at time $C_j(S) - p_j$ in schedule S. As already mentioned, we only consider the analysis of the **nondelay** schedules, that is, no machine is ever idle if a job is ready to be processed on it.

1.2 How to read this thesis

The research is focused on the performance measures for the problem of non-clairvoyantly scheduling jobs that arrive over time on identical parallel machines. Our main goal is to maximize system utilization. The analysis of the nondelay, non-preemptive schedules is addressed.

Care has been taken to make this thesis as self-contained as possible. Because of this, the remainder of this chapter contains an amount of pages concerned with the general introduction and the fundamental concepts of the scheduling problems as well as with a detailed description of online scheduling models. The reader already familiar with these concepts can skip such part and go directly to the more interesting part starting with the next chapter. The remainder of the thesis is organized as follows.

In Chapter 2, we consider the problem to determine performance measures that are well suited to evaluate the utilization of the machines in a specific time interval. In the first section of this chapter, we show that the commonly used makespan criterion is not well suited for such evaluation. Therefore, in Section 2.2 we introduce two new alternative performance measures that are well suited to quantitatively represent machines utilization, particularly for online scheduling problems. These performance measures are named *utilization* and *equal priority completion time*. To support an evaluation of these two scheduling criteria, a basic job system and a basic nondelay schedule are defined in Section 2.3. In Section 2.4, we show that those basic job systems and their basic nondelay schedules suffice to determine the worst-case ratios for our criteria. Hence, we restricted ourself to consider only this kind of job systems.

Chapter 3 is devoted to consider the maximization problem of our first criterion (utilization). Then, the worst competitive factor of such criterion is derived. Further, we provide the proof of the tightness of this factor.

Chapter 4 is devoted to consider the equal priority completion time minimization problem. First, in Section 4.3, we show that the competitive ratio of the total weighted completion time criterion is unbounded when the jobs have arbitrary weights (*i.e.* jobs have different priorities). Our main result in this chapter depends on a result from

KAWAGUCHI and KYAN [44]. the authors investigated the total weighted completion time minimization problem when all jobs are available at the same time (offline problem). Therefore, in Section 4.4, we briefly provide the proof of their result using the notations and corollaries of this thesis. Section 4.5 gives the main result of this chapter. We prove an almost tight competitive factor of our second criterion (equal priority completion time) for our online model. At the end of this chapter, in Section 4.6, we discuss the possibility whether it is appropriate to use an equal priority flow time criterion, which can be modelled in the same fashion as the equal priority completion time.

Next, we provide an experimental investigation in Chapter 5. In this chapter, we analyze experimentally the performance of the nondelay online algorithm with respect to our new criteria. A detailed description of the experimental design is given in the first section of this chapter. Finally, in Section 5.3, we discuss the obtained results and report the analysis of the experiments.

The thesis ends with Chapter 6 that gives the implications of the results in this thesis.

1.3 Classification of Scheduling Problems

The theoretical side of scheduling deals with the detailed sequencing and scheduling of jobs. In standard machine scheduling models, a characteristic of the machine environment is that a machine can process no more than one job at a time and that each job may be processed by only one machine at a time. The common goal is to sequence a collection of given jobs which are going to be performed in a given machine environment and subject to given requirements (constraints), in such a way that one or more performance criteria are optimized. An allocation that satisfies the requirements imposed by the machine environment and the job characteristics is called a *feasible or valid* schedule, or *schedule* for short. If such schedule has the optimum value with respect to the optimality criterion, then we call it *optimal*.

Within the area of machine scheduling, there are still many different research branches.

This is caused by the presence of a virtually unlimited number of problem types in machine scheduling. Various machine environments subject to multiform constraints with several objective functions make a multiplicity of scheduling problems. Hence, there is an obvious need for modelling and classification of these problems. In fact, scheduling problems can be classified in many ways. A scheduling problem is called *off-line* if all jobs are ready concurrently at the same time or the arrival times of the jobs are known in advance. By contrast, in an *online* scheduling problem, all jobs are not available simultaneously but become available over time and their arrival times are not known beforehand. As we mentioned before, we restrict our attention to consider the online scheduling models.

The standard classification scheme which is very convenient to categorize scheduling problems is introduced by GRAHAM [25]. According to this scheme, the enormous different scheduling problems can be described by a triplet-field notation $\alpha|\beta|\gamma$. The first field α describes the machine environment and contains only a single entry. The β field provides details of the processing characteristics and constraints and may contain more than one entry or being empty . The third field γ usually contains only a single entry and describes the objective to be optimized. Therefore, the specification of a machine scheduling model requires the description of a *machine environment*, *job characteristics*, and an *optimality criterion*. In the remainder of this section, we give an overview of each one of these environments.

Machine environments

The simplest machine environment is the *single* machine model, which is denoted by a 1 in the α field. Here, each job j has to spent p_j units of time for the processing on a single machine. Although this environment is simple and usually a special case of all other complex environments, its study is important for various reasons. The results, that can be obtained for single machine models, do not only provide insights into single machine models, but also provide a basis for heuristics for more complicated machine environments. In practice, scheduling problems in more complicated environments are often decomposed into subproblems that deal with single machines. The

single machine problem often provides a support to model the behavior of a complex system and quite often appears as an elementary component in a larger scheduling problem. Sometimes the basic single machine problem is solved independently, and then incorporates the results into the main problem [65]. Initial attempts to solve simple problems have paved the way to complex problems involving multiple objectives and constraints (GUPTA [27]).

A natural generalization of the single machine model is the *parallel* machine environment. The similarity between single and parallel machine environments is that every job requires only a single operation for processing. In the parallel machine model, each job has to spend its processing requirement on any one of m machines. A parallel machine environment can be *identical*, *uniform*, or *unrelated* machines. In the identical machines case, denoted by P, the m machines operate at the same speed. Therefore, the time p_{ij} that job j spends on machine i is independent of the machine and denoted by p_j. In the uniform machines case, denoted by Q, each machine i has its own speed v_i. Therefore, the processing time p_{ij} of job j on machine i is equal to $\frac{p_j}{v_i}$ (assuming machine i execute job j completely). The unrelated machines case is a generalization of the uniform machines environment. There are m parallel machines with different speeds while the speed of any machine is job-dependent. The speed of machine i is indicated by v_{ij}. Therefore, the time p_{ij} that the machine i requires to execute the job j is equal to $\frac{p_j}{v_{ij}}$ (again assuming job j processes completely on machine i).

In the machine environments that we have mentioned so far, every job consisted of a single operation. In contrast to these single-operation environments, we have the so called *shop* environment. In shop machines environment, there are m machines in series while every job consists of several operations. Each operation of a job has to be processed on a designated machine i for p_{ij} units of time. Further, no job can undergo more than one operation at a time, that is, no two operations of the same job can be processed simultaneously. There are three different types of the shop environment: *open*, *job*, and *flow* shop. As this environment is out the scope of this work, we do not explain it in details. For a deep description, we refer the reader to the book of PINEDO [66]

In general, the number of machines m is assumed to be either a part of the input data or a fixed constant. To distinguish the difference, in the last case, the letter m has to appear after the machine environment. For example, P_m refers to an identical parallel machines problem with fixed number of machines.

Job data and job characteristics

In scheduling problems, there are some pieces of data that may be associated with any job j. These data are the *processing* time, *release* date, *due* date, and the *weight*. The p_{ij} represents the processing time of job j on machine i. The subscript i is omitted if the processing time of job j does not depend on the machine or if job j is only to be processed on one given machine. Job availability may be restricted by imposing *release* date r_j, it may also be referred to as *ready* date. In this case the job is only available for processing from time r_j onwards, that is, before this time no processing of the job can take place (*i.e.* the earliest time at which job j can start its processing). The *due* date of job j, indicated by d_j, is the date at which job j should be completed (the date the job is promised to the customer). Although it is allowed that a job completes after its due date, a penalty should be incurred in this case. When the completion time of a job must met the due date, then this due date is referred to as a *deadline* and indicated by $\bar{d}_j$. The *weight* of job j, denoted by w_j, is essentially a priority factor, indicating the importance of this job relative to the other jobs in the system. Fore example, the weight may represent the actual cost of keeping the job in the system.

The second field β in the notation consists of the details of the job characteristics. In contrast to the first field α, this field may contain multiple entries or no entry at all. The job characteristics include the possibility of allowing *preemption* and of specifying *precedence constraints* or other restrictions. Traditional scheduling problems can be also categorized into two models with respect to preemptions. If *preemption* is allowed, denoted by *prmp*, then an operation may be interrupted arbitrarily at any moment in time and execute an other operation on the machine instead. The amount of processing a preempted operation already has received is not lost. When a preempted operation is resumed afterwards, at the interrupted time on another machine or at a later time on

any machine, it needs only a machine for its remaining processing time without any penalty. Therefore, processing an operation on different machines is possible, provided that this is done in non-overlapping time periods. This model is called *preempt-resume* model. An example for this model, is the scheduling of processes in a time-sharing operating system. There is another possibility for the preemption, *preempt-repeat* model, a running job may be stopped and later *restarted* from the beginning on the same or a different machine. That is, the work done on that job is completely lost. Thus in order to complete, a job has to be assigned to the same machine for its whole processing time without an interruption. This possibility can be denoted by *prmp-restart* in the middle field of the three-field notation. Scheduling a sound recording studio is an example for the preempt-restart model. A recording of a song could be interrupted. However, the entire song must be recorded again. On the other hand, if preemption is not allowed, an operation, once started, must be processed until completion without interruption. An example of a scheduling problem in this case is the car rental problem. After a customer takes off with a car, the car cannot be recalled and rented to another customer. The car can be rented to another customer only when the first customer returns it. Any no-preemption schedule is itself a preempt-restart schedule. Any preempt-restart schedule can be converted into a no-preemption schedule by eliminating all preempted executions in the preempt-restart schedule. Clearly, the elimination does not affect the completion of any job. For off-line problems, the no-preemption and the preempt-restart models are equivalent as off-line schedulers have all the information about the input instance up front and can perform any conversion before producing the output. However, in the on-line setting, the no-preemption and the preempt-restart models are different. Generally, considering the non-preemptive version of a problem is more difficult than the preemption version. In this research, we will consider the analysis of non-preemptive online schedules.

There might be an order imposed on the jobs, in this case we say that the jobs are subject to some precedence constraints. *Precedence* constraint, denoted by *prec*, may appear in a single machine or in a parallel machine environment, it stipulates that a certain job cannot start before another one or some jobs have completed. There are

several special forms of the precedence constraints. It can be *chains*, in which each job has at most one successor and at most one predecessor. An *intree* constraint occurs when each job has at most one successor. If each job has at most one predecessor, then it is referred to as an *outtree* constraint.

Optimality criteria

In classical scheduling theory, the objective is generally optimizing system performance. The performance of a given schedule is generally assessed by some measures. The performance measure to be optimized is usually a function of the *completion times* of the jobs, which clearly depend on the schedule. Also, the optimality criterion may depend on the due dates of the jobs. There are various performance measures considered in scheduling research. All measures can be classified primarily into two groups: those that are *regular* performance measures and those that are *non-regular* performance measures. The basic concept of regular performance measures is that the change of the optimal value depends on the change of at least one of the completion times of the jobs. More precisely, it is a non-decreasing function in the completion times of the jobs. That is, if any single job is made to complete later, the performance measure value stays the same or increases but never decreases. Otherwise, it is called a non-regular measure.

The scheduling objectives can be further grouped into three broad categories: (i) efficient utilization of resources; (ii) good response to demands; and (iii) close conformance to prescribed deadlines.

Many different performance measures exist for scheduling problems in general, but before these can be defined, some notation needs to be introduced. Given a feasible schedule S which must be an allocation of the jobs to time intervals on the machines such that all restrictions are met, we can compute for each job j:

- $C_j(S)$: The time at which the processing of job j is completed.
- $F_j(S) = C_j(S) - r_j$: The flow-time of job j (also called response time), defined as the amount of time, job j spends in the system.

- $L_j(S) = C_j(S) - d_j$: The lateness measures how much later than the due date the job finishes. If the job finished earlier than d_j, it is assigned a negative lateness.
- $T_j(S) = \max\{L_j(S), 0\}$: The tardiness of job j is its lateness if it fails to meet its due date, or zero otherwise.
- $U_j = 1$ if $C_j(S) > d_j$, $U_j = 0$ otherwise: The unit penalty for job j if it fails to meet its due date.

For every job j, the cost f_j usually takes one of the parameters described above or the product of the weight of the job w_j with one of these parameters. The optimality criterion can be any function of the costs f_j. For a given job system τ, the frequently used optimality criteria are in the form $f_{max} = \max_{j\in\tau} f_j$ and $\sum_{j\in\tau} f_j$. An example of the most common optimality criterion is the makespan $C_{max}(S) = \max_{j\in\tau} C_j(S)$, which is the length of the schedule, or equivalently the completion time of the last job to leave the system. We will denote the optimal makespan, over all possible valid schedules S as follows:

$$C^*_{max} = \min_S\{C_{max}(S)\}$$

Further, we mainly focus on the total weighted completion time criterion $C(S) = \sum_{j\in\tau} w_j\ C_j(S)$. Let $C_j(\sigma)$ denotes to the completion time of the job j in an optimal schedule σ. Therefore, the optimal value of the total weighted completion time of all valid schedules for the job system τ will be as follows:

$$C^*(\tau) = \sum_{j\in\tau} w_j\ C_j(\sigma)$$

For parallel-machine scheduling problems, the usual used objective function is the makespan. From the viewpoint of a user, the time it takes to finish individual independent jobs may be more important; this is especially true in interactive environments. Thus, if many jobs that are released early, are postponed to the end of the schedule, it is unacceptable for the user of the system even if the makespan is optimal. For that reason, other regular objective functions are studied such as the total weighted completion time $C(S)$; and the *total weighted flow time* $\sum_{j\in\tau} w_j\ F_j(S)$.

On the other side, this objective function (makespan) is commonly used to formalize the viewpoint of the owner of the machines. That is, if the makespan is small, the utilization of his machines is high [66, 53]; this captures the situation when the benefits of the owner are proportional to the work done. In the next chapter, we will show that this relation does not always hold especially for online scheduling. Therefore, we introduced two new optimality criteria which are well suited to represent the utilization of the machines.

1.4 Online Paradigms

In this section, the deterministic on-line scheduling models will be described and some of their fundamental properties discussed. In many real-life situations, it is likely that some of the input instances are not available to the algorithm in advance. This likeliness has led to the rapidly emerging field of *on-line* scheduling. Therefore, the main idea behind an online algorithm is that this algorithm, when it makes its decisions, is not aware of the entire information which are necessary to define a problem instance. In this case, the online algorithm must at any time construct a solution to the currently known partial input without knowledge of the future. This lack of knowledge has prompted to introduce the so called online models and provide online algorithms. There are a range of various online models which differ from each other according to the way in which the information becomes available to the algorithm. It is no surprise that the online paradigm may be the most natural and appealing one in the context of scheduling as far as real world applications are concerned.

Too frequently, when attempting to get a solution for an online problem, one is confronted with the fact that nothing is known about the future. Therefore, the online algorithms, which are provided for such kind of problems, cannot generally produce an optimal solution. In such case when the optimal solution is unattainable, it is reasonable to sacrifice optimality and settle for a good feasible solution that can be computed efficiently. Of course, we would like to sacrifice as little optimality as possible, while gaining as much as possible in efficiency. This derived to the idea of considering com-

petitive analysis that allows us to prove bounds using the so-called adversary. This means that an all-powerful malevolent adversary uses the partial schedule generated by the online algorithm to decide what further jobs should be generated. Therefore, this malicious omnipotent adversary specifies the input instance and schedules such instance optimally. BORODIN *et al.* [11] addressed the approach of modelling the uncertainty in the input data by considering various adversarial online scenarios. For further details on online scheduling algorithms, we refer the reader to the book by BORODIN and EL YANIV [11] and the paper of surveys [18]. For the most important results obtained for the various online models, please refer to the survey by SGALL [72]. For online scheduling, the commonly used classification of the on-line problems depends on which part of the problem is given online. The possibility of different situations for these online paradigms can be distinguished as follows:

1.4.1 Jobs arriving one by one

In this paradigm, denoted by *online-list*, the jobs are ordered in some list and presented one by one to the algorithm according to this list. The algorithm learns all of the job characteristics, including its processing time, as soon as it appears. In this online-list model, release dates and precedence constrains are not allowed as with scheduling jobs one by one these restrictions appear to be unnatural. The scheduling algorithm must assign each job to some machine for some length of time, but it is not necessary to specify the actual interval of time, before the next job is revealed. This assignment must be consistent with the restrictions of the given problem. Moreover, the job must be irrevocably assigned to machine. Namely, once the assignment has been made, the scheduling algorithm cannot change it after the next job in the list is revealed. In this paradigm, the time between when jobs are assigned is meaningless, in other words it is not necessary or useful to introduce idle time on any machine. This paradigm corresponds most closely to the standard online model of request sequences. It might be an appropriate model for online problems such as loading balancing, graph coloring and paging. Clearly, none of these problems includes the notion of time and the only

online feature is the lack of the information of future requests. The number of future requests the algorithm learns when it makes its decision is often referred to as the *look-ahead* of the algorithm. Therefore, the *online-list* paradigm refers to algorithms with look-ahead one. This paradigm was widely studied in many papers, for example, in KARGER *et al.* [40], ALBERS [2] and SEIDEN [71]

1.4.2 Jobs arriving over time

In this paradigm, denoted by *online-time*, the jobs become available over time according to their arrival dates. Further, at each instant of time t, the algorithm must decide which job to execute at this time t. In this online-time model, the scheduling problem typically has release dates, and the algorithm is not aware of the existence of a job until its release date. Further, as soon as the job becomes available at its arrival time its characteristics become known. However, in some situations the processing time of the job is only determined one this job has completed. This online-time paradigm is further classified into two paradigms based on what information is revealed about a job once it has arrived.

- *Online Clairvoyant Scheduling*: In this model, once a job is released, the algorithm learns the processing requirement of this job. Therefore, the only online feature in this case is the lack of knowledge of jobs arriving in the future. This model has a number of motivating applications where clairvoyance arises in practice. The most classical one is the web server. A web server serving *static* documents might reasonably be modelled by the clairvoyant model as the size of the requested file is known to the web server.

- *Online Non-clairvoyant Scheduling*: In this model, only the existence of a job is revealed to the algorithm at its release date. Therefore, the processing requirement of a job is still unknown when this job is available. As long as the job is not completed, a non-clairvoyant online algorithm only knows that this job is still being processed. The scheduler learns the processing requirement of a job only when

this job has finished. Therefore, the main online feature here is that the scheduler has no *a-priori* knowledge of how long time requires a job for processing and when future jobs will enter the system. Further, the situation when all the jobs are available at the beginning plays an important role in this paradigm too. If there are other characteristics of a job than its processing time, they are known when the task becomes available. This paradigm is motivated by the situation of a scheduling algorithm which receives the jobs from multiple users and has no way to know how long each job will take to complete. It was first introduced formally and considered by MOTWANI *et al.* [58]. This paradigm closely models the scenarios that describe for example the processor scheduling in a time-sharing multi-tasking operating system, in which scheduling decisions must be taken without knowledge of the time a job needs to complete. For instance, it is better to model the process scheduling component of an operating system by the non-clairvoyant model as the execution times of the various client processes typically will not be known to the operating system.

- *Online Semi-clairvoyant Scheduling*: Often in a real system, while job size may not be known exactly in advance, it is usually possible to get a rough idea of the job size by learning from past data. This naturally leads to a setting that lies between the clairvoyant and the non-clairvoyant model. An attempt at this has been recently made via the study of semi-clairvoyant scheduling by BENDER *et al.* [10] and BECCHETTI *et al.* [8]. Therefore, a semi-clairvoyant online scheduling algorithm only requires *approximate* knowledge of the initial processing time of each job. There are two different kinds of this model. A *strong* semi-clairvoyant algorithm learns a constant approximation of the remaining processing time of a job, and a *weak* semi-clairvoyant algorithm learns only a constant approximation of the original processing time of a job. One of the most practical applications for these models is the dynamic serving in a web server. whereas the document size is only an approximation of the time required by the server to handle a request. In this case, a web server serving *dynamic* documents may only be able to estimate

the size of resulting document as it dynamically constructs the document.

In this online paradigm, the algorithm has more freedom as in the previous one. Here, at any point of time, all currently available jobs are at the disposal of the algorithm. Therefore, each job that is already released and not hindered by any other constraints can be started on any machine or be delayed further. Moreover, if preemptions are allowed the algorithm can decide to preempt or stop any job that is currently being processed.

The situation when the clairvoyant online algorithm learns the arrival date of the next job is known as *nearly online* scheduling [46]. Moreover, if the clairvoyant online algorithm is given all arrival dates of the jobs in advance, then we go back to *off-line* scheduling. Namely, the jobs have to be scheduled with entire knowledge of the problem instance beforehand.

1.4.3 Scheduling with rejection (Interval Scheduling)

In all previous paradigms, each job can start at any point of time provided that it does not violate the problem restrictions, that is, a job may be delayed. Contrary to that, in a paradigm of scheduling with rejection each job has to be processed in a precisely given time interval with fixed start and end points, so it cannot be executed either early or late. If there is no way to process a job in its predetermined interval, then it may be rejected. It is clear that this scenario is complectly different from the previous paradigms. For instance, measuring the length of the schedule is meaningless and not useful in this paradigm as this length is essentially fixed. So in this case, it might be appropriate to measure the weight, or the number, of accepted jobs instead. This paradigm has been studied for example by BARTAL *et al.* [6] and DAS GUPTA *et al.* [26]. Further, GELEMBE *et al.* [21] considered this paradigm for some applications.

1.5 Competitive Analysis

Decision making can be considered in two different phases: making decision with complete information and making decision with partial information. The study of computational complexity of algorithms is useful to distinguish the quality of algorithms based on the computational resources used and the quality of the solution they compute. However, the computational complexity of algorithms may be irrelevant or a secondary issue when dealing with algorithms that operate in a state of uncertainty. Given that an online algorithm has only partial knowledge of the input instance, for most scheduling problems, no online algorithm can produce an optimal solution for all input instances. The common approach for evaluating the worst-case performance of an online algorithm has been first introduced in the seminal paper of SLEATOR and TARAJAN [75]. In this paper, the authors analyzed various paging and list update strategies in an online setting. This approach laid the foundations of the *competitive analysis* technique. However, the actual term " competitive analysis" was coined by KARLIN *et al.* [41]. The competitive analysis technique compares the results produced by an online algorithm to the optimal result, which could have been produced if the complete knowledge about the whole input instance had been available beforehand. The quality of an online algorithm is expressed in its *competitive ratio*, that quantifies by how much, in the worst case, the online solution deviates from the optimal off-line solution. In the literature, the competitive ratio is sometimes also called the *worst case ratio* or the *worst case performance guarantee*. Mathematically speaking, an online algorithm $\mathcal{A}$ is said to be c-competitive if the objective function value of the schedule produced by this online algorithm on any input instance $\mathcal{I}$ is at most c times objective function of the schedule of the optimal off-line algorithm $\mathcal{A}^*$ on the same input. Here, the optimal off-line algorithm has complete knowledge about the whole input sequence in advance. Let $f(\mathcal{A}, \mathcal{I})$ denote the objective value of the schedule produced by algorithm $\mathcal{A}$ on input instance $\mathcal{I}$ where $\mathcal{A}$ could be an online or off-line algorithm and f be an objective value that we are trying to minimize.

Formally, a deterministic online algorithm has a competitive ratio c if

$$f(\mathcal{A}, \mathcal{I}) \leq c\, f(\mathcal{A}^*, \mathcal{I})$$

holds for every possible instance $\mathcal{I}$. The aim in any online scheduling problem is to find an algorithm with a competitive ratio as small as possible. Moreover, a competitive ratio c is said to be *tight* if there exists an specific instance that obtains the stated value. Ideally, this competitive ratio should be a constant independent of any parameter of the input instance such as the number of jobs faced but this is not always possible.

In general, the above definition of the competitive ratio can be extended to allow a fixed constant b which should be independent of the input instance. Often an online algorithm is called $c-$competitive if there exists a fixed constant b such that,

$$f(\mathcal{A}, \mathcal{I}) \leq c\, f(\mathcal{A}^*, \mathcal{I}) + b$$

holds for any input instance $\mathcal{I}$. Some authors even allow b to depend on some problem or instance specific parameters. However, in most scheduling problems, this additive constant b can be ignored. This follows from the fact that scheduling problems are typically scalable; by scaling all the jobs so that the objective value is arbitrarily large, the possible benefit of the additive constant disappears. In this thesis, we will stick to the first definition which is the commonly used.

It is obvious from the above definition of the competitive ratio that there is no restriction on the computational resources of the online algorithm. The only scarce resource in competitive analysis is information. Competitive analysis of online algorithms can be imagined as a game between an online player (an online algorithm) and a malicious omnipotent adversary. The online player uses an online algorithm to process an input which is generated by the adversary. If the adversary knows the strategy of the online player, he can construct a request sequence that maximizes the ratio between the player's cost and the optimum off-line cost. For a great and in-depth treatment on online algorithms and competitive analysis, we refer the reader to [11].

In the non-clairvoyant setting in which schedulers lack knowledge of the characteristics of existing jobs, the performance is measured in a similar way. In particular, an

algorithm is $c-$competitive if the objective function value of the solution produced by $\mathcal{A}$ on any input instance is at most c times that of the optimum off-line algorithm (and hence clairvoyant) on the same input. We are interested to apply such competitiveness technique for our scenario and it is worth to mention some alternative techniques that also use to analyze the online scheduling problems in general. In this research, those alternative techniques are not the focus of attention; however, we should make some remarks on the situation here.

1.6 Alternative techniques for analyzing online algorithms

Randomized algorithms

The first and standard alternative approach to analyze online scheduling problems is the consideration of randomized algorithms. These randomized algorithms make random choices as they produce a schedule. In other words, probabilistic assumptions have to be made on the input distribution. For any sequence $\mathcal{I}$ of jobs, let $E[f(\mathcal{A},\mathcal{I})]$ denote the expected corresponding objective value of a schedule constructed by a randomized algorithm $\mathcal{A}$, and let $f(Opt,\mathcal{I})$ denote the optimal objective value. Then the competitive ratio of $\mathcal{A}$ is defined as the smallest number c such that $E[f(\mathcal{A},\mathcal{I})] \leq c \cdot f(Opt,\mathcal{I})$ for all sequences. A randomized algorithm with a competitive ratio c is called a c-competitive algorithm. In the context of the online algorithms terminology, this corresponds to the so-called *oblivious adversary*, see BORODIN and EL-YANIV [11]. An oblivious adversary has to commit to an input instance without any knowledge of the random events internal to the algorithm. It is clear that the oblivious adversary concept is appropriate for scheduling problems where the scheduling decisions do not affect future input. For some online scheduling problems, applying the randomized algorithms dramatically decreases the value of the competitive factor. For instance, when the randomized algorithms are allowed against an oblivious adversary for the non-clairvoyant scheduling problem $1|r_j,prmp|\sum F_j$, then the competitive factor drops from $\Omega(n^{\frac{1}{3}})$ to $\Theta(\log n)$ as was shown in BECCHETTI and LEONARDI [7].

Resource augmentation

Another alternative means to measure the quality of online solutions has been emanated recently from a paper by KALYANASUNDARAM and PRUHS [37]. This analysis is useful especially in the context of online scheduling problems. Under their paradigm the online algorithm is equipped with extra resources in the form of faster machines or extra machines. So far, most results that obtained by using this approach utilize faster machines. PHILLIS *et al.* [63] generalized this approach calling it *resource augmentation*. Let $\mathcal{A}_v$ denote an online algorithm that is "enhanced" by being run on a machine that is faster by a factor of v than the adversaries machine, where $v \geq 1$. We say that an online algorithm $\mathcal{A}_v$ is an v-speed c-competitive algorithm if $f(\mathcal{A}_v, \mathcal{I}) \leq c \cdot f(Opt_1, \mathcal{I})$ for all input instances $\mathcal{I}$.

Research with resource augmentation has focused on two primary goals. The first objective is to minimize the speed subject to the constraint that the competitive ratio is $O(1)$. Thus an ideal resource augmentation result would be to prove an algorithm is $(1+\epsilon)$-speed $O(1)$-competitive. The second objective focuses on finding an v-speed 1-competitive algorithms for these problems for as small a value of v as possible. The intuition behind these results is that v represents the tradeoff between extra resources and the partial knowledge that the online algorithm faces. Therefore, with v times faster machines, the online algorithm can be able to overcome its lack of knowledge of the input instance and construct a schedule that is at least as good as the one constructed by the optimal off-line algorithm.

Semi-online algorithms

The relative dismal performance of online algorithms is due to a possibly arbitrarily large variance of job parameters. For instance, most greedy algorithms produce bad schedules if they are applied to handle many jobs of same processing time and a few very long jobs. Such inputs may be rare in applications, and we want to avoid them in the analysis by equipping the algorithm with some partial additional information about the jobs in advance. Semi-online is neither off-line nor online, but somehow in between. A semi-online algorithm may be aware in advance of the optimum value,

the length of the longest job, or the jobs may be required to arrive sorted. In recent years, semi-online scheduling problems have received increasing attention from the scheduling community due to their application in practice.

Average-case analysis

It is desirable to analyze the average case behavior of algorithms if the algorithm exhibits its worse case behavior only in some extreme cases but overall performs quite well. There are several examples like *quicksort*, where the worst case time is considerably worse than the average running time. In an average case analysis, we are not only interested in a good average performance, but we would like that this good performance is attained with high probability. The first difficulty with the consideration of the average-case analysis is to determine what the average is, i.e. which probability distribution of the input data is meaningful and still analyzable. Therefore, the average-case analysis is appropriate only if we have a reasonable approximation of what the input distribution should be. This is known for some client server systems. For example, Poisson distribution for job arrivals and independent identical Zipf distributions for job sizes are often used to model the traffic for a web server.

Although all those several alternative techniques mentioned above, the worst-case analysis of online algorithms is of fundamental importance. For many scheduling online problems, worst case competitive analysis gives quite strong bounds.

1.7 History and List Scheduling

Scheduling and sequencing problems have been studied for many decades. Scheduling began to be taken seriously at the beginning of the 20^{th} century when motivated by production planning and manufacturing. Henry Gantt (1861-1919) was one of the first pioneers of this area. However, the first scheduling publications have appeared after many years in the operational research and industrial engineering literature. Some of those publications were in the early 1950's and contained results by JOHNSON [36], SMITH [76], and JACKSON [34]. In the late 1960's and 1970's the area had a sharp

growth, and has retained its momentum ever since. A significant amount of research has been made during 60's on dynamic programming [9, 31, 47, 78] and integer programming formulations [22] of scheduling problems. After the famous complexity theory presented by RICHARD KARP [42, 43], an increasing amount of attention paid during 1970's to consider the complexity hierarchy of scheduling problems. In the 80's, stochastic scheduling problems (vs. deterministic scheduling problems) absorbed an increasing amount of attention in both academia and industry [16, 56, 67]. Further, during this time, the area had another stimulus because of the use of personal computers permeating manufacturing facilities. This advent of the computers and their widespread use had considerable impact both on scheduling problems and solution strategies. From that time on, many scheduling systems were developed for the generation of usable schedules in practice. Moreover, a number of new problems and variations has been motivated by applications areas in computer science such as parallel computing, databases, compilers, and time sharing.

Over these years of fertile research in the scheduling field, some problems in this area were formally classified and defined. As we are interested in the on-line scheduling problems, we provide a brief overview on the scheduling models with the online setting. Starting from the middle of the 1960's, the researchers appreciated the fact that parts or all of the relevant information might not be available to the algorithm. From this time onwards, much attention in scheduling theory has been devoted to study online (dynamic) problems. In 1966 GRAHAM [23] provided the first proof of competitiveness of an online algorithm for a scheduling problem. His algorithm is a simple deterministic greedy algorithm, now commonly called *List Scheduling Algorithm*, and is used extensively both in theory and practice. The investigated model was the basic one in which m identical machines and a set of sequential jobs characterized by their processing times are considered while the objective was to minimize the makespan, *i.e.* $P||C_{max}$. Preemption was not allowed. In the literature, the name list scheduling is used to refer to different algorithms that have similarities. The list algorithm first orders the jobs in an arbitrary sequence, jobs that come earlier in the list have higher priority. Then it assigns immediately the earliest available job in this sequence to the vacant machine. Next, we give the precise description of such algorithms:

Algorithm: GRAHAM's list scheduling algorithm.

Input: Instance for $P_m|r_j|\gamma$, a job system τ.
Output: A feasible schedule.

while *there is still an unscheduled job in* τ **do**
  pick any unscheduled and released job j (if any);
  if *such a job exists and a machine is idle* **then**
    start j immediately on an idle machine
  else
    wait until any machine becomes idle or
    the new job is released;

By their non-idling property, GRAHAM's list scheduling algorithms are well appropriate when machine utilization is an important consideration as was shown by MUNIER *et al.* [60]. The list algorithm was already designed both for the model with and without precedence constraints on the jobs. In addition, the multi-release date case can be handled also by a simple modification as has been done later by HALL and SHMOYS [30]. For the model of scheduling jobs one by one, once a job is presented, the list algorithm assigns it to a machine that has currently the smallest load. GRAHAM showed that the job arrival order (online scenario) can change the resulting makespan by a factor of at most $2 - \frac{1}{m}$ and that this bound is tight. In the same seminal paper, the case when the number of machines is not fixed has been covered and the best possible bound was giving for this algorithm. The bound $2 - \frac{1}{m}$ decreases to the bound $\frac{4}{3} - \frac{1}{3m}$ if the algorithm requires the jobs to arrive in a list sorted according to decreasing processing times, as was shown in a follow-up paper of GRAHAM [24].

Two other early famous papers that analyze online scheduling algorithms can be found in [68] by SAHNI and CHO and in [15] by DAVIS and JAFFE . The first one presents an optimal algorithm for minimizing the makespan of a preemptive schedule on identical machines where jobs is releasing over time and their processing times are known at the arrival. In fact, this algorithm requires to know the release date of a job in advance. This additional restriction on the online problems was later removed by HONG and LEUNG [33]. The second paper is, to the best of our knowledge, the first

one that mentions explicitly a lower bound on the performance ratio of any online algorithm for some scheduling problem. It provided an efficient algorithm with a behavior which is $\Theta(\sqrt{m})$ times worse than optimal in the worst case. This bound obtained for the non-preemptive assignment of independent jobs that arriving over time with unknown processing times on uniformly related machines. Also, the authors conjectured that it may be useful to allow restarts in this case, indeed a validity of this suggestion has been proven later by SHMOYS *et al.* [73].

During 1990's several online algorithms were designed and new results obtained to handle many variants of online scheduling. The makespan was commonly used as the objective function. However, the appearing of several new practical applications is drawing to more attention for considering other objective functions, which are more suitable in this case than the makespan.

1.8 Practical Examples of online models

It is important to have in mind a set of practical problems that occur in the real-life world. There is a tremendous variety of problems that pose online modelling and algorithmic issues. Below there are some online resource allocation problems that help to highlight some of various online models. The problems arise in planning, production systems, and computer control, respectively.

Example 1.8.1

Consider a factory that produces cardboard boxes for various types of productions such as dog food, charcoal, $\cdots$, and so on. A factory receives over time a continuous demand from the customers. The cardboard boxes are produced by a group of machines that have been bought over the years, that is, each machine has its own processing speed. Every machine can process all stages to make a complete box from a single cardboard sheet. This includes printing it in multi-colors, cutting, folding, and gluing. Each production order (request) indicates a given quantity of a specific box. Moreover, this order has to be produced and delivered by a promised date. Now, one

has to make his decision how the orders can be assigned to the machines so as to minimize the number of late delivery orders. A similar example can be found in [66].

In this example, the orders that have to be produced are the jobs. The machines in the factory clearly represent the machine environment. As each job can be processed on any one of the machines that have different speeds, the machine environment corresponds to a group of uniform parallel machines. Here, the minimization of the number of tardy jobs is the scheduling objective. Obviously, this scenario belongs to the online paradigm in which jobs arrive over time and all characteristics of each job are available at it's arrival time.

Example 1.8.2

Another application area of scheduling is network systems. A number of computers are connected to compose a computer network that is used for transmitting messages between the users of the computers. Every computer has several users that want to transmit messages through this network. A message route is determined along which the transmission should occur before actually transmitting this message. In this computer network, there are many different routes to sent a message - routing is one important research topic in this area. Information (messages) may be sent on cables or wireless. From a functional point of view, cables or wireless links can equally well be regarded as resources. This is a good example of a resource which is not a physical machine in the usual sense of the word. The communication links are machines which provide alternative ways of executing an information transmission task. An example of such computer network can be seen in Figure. 1.2. The computers and the connections are represented by the vertices and the edges, respectively. A possible transmission route between computers A and B is indicated by a dashed line. The bandwidth of each connection in the route has to be fixed, that is, the number of bits per second that the connections can transmit is limited. The requests for sending a message arrive dynamically over time at the computers. In addition, each message has a size in bits, which is not available at the request time. No computer in the route can forward a message until this message has been received completely. Now, we want to look for a protocol that, given the transmission route, minimizes the average waiting time by

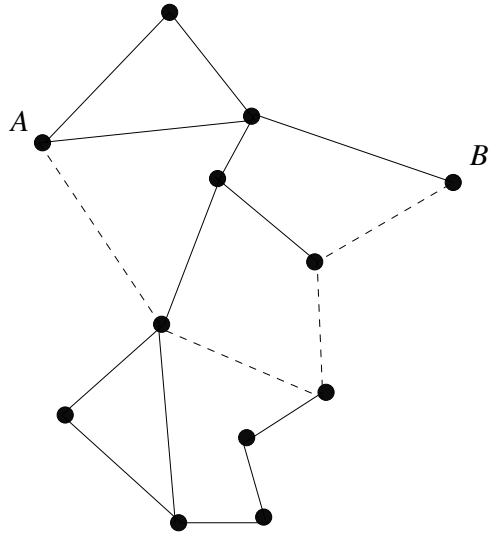

Figure 1.2: *Possible transmission route for messages between computers A and B in a network*

regulating the communication traffic.

The messages in this example can be viewed as jobs. The connection between any two computers can be regarded as a machine. All messages are moving through a route which is known in advance. Therefore, we learn in advance the order in which the machines are visited. Here, the objective is to find a schedule that minimizes the average waiting time. It is obvious that this is equivalent to minimize the total waiting time to serve all messages. Furthermore, the difference between the total waiting time and the total completion time is just a constant. Hence, this problem can be modelled as a open shop, where the minimization of the total completion time is the scheduling objective. clearly, the online feature in this example is that the messages arrive over time. Further, the time a message requires for transmission is unknown even during the transmission process. Consequently,a non-clairvoyant algorithm is requiring to handle this problem.

Example 1.8.3

A ferry-ship transports vehicles from one shore to the other. A queue of the vehicles is

waiting in front of this ferry for transportation. It is the task of the ferry man, who is standing in front of this queue, to assign a location on the ferry-ship for each one of the vehicles. His final goal is minimizing the free area of the boat. The view of the ferry man is partially blocked such that he can only see the first four vehicles in the queue at a time. Once a position on the boat has been assigned for the first one of those four vehicles, the vehicles move up and the next vehicle in the queue can be seen. Due to his years of experience, the ferry man knows exactly the space required by a vehicle once he sees it. For obvious reasons he can not change the assignment of a vehicle once the decision has been made. The assignment of the vehicles to the deck of the ferry-ship has been highly structured to avoid chaos. Parallel lanes with the same width and length have been constructed on the deck of the ferry-ship. Each lane has enough width that can contain any of the vehicles. Also, vehicles can only be assigned to the prescribed lanes and must be located one after another in each of the lanes.

To formulate the problem in context of scheduling we need to look for the machine environment, job characteristics, and an optimality criterion. The jobs in this example are the vehicles, where the length of the vehicle represents the processing requirement of a job. The machines here are the lanes into which the deck is partitioned. Each job can be assigned to any one of the machines (lanes). Clearly, the amount of area that is used by the vehicle is independent of the lane. This indicates that the machine environment corresponds to a group of identical parallel machines. The objective is the minimization of the free area. This is clearly the same as minimizing the total length of the vehicles left ashore. Further, the objective will be the minimization of the weighted number of tardy jobs if a weight and a due date are defined to be equal to the length of the vehicle it represents and the length of the lanes, respectively. The online feature that is used is obviously the paradigm where jobs arrive sequentially in a list (one by one). Further, the first job in the list has to be assigned immediately and uniquely to a machine. Therefore, we need only algorithms with a look-ahead of four to handle this problem.

Chapter 2

Criteria for system utilization

A central question that must be considered when analyzing an algorithm for parallel-machine systems is: what is the objective function being used to determine how well such algorithm is performing. Clearly, the answer depends on your goal. From a system viewpoint, it is needed to streamline the use of the resources and increase overall system utilization. That is, system managers would look for higher system utilization. In this research, we are interested to maximize system efficiency during a specific time interval (from system start until a target time). As we mentioned, the first step in solving a scheduling problem is thus to define an optimality criterion that is appropriate to achieve the scheduling aim. However, choosing a scheduling objective function is, itself, a challenging problem. This chapter is devoted to consider the problem of determining the performance measures that are well suited to evaluate the utilization of the machines within a target time interval.

It is well known that the makespan criterion is the common way to evaluate the utilization of the machines. However, in the next section we will show that the makespan criterion is not always closely related to machine utilization. Consequently, in Section 2.2, we will introduce new alternative optimality criteria for the purpose of maximizing the machine utilization. Further, we will show that our new criteria are well suited to represent quantitatively the machine utilization. To simplify our analysis, in Section 2.3, we will introduce the definition of certain job systems which passes some special properties. These specific job systems are named basic job systems. In sec-

tion 2.4, we will show that it is sufficient to consider only those basic job systems and their basic nondelay schedules for the purpose of a worst case analysis.

2.1 Is makespan suitable for utilization?

The issue of what performance measure to use in assessing the quality of a schedule is far from trivial. For parallel machines, the makespan criterion is usually considered to represent machines utilization, see, for instance, PINEDO's book [66] and [53]. Further, RINNOOY KAN [38] developed some equivalence relationships that exist among scheduling objectives. One of the results in this paper shows that minimizing makespan is equivalent to maximizing utilization of resources. Therefore, it is taken commonly for granted that a schedule with a smaller makespan has a higher machine utilization. However, this relation does not always hold for online scheduling models with independent jobs. For instance, if a system is running empty, that is, all job submissions have been completed and there is no further open request, then the utilization of the current schedule is optimal although this may not be true for the makespan. On the other hand, if there are still running jobs or open requests, a schedule with a higher makespan may have a utilization in the time interval up to the current time better than that of a schedule with the optimal makespan.

To explain that, consider the following instance. A job system contains three machines and five jobs with the following processing times:

job	1	2	3	4	5
p_j	2	1	10	6	1

All jobs are released at time zero and the target (specific) time interval is $(0, 5]$. At any time instant in this interval, we assume that the processing times of all jobs are known although some of them are not yet completed. Figure 2.1 compares between two different nondelay schedules of this simple instance. This figure shows that a nondelay schedule with a higher makespan may have a better machine utilization in the target time interval a nondelay schedule with the optimal makespan. For example,

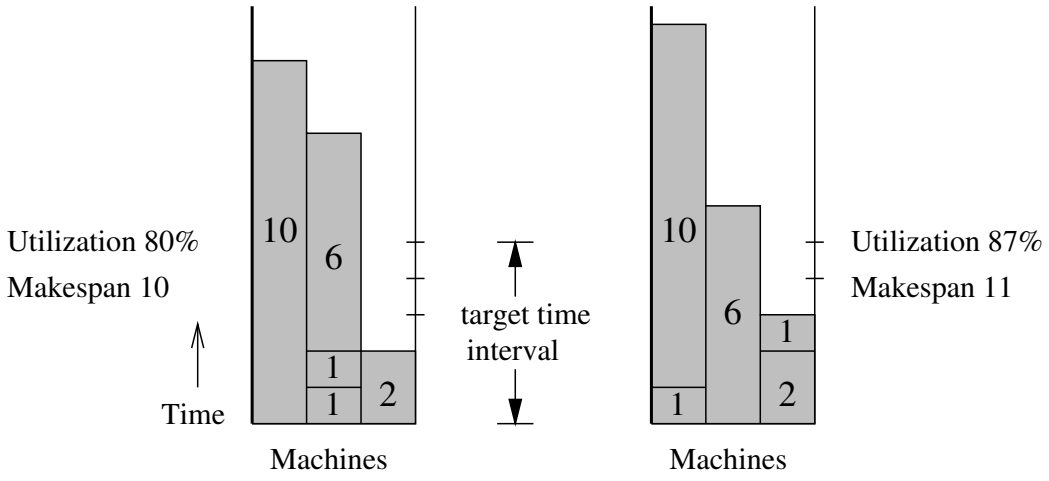

Figure 2.1: *Comparison of Makespan and Utilization for 2 Schedules*

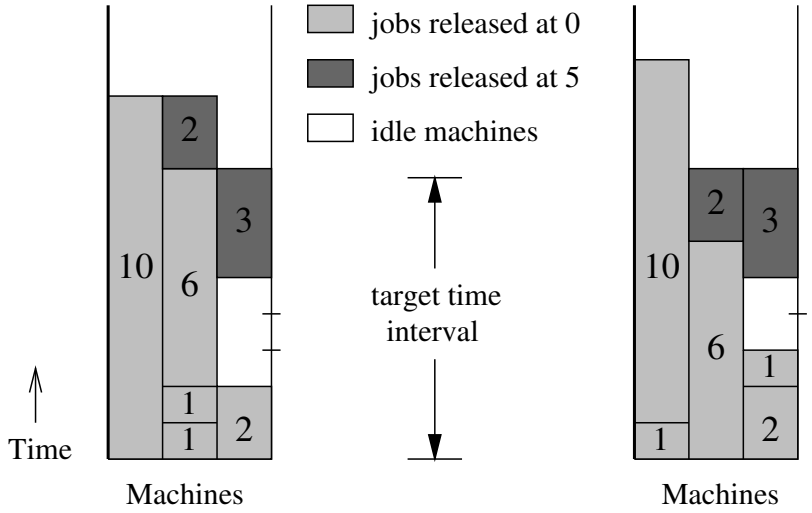

Figure 2.2: *Effect on Future Job Submissions on 2 Schedules*

at the end of the current target time interval (after five units of time) the schedule with makespan equal to 10 has $\frac{12}{15} = 80\%$ utilization of the machines but the schedule with makespan equal to 11 has $\frac{13}{15} \approx 87\%$ utilization of the machines.

To show that machine utilization might be influenced by future job submissions (even if an oracle provides us with the processing times of all currently running jobs), we consider the following instance: here, we add two new jobs to the same previous job system. Those new jobs are released at time 5 with processing times $p_6 = 2$ and $p_7 = 3$. In this case, we assume that the interval from system start to 8 units of time is the target time interval. Figure 2.2 presents two different nondelay schedules for this job system. The first one has minimal makespan 10 and $\frac{21}{24} \approx 88\%$ machine utilization at the end of the target time interval. The second schedule has a makespan of 11 while the machine utilization at the end of the target time interval is $\frac{22}{24} \approx 92\%$. This figure illustrates that, although all new (future) jobs are scheduled in an optimal fashion in both schedules the schedule with the optimal makespan may result in a smaller machine utilization.

SHMOYS *et al.* [73] have investigated the makespan minimization for the problem of non-clairvoyantly scheduling jobs that arrive over time on identical parallel machines. The authors have already shown that GRAHAM's list scheduling algorithm has a competitive factor of $2 - \frac{1}{m}$. That is, $\frac{C_{max}(S)}{C^*_{max}} \leq 2 - \frac{1}{m}$ holds with S being an arbitrary nondelay schedule and C^*_{max} denoting the optimal makespan. This is the best possible factor in the deterministic case, *i.e.* this bound is tight. According to this result, the owner must assume that the selection process may be responsible for up to almost 50% idleness. However, in this thesis, we show that the selection process may account for 33% idleness at most. In addition to the previous simple example, this also shows that the makespan criterion may not always be an appropriate criterion to represent utilization in all scheduling problems.

From the above, one concludes that the makespan criterion is not always appropriate to measure the machines utilization for this kind of online problems. This leads us, in the next section, to introduce another new criteria that enables us to represent well the utilization of the machines. Hence, we can directly consider system utilization from system start to a given end time, *i.e.*, during the target time interval. In this

thesis, we are interested to determine the ratio between the utilization of an optimal and a worst case nondelay schedule in such a specific time interval. To determine the best schedule for this time frame, we assume that the processing times of all jobs submitted in this time frame are available. As jobs that are submitted after the end time cannot influence the value of our criterion in any schedule we can ignore them in our analysis. To the best of our knowledge, no theoretical analysis has been provided for this utilization criterion yet.

2.2 New criteria for machine utilization

The ultimate goal of any scheduling problem is to obtain the best system performance possible. As already mentioned before, we are interested in the machine utilization of a schedule. It is an established fact in the parallel computing literature that the makespan influences the machine utilization and it is commonly considered to be the best criterion to represent machine utilization. In the previous section, we however have shown that this fact is not always true by showing that the makespan criterion is not well suited to describe machines utilization for online scheduling problems (even maybe for multi-machines scheduling problems in general). Therefore, makespan is not always closely related to the utilization of the machines. This stimulate us to introduce formally two new alternative performance measures for the purpose of evaluating the utilization of the machines. Further, in this section, we will show that our new criteria are better than well-known and usually used makespan to evaluate machine utilization for the online scheduling models. To this end, we first denote the amount of machine resources used by a given job system τ within a certain time interval $[t_1, t_2)$ of any schedule S by

$$U_{[t_1,t_2)}(S,\tau) = \sum_{j\in\tau} \max\Big\{0,\ \min\Big\{t_2, C_j(S)\Big\} - \max\Big\{t_1, C_j(S) - p_j\Big\}\Big\}.$$

Note that this definition can also be applied in a case where the job system τ is only a subset of all jobs in schedule S. In this case, utilization is restricted to the jobs from this subset within the given interval $[t_1, t_2)$.

As already mentioned in section (1.1), we concern with system utilization only on the interval of time in which the machines are bottleneck. Paying much attention to consider the utilization during an active (bottleneck) interval makes a lot more sense since we are maximizing utilization where it matters, instead of indiscriminately across the whole schedule, see for instance KEMPF *et al.* [45]. This encouraged us to consider the system utilization from system start to a certain target time t. Consequently, we can formally introduce our first new scheduling objective for a job system τ and its schedule S as follows:

Maximization of Utilization for Target Time t: $U_{[0,t)}(S,\tau)$

We denote the optimal value of the utilization criterion for a job system τ by $U^*_{[0,t)}(\tau)$.

Clearly, the limited applicability of the makespan criterion for system utilization is partly due to the fact that usually only a single machine determines the makespan of a schedule. In order to overcome this problem, it may be appropriate to consider a criterion that combines the makespan of all machines. Intuitively, the combination should give priority to schedules where the makespan of all machines is similar. In a schedule without intermediate idle times on any machine, this is achieved by minimizing the l_2-norm of the makespans, that is, $\left(\sum_{i=1}^{m}(C^i_{max})^2\right)^{1/2}$, where C^i_{max} is the completion time of the last job on machine i.

To this end, we can simply use the well known *total weighted completion time* criterion $C(S) = \sum_{j\in\tau} w_j C_j(S)$ which takes into account the completion times of all jobs. In fact, this criterion prioritizes jobs with a large ratio of weight w_j to processing time p_j as has been shown in [76]. In order to overcome this problem, we define the weight of each job to be its processing time. A similar approach of this weight selection has been suggested by SCHWIEGELSHOHN and YAHYAPOUR [70] for the preemptive scheduling of parallel jobs. Further, KAWAGUCHI and KYAN [44] applied this approach for their proofs. This weight selection guarantees that all jobs have the same Smith ratio [76] and therefore the same priority. Moreover, it favors schedules with a balanced load while, contrary to the makespan criterion, a single machine occupied by a long running job has limited influence on the schedule quality if many machines are available.

As there are already many results for total weighted completion time scheduling, we also consider this criterion as an alternative for the utilization and the makespan criteria and name it **equal priority completion time** criterion. However, as there may be intermediate idle times in the schedule due to the release dates of the jobs, the equal priority completion time criterion is not the same as the l_2-norm of the makespans in the online scenarios. Next, we can formally again introduce our second scheduling objective function for a job system τ and its schedule S as:

Minimization of Equal Priority Completion Time: $C_{equ}(S) = \sum_{j \in \tau} p_j \, C_j(S)$

Note that the equal priority completion time $C_{equ}(S)$ of a schedule S is derived from the sum of the weighted completion time by demanding that $w_j = p_j$ holds for all jobs. We denote the minimal value of the equal priority completion time over all feasible nondelay schedules for a job system τ by $C^*_{equ}(\tau)$.

In this research, we analyze the worst case of the ratios $\frac{U^*_{[0,t)}(\tau)}{U_{[0,t)}(S,\tau)}$ and $\frac{C_{equ}(S)}{C^*_{equ}(\tau)}$ where S is a nondelay, nonpreemptive schedule. In Chapter 3, we will consider the determination of competitive factor of the utilization criterion and derive the upper bound of such factor, while Chapter 4 is devoted to investigate the competitive factor of the equal priority completion time criterion. Then, we will show a close relationship between the utilization and the equal priority completion time criteria.

Although using the equal priority completion time criterion to evaluate machine utilization has not been addressed so far, it can be directly concluded from the results of KAWAGUCHI and KYAN [44] that

$$\frac{C_{equ}(S)}{C^*_{equ}(\tau)} \leq \frac{\sqrt{2}+1}{2}$$

holds for all nondelay schedules if all jobs are released at time 0 (offline problem). Further, Kawaguchi and Kyan have shown that this factor is tight.

After the formal definitions of the new optimality criteria, we can quantitatively compare them with the makespan criterion by executing two different job systems τ_1 and τ_2 on an m identical parallel machines.

• Job system τ_1

The first job system τ_1 consisting of $2m$ independent jobs with the following processing times:

$$p_j = \begin{cases} m-1, & \text{for each } j = 1, \cdots, m\,; \\ 1, & \text{for each } j = m+1, \cdots, 2m\,. \end{cases}$$

In the nondelay schedule S_1, we start $m-1$ long jobs at time 0 and execute all short jobs on a single machine in the interval $[0, m)$. This requires a single long job to start at time $m-1$ and to run until time $2m-2$. Note that schedule S_1 represents the worst case of makespan for job system τ_1. In the optimal schedule, all long jobs start at time 0 and all short jobs are executed in parallel in the interval $[m-1, m)$, see Figure 2.3. Observe that in both schedules S_1 and the optimal schedule σ, all jobs are started in the target time interval [0,m). Now, we compute the competitive ratio of each criterion as follows:

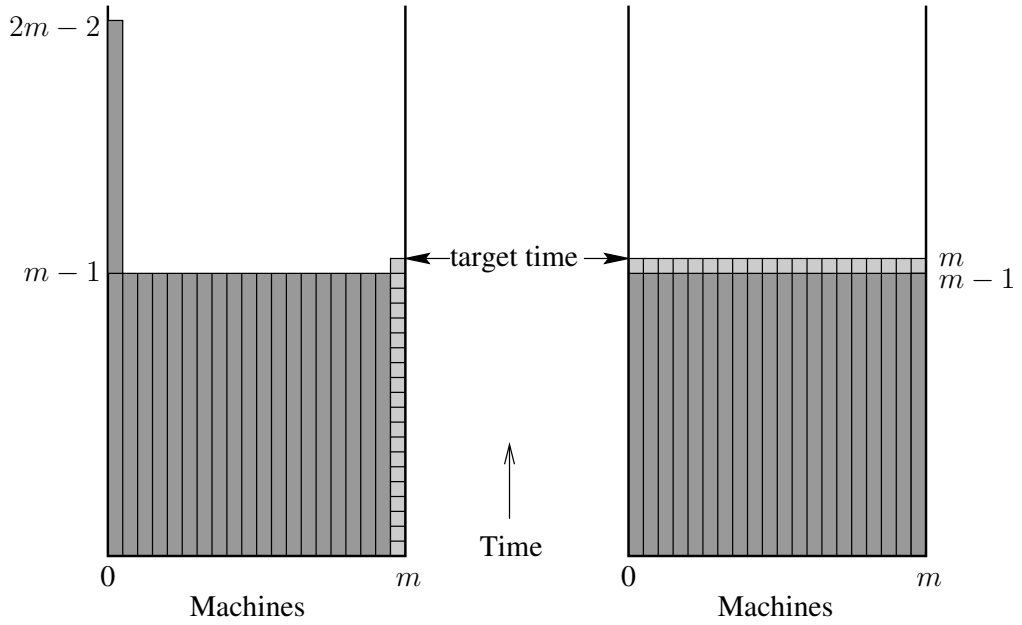

Figure 2.3: *Nondelay schedule S_1 (left) and the optimal schedule (right) for τ_1*

For the makespan criterion, we have

$$C_{max}(S_1) = 2m-2, \quad C^*_{max}(\tau_1) = m \quad \Rightarrow \quad \frac{C_{max}(S_1)}{C^*_{max}(\tau_1)} = 2 - \frac{2}{m} \tag{2.1}$$

For the utilization criterion, we get

$$\begin{aligned} U_{[0,m)}(S_1,\tau_1) &= \sum_{j\in\tau_1} \max\Big\{0,\ \min\Big\{m, C_j(S_1)\Big\} - C_j(S_1) + p_j\Big\} \\ &= (m-1)\cdot(m-1) + m + 1 = m^2 - m + 2, \\ U^*_{[0,m)}(\tau_1) &= m^2. \end{aligned}$$

Terefore, we have

$$\frac{U^*_{[0,m)}(\tau_1)}{U_{[0,m)}(S_1,\tau_1)} = \frac{m^2}{m^2-m+2} = 1 + \frac{m-2}{m^2-m+2} \tag{2.2}$$

The value of the equal priority completion time criterion of schedule S_1 can be obtained as follows:

$$\begin{aligned} C_{equ}(S_1) &= \sum_{j\in\tau_1} p_j\, C_j(S_1) \\ &= (m-1) \sum_{j\in\{\tau_1 | p_j = m-1\}} C_j(S_1) + \sum_{j\in\{\tau_1|p_j=1\}} C_j(S_1) \\ &= (m-1)\Big[(m-1)^2 + (2m-2)\Big] + \frac{m(m+1)}{2} \\ &= (m^3 - 3m^2 + 3m - 1) + 2(m^2 - 2m + 1) + \frac{m(m+1)}{2} \\ &= m^3 - m^2 - m + 1 + \frac{m(m+1)}{2} \\ &= m^3 - m^2 + m + \frac{m^2 - 3m + 2}{2} \end{aligned}$$

Note that the completion times of the short jobs in schedule S_1 are $1, 2, \cdots, m$. The optimum value of such criterion for job system τ_1 is

$$\begin{aligned} C^*_{equ}(\tau_1) &= \sum_{j\in\tau_1} p_j\, C^*_j \\ &= (m-1) \sum_{j\in\{\tau_1|p_j=m-1\}} C^*_j + \sum_{j\in\{\tau_1|p_j=1\}} C^*_j \\ &= (m-1)\cdot m\cdot(m-1) + m\cdot m = m^3 - m^2 + m. \end{aligned}$$

Therefore the worst case ratio of the equal priority completion time is

$$\frac{C_{equ}(S_1)}{C^*_{equ}(\tau_1)} = 1 + \frac{m^2 - 3m + 2}{2(m^3 - m^2 + m)} \tag{2.3}$$

• Job system τ_2

The second job system τ_2 consists of $\frac{m}{2}$ long jobs and $\frac{m^2}{2}$ short jobs with the following processing times:

$$p_j = \begin{cases} m, & \text{for each } j = 1, \cdots, \frac{m}{2}\ ; \\ 1, & \text{for each } j = \frac{m}{2}+1, \cdots, \frac{m}{2}(m+1)\ . \end{cases}$$

In the nondelay schedule S_2 we execute all short jobs in the interval $[0, \frac{m}{2})$ using all machines and start all long jobs at time $\frac{m}{2}$ while in the optimal schedule, all long jobs start at time 0 and all short jobs are processed in the interval $[0, m)$ using only $\frac{m}{2}$ machines. Note that in all schedules, all jobs are started in the target time interval $[0, m)$, see Figure 2.4. In this case, we have

$$C_{max}(S_2) = \frac{3m}{2}, \quad C^*_{max}(\tau_2) = m \quad \Rightarrow \quad \frac{C_{max}(S_2)}{C^*_{max}(\tau_2)} = 1.5 \tag{2.1′}$$

For the utilization criterion we have,

$$\begin{aligned} U_{[0,m)}(S_2, \tau_2) &= m \cdot \frac{m}{2} + \frac{m}{2} \cdot \left(m - \frac{m}{2}\right) = \frac{3m^2}{4}, \\ U^*_{[0,m)}(\tau_2) &= m^2. \end{aligned}$$

Therefore, we get

$$\frac{U^*_{[0,m)}(\tau_2)}{U_{[0,m)}(S_2, \tau_2)} = \frac{4}{3} = 1 + \frac{1}{3} \tag{2.2′}$$

Next, we can obtain the worst case and the optimum value of equal priority completion time criterion for job system τ_2 as follows:

$$\begin{aligned} C_{equ}(S_2) &= \sum_{j \in \tau_2} p_j\, C_j(S_2) \\ &= \sum_{j \in \{\tau_2 | p_j = 1\}} C_j(S_2) + m \sum_{j \in \{\tau_2 | p_j = m\}} C_j(S_2) \\ &= m \cdot \frac{\frac{m}{2}(\frac{m}{2}+1)}{2} + m \cdot \frac{m}{2} \cdot \frac{3m}{2} \\ &= \frac{m^2}{8} \cdot (7m + 2) \end{aligned}$$

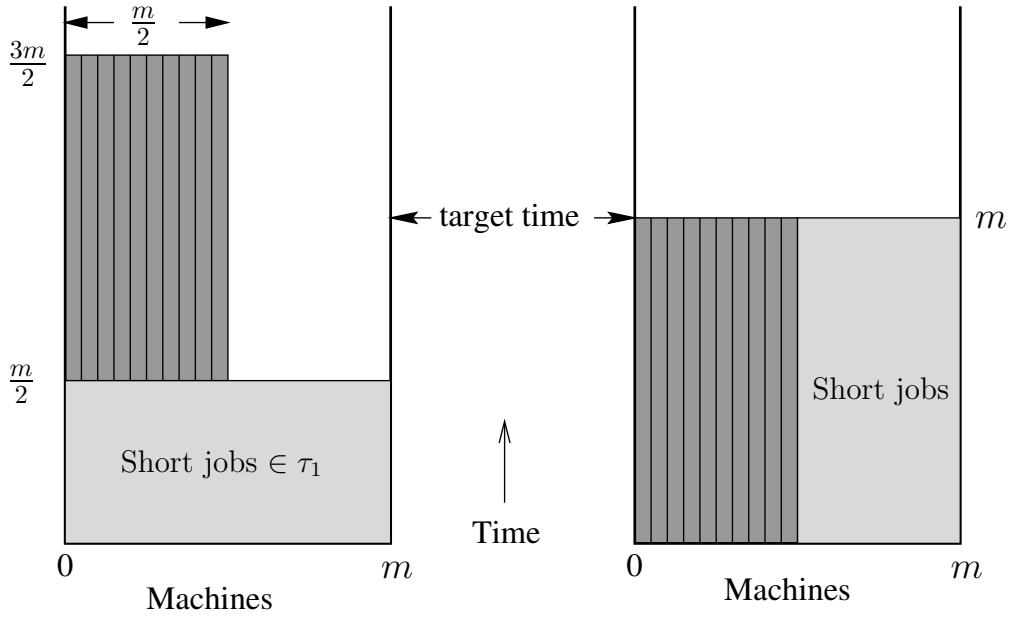

Figure 2.4: *Nondelay schedule S_2 (left) and the optimal schedule (right) for τ_2*

$$\begin{aligned} C^*_{equ}(\tau_2) &= \sum_{j\in\tau_2} p_j\, C^*_j \\ &= \sum_{j\in\{\tau_2|p_j=1\}} C^*_j + m \sum_{j\in\{\tau_2|p_j=m\}} C^*_j \\ &= \frac{m}{2}\cdot\frac{m(m+1)}{2} + m\cdot\frac{m}{2}\cdot m = \frac{m^2}{4}\cdot(3m+1) \end{aligned}$$

Finally, we get the worst case ratio of the equal priority completion time criterion for job system τ_2

$$\frac{C_{equ}(S_2)}{C^*_{equ}(\tau_2)} = \frac{7m+2}{6m+2} = 1 + \frac{m}{6m+2} \qquad (2.3')$$

Table 2.1 summarizes the results from Equations (2.1-2.3) and Equations (2.1′-2.3′) and provides a comparison between the worst case ratios of the makespan, the utilization, and the equal priority completion time criteria for both job systems τ_1, τ_2. The previous two examples also demonstrate that better machine utilization does not necessarily correspond to a smaller makespan while utilization and equal priority com-

	$\frac{U^*_{[0,m)}(\tau_i)}{U_{[0,m)}(S_i,\tau_i)}$	$\frac{C_{equ}(S_i)}{C^*_{equ}(\tau_i)}$	$\frac{C_{max}(S_i)}{C^*_{max}(\tau_i)}$
$i=1$	$\frac{m^2}{m^2-m+2} < 1+\frac{1}{m}$	$1+\frac{m^2-3m+2}{2(m^3-m^2+m)} < 1+\frac{1}{2m}$	$2-\frac{2}{m}$
$i=2$	$1+\frac{1}{3}$	$1+\frac{m}{6m+2} < 1+\frac{1}{6}$	1.5

Table 2.1: *Comparison between Utilization, Equal Priority Completion Time, and Makespan for Job Systems τ_1 and τ_2.*

pletion time criteria exhibit the same trend.

2.3 Basic Job Systems

In this section, we define some notations and terminologies which will be used in the rest of this thesis. Further, we will introduce the definition of a specific job system that simplifies our analysis. Let us start with the following definitions.

Definition 2.1 *An interval $[t_a, t_b)$ of any schedule S is called fully utilized or simply full interval if all machines are busy executing jobs at any time instant during this interval.*

Definition 2.2 *A full interval $[t_a, t_b)$ of the schedule S is max-full interval if it is not a true subset of another full interval in this schedule.*

Therefore, each max-full interval has maximum size. That is, there is at least one idle machine at the beginning and at the end of each max-full interval. Further, it is obvious from Definition 2.2 that there is a unique partitioning of every schedule into those max-full intervals. Note that at least one job starts at the beginning of any max-full interval. Moreover, in any max-full interval $[t_a, t_b)$ of any *nondelay* schedule we have $r_j \geq t_a$ for every job $j \in \tau$ that starts in such interval $[t_a, t_b)$. Otherwise, this violates the nondelay

property as job j could start before this interval. Further, we will need the next two notations in our consideration.

- For a given schedule S, we define $[t_a(S), t_b(S))$ to be the *last* max-full interval of such schedule S. If the schedule S has no full interval then we set $t_a(S) = t_b(S) = 0$.
- Let $\tau([t_a, t_b]) \subseteq \tau$ such that $t_a \leq C_j(S) - p_j \leq t_b$ holds for each job $j \in \tau([t_a, t_b])$. That is, it is the set of jobs that start in the interval $[t_a, t_b]$ of schedule S. Note that both t_a and t_b are included.

Next, we provide the definition and the concept of a *basic* job system. In the next section, we will show that the worst-case ratios of utilization and equal priority completion time can be produced by such kind of job systems. For a basic job system, there must be a nondelay schedule S such that the processing time of each job starting in a max-full interval of S is very small and those jobs are released at the beginning of this interval. Further, if a job is released before the beginning of the max-full interval of S and always completes after this time in any schedule then the start time of this job in schedule S is greater than its release date plus the length of the max-full interval. On other words, all jobs that are released before the beginning of max-full interval, but cannot finish before this interval in any case, must wait at least the length of the max-full interval before starting. We will explain the intuitive reason of these properties in the next section. Mathematically, the properties of basic job systems can be described by the following definition.

Definition 2.3 *A job system τ is called a* **basic** *job system if there is a nondelay schedule S for τ such that the following conditions are valid for any max-full interval $[t_a, t_b)$ of S and a fixed $\epsilon > 0$:*

1. *$p_j \leq \epsilon$ holds for all $j \in \tau([t_a, t_b))$.*
2. *$r_j = t_a$ holds for all jobs $j \in \tau([t_a, t_b])$.*
3. *$C_j(S) - p_j > r_j + (t_b - t_a)$ holds for all jobs $j \in \tau$ with $r_j < t_a < r_j + p_j$.*

Schedule S is called a **basic** *nondelay schedule.*

Observe that in the first property time t_b is excluded from the interval.

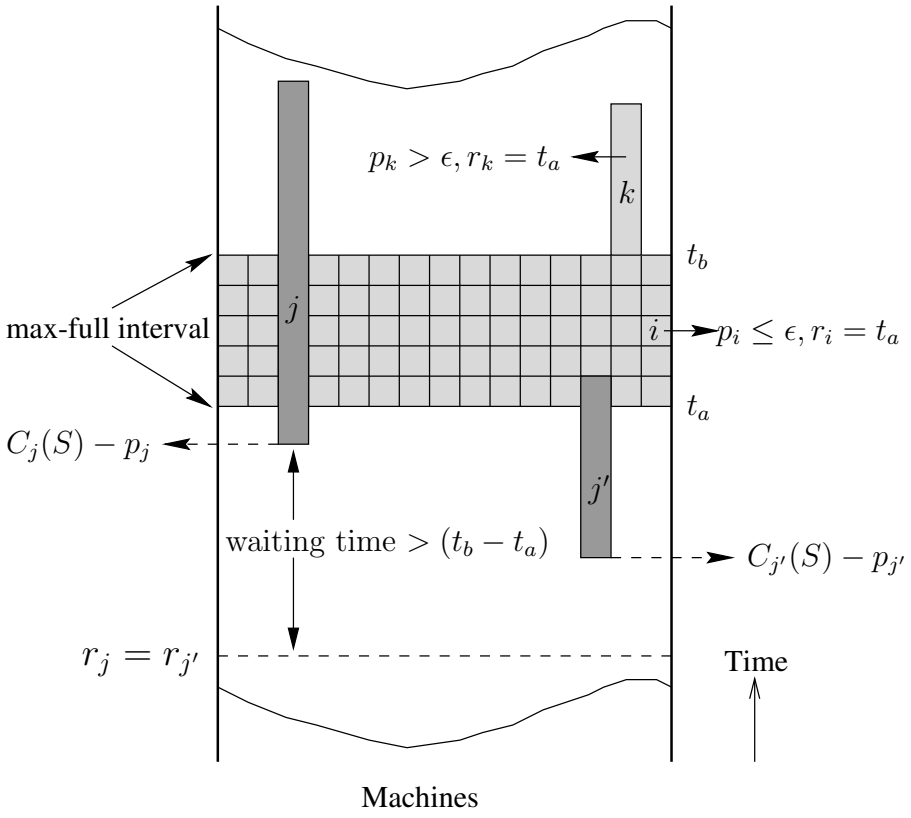

Figure 2.5: *Basic job system and its basic nondelay schedule S*

Figure 2.5 shows the concept of a basic job system. According to the first property of Definition 2.3, all jobs that start in the max-full interval $[t_a, t_b)$ are very short, such as job i. Further, all those short jobs have the same release date t_a as well as each job that start at time t_b (such as job k) and this represents the second property. If a job j is released before t_a and cannot complete at t_a or earlier even it starts at its release date r_j, then it has to start after time $r_j + (t_b - t_a)$ in the basic schedule S. On the other hand, job j' with $t_a < r_{j'} + p_{j'}$ that is also released before time t_a may start at any time before

t_a in the basic schedule S. From the third property of Definition 2.3, we have

$$\begin{aligned} C_j(S) &> r_j + p_j + t_b - t_a \\ &> t_a + t_b - t_a = t_b \end{aligned}$$

Therefore, in any basic nondelay schedule S, no job $j \in \tau$ with $r_j < t_a < r_j + p_j$ can complete at or before the end of the max-full interval t_b.

2.4 Transformation into Basic job system

In this section, we will show that it is sufficient to consider only basic job systems and basic nondelay schedules for the purpose of a worst case analysis. Therefore, we need to show that any job system τ' with a nondelay schedule S' can be transformed into a basic job system τ with a basic nondelay schedule S such that we have $\frac{C_{equ}(S')}{C^*_{equ}(\tau')} \leq \frac{C_{equ}(S)}{C^*_{equ}(\tau)}$ and $\frac{U^*_{[0,t)}(\tau')}{U_{[0,t)}(S',\tau')} \leq \frac{U^*_{[0,t)}(\tau)}{U_{[0,t)}(S,\tau)}$ for all t. To this end, we start with the discussion of a simple modification of a given job system.

Corollary 2.1 *Let τ' be a job system and S' be a schedule for τ' on m identical machines. Job system τ is generated from τ' by dividing an arbitrary job $j \in \tau'$ into two jobs j_1 and j_2 such that $0 < p_{j_1} < p_j$, $r_{j_1} = r_j$, $p_{j_2} = p_j - p_{j_1}$, and $r_{j_2} = r_j + p_{j_1}$ hold. Schedule S is derived from schedule S' by simply starting job j_1 instead of job j and starting job j_2 immediately after the completion of job j_1. Then the inequalities*

$$\frac{C_{equ}(S')}{C^*_{equ}(\tau')} \leq \frac{C_{equ}(S)}{C^*_{equ}(\tau)} \quad \textit{and} \quad \frac{U^*_{[0,t)}(\tau')}{U_{[0,t)}(S',\tau')} \leq \frac{U^*_{[0,t)}(\tau)}{U_{[0,t)}(S,\tau)}$$

hold for all t.

Proof: Note that the completion time of each job $j' \neq \{j, j_1, j_2\}$ is identical in both schedules S and S'. Further, the release date of job j_2 does not interfere with scheduling j_2 immediately after the completion of j_1 if schedule S' is legal.

It is obvious that the schedule S is a legal schedule as no job is started before its release date and no machine is used to execute two jobs at the same time. Therefore,

we have

$$C_{j_1}(S) = C_j(S') - p_{j_2} \quad \text{and} \quad C_{j_2}(S) = C_j(S')$$

Clearly, for the utilization criterion, we have $U_{[0,t)}(S,\tau) = U_{[0,t)}(S',\tau')$ for all t. Further, $U^*_{[0,t)}(\tau) \geq U^*_{[0,t)}(\tau')$ holds for all t as the splitting of job j cannot decrease $U^*_{[0,t)}(\tau')$ for any t. This leads to the first result

$$\frac{U^*_{[0,t)}(\tau')}{U_{[0,t)}(S',\tau')} \leq \frac{U^*_{[0,t)}(\tau)}{U_{[0,t)}(S,\tau)} \quad \text{for all } t.$$

With respect to the equal priority completion time, we observe

$$\begin{aligned} C_{equ}(S') - C_{equ}(S) &= p_j\, C_j(S') - p_{j_1}\, C_{j_1}(S) - p_{j_2}\, C_{j_2}(S) \\ &= \left(p_{j_1} + p_{j_2}\right) C_j(S') - p_{j_1} \left(C_j(S') - p_{j_2}\right) - p_{j_2}\, C_j(S') \\ &= p_{j_1}\, p_{j_2}. \end{aligned}$$

This result is independent of schedule S'. Therefore, it also holds for the optimal schedule. However, the schedule derived from the optimal schedule for job system τ' need not be an optimal schedule for job system τ. Hence, we have

$$C^*_{equ}(\tau') - p_{j_1} p_{j_2} \geq C^*_{equ}(\tau)$$

This leads to the second desired result

$$\frac{C_{equ}(S)}{C^*_{equ}(\tau)} \geq \frac{C_{equ}(S') - p_{j_1} p_{j_2}}{C^*_{equ}(\tau') - p_{j_1} p_{j_2}} \geq \frac{C_{equ}(S')}{C^*_{equ}(\tau')} \geq 1.$$

■

Clearly, splitting a job within or at the end of a full interval produces again a nondelay schedule if the original schedule was already a nondelay schedule.

Next, we transform an arbitrary job system and its nondelay schedule into a basic job system and its basic nondelay schedule.

Corollary 2.2 *For any job system τ', a nondelay schedule S', and an arbitrary but fixed $\epsilon > 0$ there is a basic job system τ and a basic nondelay schedule S such that*

$$\frac{C_{equ}(S')}{C^*_{equ}(\tau')} \leq \frac{C_{equ}(S)}{C^*_{equ}(\tau)} \quad \textit{and} \quad \frac{U^*_{[0,t)}(\tau')}{U_{[0,t)}(S',\tau')} \leq \frac{U^*_{[0,t)}(\tau)}{U_{[0,t)}(S,\tau)}$$

hold for all t.

Proof: Consider a max-full interval $[t_a, t_b)$ in schedule S'.

We assume a job $j \in \tau'$ with $r_j < t_a < r_j + p_j$ and $C_j(S') - p_j \leq r_j + (t_b - t_a)$, that is, job j violates the third property of Definition 2.3 in schedule S'. Therefore, we have

$$C_j(S') - p_j - r_j + t_a \leq t_b \tag{2.4}$$

As it is not allowed for any job to start before its release date, the inequality $r_j \leq C_j(S') - p_j$ holds and leads to

$$C_j(S') - p_j - r_j + t_a \geq t_a \tag{2.5}$$

From Equations (2.4, 2.5), we obtain that $C_j(S') - r_j - p_j + t_a \in [t_a, t_b]$.

Similarly, we obtain immediately $C_j(S') > C_j(S') - p_j - r_j + t_a > C_j(S') - p_j$ as $r_j < t_a < r_j + p_j$ holds for job j. That is, job j is executed in schedule S' at time instance $C_j(S') - r_j - p_j + t_a$. Hence, we split job j in schedule S' at time $C_j(S') - r_j - p_j + t_a$ by using Corollary 2.1. Note that this splitting time is within the full-interval.

Then, we repeatedly apply Corollary 2.1 to all jobs starting in $[t_a, t_b)$ until the first property of Definition 2.3 is valid for all those jobs.

The smallest release date of all jobs starting in $\tau([t_a, t_b])$ is t_a as schedule S' is a nondelay schedule. Therefore, we finally reduce the release date of all jobs starting in $\tau([t_a, t_b])$ to t_a. Clearly, the resulting schedule will again be nondelay.

The same transformations are applied to all other max-full intervals of S'. As each transformation cannot decrease the ratios $\frac{C_{equ}(S')}{C^*_{equ}(\tau')}$ and $\frac{U^*_{[0,t)}(\tau')}{U_{[0,t)}(S',\tau')}$ for any t, we finally have

$$\frac{C_{equ}(S')}{C^*_{equ}(\tau')} \leq \frac{C_{equ}(S)}{C^*_{equ}(\tau)} \quad \text{and} \quad \frac{U^*_{[0,t)}(\tau')}{U_{[0,t)}(S',\tau')} \leq \frac{U^*_{[0,t)}(\tau)}{U_{[0,t)}(S,\tau)}$$

for all t for the resulting basic job system τ and its basic nondelay schedule S. ■

In the remaining part of the thesis, every job of basic job system τ with processing time $p_i \leq \epsilon$ will be called a **short** job while all other jobs are **long** jobs. In a basic nondelay schedule, long jobs are either started at their release dates or immediately

after a max-full interval while short jobs are also started within a max-full interval. Note that the starting time of any long job is always different from the starting time of a short job in the basic nondelay schedule if ϵ is sufficiently small. Without restriction of generality we can assume that all jobs start in order of their release dates in a basic nondelay schedule, that is, job j_1 does not start after job j_2 if $r_{j_1} < r_{j_2}$ holds.

Observe that due to Definition 2.3, at most $m-1$ long jobs can have the same release date in a basic job system. Remember that in a basic schedule, all long jobs with the same release date start either at their release date or at the end of a max-full interval. Therefore, in any basic schedule no more than $m-1$ long jobs with the same release date are executing concurrently at any instant of time. We will use this observation in the proof of the next corollary.

Next, we consider a nondelay schedule for a basic job system where all long jobs start at their release dates. In the following two corollaries, we show that this schedule has an optimal equal priority completion time and optimal utilization for all t, if $\epsilon \to 0$ holds.

Corollary 2.3 *For each basic job system τ with $\epsilon \to 0$ and a nondelay schedule S where all long jobs start at their release dates, $U_{[0,t)}(S,\tau) = U^*_{[0,t)}(\tau)$ holds for all t.*

Proof: We prove this corollary by contradiction. Assume that time t' is the last time instant such that $U_{[0,t)}(S,\tau) = U^*_{[0,t)}(\tau)$ holds for all $t \leq t'$. Hence, there is at least one machine idle in schedule S at time t'. As S is a nondelay schedule, no job j with $r_j \leq t'$ starts in schedule S after time t'. Due to $\epsilon \to 0$, no short job with $r_j \leq t'$ completes after t'. Further, any long job cannot start earlier in any schedule than in schedule S. Therefore, any job that executes at time t' in schedule S does not start earlier in any schedule than in S. This is a contradiction to the assumption. ■

Corollary 2.4 *$C_{equ}(S) = C^*_{equ}(\tau)$ holds for each basic job system τ with $\epsilon \to 0$ and a nondelay schedule S where all long jobs start at their release dates.*

Proof: We start this proof with two simple observations:

Observation[1]: Assume a basic job system τ', a schedule S', and a time instant t' such that on machines i_1 and i_2, either a job starts at t' or no job executes at t'. Then $C_{equ}(S')$ does not change if we move all jobs that execute on machine i_1 and start at time t' or later to machine i_2 and vice versa.

Observation[2]: Let $I = [t_a, t_b]$ be an interval of schedule S' such that only short jobs execute on machine i during I and machine i is busy at any moment of I. Due to $\epsilon \to 0$, we can assume that a short job completes at any given time $t' \in]t_a, t_b]$.

Now, let $\sigma \neq S$ be a schedule for job system τ with $C_{equ}(\sigma) = C^*_{equ}(\tau)$, that is, schedule σ is optimal for such job system. Further, let job j be the first long job that does not start at its release date in the optimal schedule σ. Assume that job j is executed on machine $\mathcal{M}_l$ in σ. We show that there is another optimal schedule in which j starts earlier by considering 4 cases:

1. Some machine is idle in a whole interval $\Big[t, C_j(\sigma) - p_j\Big)$ for some $t < C_j(\sigma) - p_j$. Then schedule σ clearly is not optimal.

2. A short job j_s directly precedes job j on machine $\mathcal{M}_l$. Then we reduce the start time of job j by simply exchanging the execution order of j and j_s on machine $\mathcal{M}_l$. This does not change $C_{equ}(\sigma)$ as $p_{j_s} \leq \epsilon < C_j(\sigma) - p_j - r_j$ holds, if ϵ is small enough. Note that in this case, schedule σ is not optimal if $\Big[C_j(\sigma) - p_j, C_j(\sigma)\Big)$ is not a full interval.

3. A long job directly precedes job j on machine $\mathcal{M}_l$ and a short job completes at time $C_j(\sigma) - p_j$ in schedule σ on machine $\mathcal{M}_s \neq \mathcal{M}_l$. Based on our second observation[2], we simply switch the allocation between machines $\mathcal{M}_l$ and $\mathcal{M}_s$ for all jobs starting in schedule σ on any one of those both machines at time $C_j(\sigma) - p_j$ or later and apply again Case 2.

4. m long jobs execute concurrently in a full interval $\Big[t, C_j(\sigma) - p_j\Big)$ of schedule σ for some $t < C_j(\sigma) - p_j$. Let $\tau_s \subset \tau$ such that for each job $i \in \tau_s$ we have $C_i(\sigma) \geq$

$C_j(\sigma) - p_j$ and $C_i(\sigma) - p_i \leq t$. That is, τ_s denotes this set of m long jobs. Remember that job j is the first long job that starts after its release date in schedule σ. Hence, all jobs in τ_s start at their release dates in the same schedule σ. Further, let $r \leq t$ be the highest release date of any job in τ_s and S' be a basic nondelay schedule for τ. Now, we consider the following two possibilities:

(a) r is not the beginning of a max-full interval in S'. Then at least one machine must be idle at time r in schedule S'. This is a contradiction to the nondelay property of S' as no job from τ_s can complete at time r or earlier in any legal schedule for τ.

(b) $[r, t'_b)$ is a max-full interval in schedule S' for some time $t'_b > r$. Then all jobs from τ_s with release date r start at time t'_b in schedule S' while all other jobs from τ_s start before r and complete after t'_b due to the third property of Definition 2.3. Therefore, t'_b cannot be the end of a max-full interval as m jobs execute at time t'_b in schedule S'.

■

Now, we are able to provide an intuitive reason for the properties of Definition 2.3. The first property guarantees that at the end of a full interval, machines become idle at the same time. Therefore, all long jobs, with the same release date, start as late as possible leading to a worst case behavior. The second property does not change the basic schedule but increases the flexibility of the optimal schedule. This may result in a higher worst case ratio. Finally, the third property guarantees that we cannot split any remaining long job of a basic job system without violating the nondelay property or potentially decreasing the worst case ratio of both criteria utilization and equal priority completion time.

In the remaining parts of the thesis we will use the expression *optimal schedule* when talking of a nondelay schedule for a basic job system with $\epsilon \to 0$, if all long jobs in this schedule are started at their release dates. As we need only one optimal schedule for

our worst case analysis we can ignore all other optimal schedules for these job systems if they exist. Without restriction of generality we assume that in such an optimal schedule, all short jobs also start in order of their release dates. However in an optimal schedule, a long job with a higher release date can start before a short job with a lower release date.

Chapter 3

Online scheduling to maximize utilization

3.1 introduction

The efficient operation of multi-machine systems requires the best possible use of the resources that a system owner provides. The allocation and management of resources in parallel systems is fundamental to sustaining and improving the benefits of multiprocessing, see DOWDY *et al.* [17], and KARATZA [39]. Thus, task scheduling is a key element in achieving high performance from parallel machine systems. The main goal of this chapter is to achieve the highest possible utilization of the machines. To do so effectively, one must answer the question "which criterion is adequate to use?". While this seems like an easy question to be answered, it is often far more difficult to quantify than one might think. An appropriate criterion to reflect the true utilization performance of parallel machine systems has not been established yet. However, one common criterion for assessing a schedule, and thus for measuring system utilization is makespan. The usual goal of scheduling has been to achieve a small makespan. In the previous chapter, we have already provided two new alternative optimality criteria, termed *utilization* and *equal priority completion time*, to evaluate how much the machines are utilized. Further, we have shown that those criteria are better than the well-known and commonly used makespan. Whereas we demonstrated that they more accurately capture the machine utilization if all machines are identical, and thus more

accurately represent the quality of the schedule and the system utilization performance being achieved. Consequently, our optimality criteria are well adequate to describe quantitatively the utilization of the machines. The research in this chapter is focused on studying the online competitiveness for scheduling an identical-machines system non-preemptively in order to maximize the gain of such system when the performance measure is our first *utilization* criterion.

As we mentioned above, the formal form of the utilization criterion $U_{[t_1,t_2)}(S,\tau)$ is introduced in Section 2.2. It represents the resource (machines) consumption of the job system τ within the interval $[t_1,t_2)$ of schedule S. Intuitively, to obtain the highest possible gain of the machines one has to pay much attention to concentrate on a bottleneck time interval of the machines, *i.e.* the interval of time in which most of the machine consumers submit their jobs. Therefore, we devote this chapter to apply the first optimality criterion *utilization* for our nonclairvoyant online scenario to maximize machine utilization particularly during such specific (target) time interval. More precisely, our scheduling objective considers the analysis of a nondelay, non-preemptive, non-clairvoyant online schedule that maximize such utilization criterion from system start until a given end time (*i.e.* during the productive time of the machines). In this work, we determine the ratio between the worst nondelay schedule and the one that has the highest attainable utilization. That is, we are interested to consider the performance guarantees.

Depending on the results of the previous chapter, we need only to analyze basic job systems with $\epsilon \to 0$, their basic nondelay schedules, and their optimal schedules in order to determine a worst case deviation. Remember that in any optimal schedule of a basic job system, all long jobs start at their release dates. As will be shown later, the main result of this chapter will be summarized in Theorem 3.1. This theorem gives the worst case of the ratio $\frac{U^*_{[0,t)}(\tau)}{U_{[0,t)}(S,\tau)}$ for any time instant $t \geq 0$.

3.2 Scheduling jobs online with unknown size

While scheduling problems in general have received a lot of interest in the past, most considered models assume that the input data is completely or partially known. However, in the design of real-time systems, it is often the case that certain job parameters, such as the execution time, are not known in advance. Thus, the challenge in real-life system design is to consider more realistic models that efficiently confront the requirements of such uncertainty. This lack of knowledge (incompleteness) is a significant impediment in the scheduler's task, as one might expect. In this situation, the scheduler has to coordinate the jobs over time non-clairvoyantly. The study of non-clairvoyant scheduling algorithms was initiated by MOTWANI *et al.* [58].

A non-clairvoyant online scheduler makes scheduling decisions at run time having no complete knowledge of jobs, that is, it construct a schedule partially over time. Typically, it does not possess a prior knowledge about the occurrence of future submissions and even it has no idea about the required execution times of the existence jobs (scheduling jobs non-clairvoyantly). The scheduler recognizes only that the jobs arrive or that the jobs are completed. Such kind of schedulers are used in many practical systems. A typical online problem for non-clairvoyant scheduling might be found in electronic commerce applications.

This uncertainty of knowledge in problem data is of both theoretical and practical significance. From an empirical perspective, system designers have used worst case values in order to overcome the non-determinism of execution time values PINEDO [65]. However, the assumption that every job will have an execution time equal to the maximum value in its allowable range is unrealistic and at the same time, may cause constraint violation at run-time. On the other side, the estimates of how long each job will run are used to figure out for the system owner when additional machines will become available. Of course, the source of these estimates is typically the customer who runs the job. However, it may be a cumbersome for the customers to provide in advance reliable estimates. Further, comparisons of customer estimates with real run times show that these estimates tend to be inaccurate, even when customers are

requested to provide their best possible estimates for batch jobs, as has been shown in [50, 59]. Moreover, several attempts to derive better estimates automatically based on historical information from previous runs have not been successful, as too many under-estimations have been faced.

Having no knowledge of the jobs being scheduled (non-clairvoyance) one would not expect to obtain an optimal solution. In our performance evaluation, we wish to compare the *worst case* utilization performance of non-clairvoyant online scheduler with respect to the optimal off-line scheduler, which has complete knowledge about the whole input instance. That is, we are interested in performance guarantees.

Obviously, the problem of scheduling a collection of dynamically arriving jobs with unknown execution times is that scheduling decisions have a potentially large, persistent, and unpredictable impact on the future. Specifically, when some new jobs arrive at the same time with unknown processing times, the system owner face the following dilemma after random selecting of a job for processing:

- Scheduling such job immediately on an ideal machine will utilize unused machines, so it is good.

- however, if this job is followed by a longer job. and this long job will block other jobs in the future, it may lead to more future loss than current gain. So it is a benefit if the vacant machine is occupied first by the longer job.

As the future is usually unknown and the non-clairvoyance is dark, there is no job selection strategy guarantee finding a solution for this dilemma. Therefore, it is reasonable to schedule the available jobs immediately on the idle machines to keep all machines as busy as possible, risking that some long jobs will block other future jobs. Clearly, the assignment of a job to a machine is postponed until a machine becomes available. As there is no information about the machine occupation of any waiting job is available, any arbitrary policy can be used to pick any one from jobs that are available at the same time. In general, this technique which allocate all vacant machines to the jobs waiting for processing is known as nondelay policy. Although a system owner

uses nondelay schedules, he may suffer from some idleness that may be produced by an unfortunate selection of jobs. Therefore, he may be interested in the ratio between the utilizations of schedules with an optimal and a worst case job selection. In particular, his goal is to compare the utilization performance of online nondelay schedules against that of an optimal off-line (or clairvoyant) schedule.

3.3 Productive interval of machines

As commercial multi-machine systems become more popular, there is a growing need for accurate (efficient) evaluation of the utilization of these expensive resources. Indeed, many researchers investigate scheduling models from a system point of view, asking what the system can do to improve system utilization, but disregarding the effect on some important issues that relate to the customers. Sometimes, ignorance of such significant issues may lead to unreasonable investments. Consequently, we consider an additional and effective issue by taking into account a viewpoint of customer's requirements, and ask how much the utilization can be improved with the observation that most machine consumers submit their jobs within a specific time interval and they do not have the patience to receive their completed jobs after such specific interval with much delay, that is, in other words, they ( ) the system provider to execute their jobs within such a specific time interval otherwise they resubmit the jobs to another system.

Clearly, the utilization criterion measures the fraction of time that the machines spend on executing jobs during the elapsed time. Which performance measure is adequate in a given situation depends, of course, upon the application. For example, utilization may be a reasonable measure in situations where customer pay at a uniform rate for the use of the machine, however the owner only is paid if customer's job is completed within this time frame.

In real world, the following scenario may occur in electronic commerce: Let us presume that many customers submit their jobs dynamically over time to a system consisting of some number of identical parallel machines. There is no relation between those customers, that is, they are independent from each other. A customer leaves his

job with the system provider essentially committing to future payment for a completed job. More precisely, he will pay a fixed amount to the system provider for each minute his job occupies a machine. However, The customer does not get paid for machine utilization after a specific target time. The system owner has flexibility in deciding when to run jobs, however he must use the machines non-preemptively. That means the customer's job always occupies its assigned machine exclusively for its whole execution time without any interruption or termination. This is a reasonable assumption because of the large overhead for the jobs on the parallel machines. Further, in many applications, any interruption in processing once a job begins might cause permanent ruin to it. Due to security reasons no two jobs are allowed to share the same machine during execution. The machines consumers are likely unaware of the real execution times of their jobs.

In this scenario, most customers only submit jobs during core business hours. Intuitively, the machine consumers have an option of choosing the machine provider that best meet their requirements within such a desired time frame (core business hours). Otherwise, potential customers will switch to a competitor rather than waiting for their jobs being completed much later during off business hours. On the other side, the machine provider has the flexibility in deciding which jobs to run, as well as when to run them. However, he is aware that the demand for his resources is likely to be very limited during the rest of the day (off business hours). In other words, a large percentage of the machines will be vacant during this interval of time. Of course, the system owner wants to obtain the best possible return on his investment. Therefore, it is reasonable to assume that he is interested to maximize the utilization of his machines during those hours of core business. To achieve that effectively, he has to try to perform the customer's requests within the required time frame aiming at attracting customers.

Recently, KEMPF *et al.* [45] describe a number of different considerations that must be taken into account when assessing the quality of a schedule. The authors have shown that Looking at utilization only on bottleneck machines makes a lot more sense since one is maximizing utilization where it matters, instead of indiscriminately across

the whole schedule. Further, they corroborate this added issue, and show conclusively that concentrating on bottleneck interval can be very effective to improve system utilization. Consequently, this motivated us to consider the system utilization from system start to target time, *i.e.* during the productive time interval.

3.4 An upper bound for the utilization

This section provides the main result of this chapter. Clearly, the execution of some jobs can be delayed by other jobs that are not completed before the release date of the new jobs. Hence, for any time instant t, we are interested in the difference between the resource consumption after t by the basic schedule S and the optimal schedule σ in relation to t. In the remainder of this chapter, we will use a notation T to be a large time instant that greater than the makespan of any schedule under consideration. For the basic job system τ with $\epsilon \to 0$, its basic nondelay schedule S and its optimal schedule σ, we want to determine the difference

$$\begin{aligned} D_t(S,\tau) &= U^*_{[0,t)}(\tau) - U_{[0,t)}(S,\tau) \\ &= U_{[0,t)}(\sigma,\tau) - U_{[0,t)}(S,\tau) = U_{[t,T)}(S,\tau) - U_{[t,T)}(\sigma,\tau) \end{aligned}$$

for some time instant $t > 0$. Intuitively, this value describes the sum of the machine resources that are not busy executing jobs from τ in schedule S before time t while they are used to process jobs from τ before time t in the optimal schedule σ. From this definition, we obtain the following relation for any time instant $t' < t$:

$$\begin{aligned} D_t(S,\tau) - D_{t'}(S,\tau) &= \Big(U_{[t,T)}(S,\tau) - U_{[t,T)}(\sigma,\tau)\Big) - \Big(U_{[t',T)}(S,\tau) - U_{[t',T)}(\sigma,\tau)\Big) \\ &= \Big(U_{[t',T)}(\sigma,\tau) - U_{[t,T)}(\sigma,\tau)\Big) - \Big(U_{[t',T)}(S,\tau) - U_{[t,T)}(S,\tau)\Big) \end{aligned}$$

Therefore, we have

$$D_t(S,\tau) = D_{t'}(S,\tau) + U_{[t',t)}(\sigma,\tau) - U_{[t',t)}(S,\tau) \tag{3.1}$$

An upper bound of $D_t(S,\tau)$ for a basic job system τ with $\epsilon \to 0$ is given by the following lemma.

Lemma 3.1 *$D_t(S,\tau) \leq \frac{1}{4}U^*_{[0,t)}(\tau)$ holds for each basic job system τ, its basic nondelay schedule S and every time instant $t \geq 0$.*

Proof: We prove this lemma by induction on the number k of different release dates. The lemma trivially holds for $k = 0$, that is, if τ is empty. Therefore, we assume that it is true for all basic job systems with at most k different release dates. Then we consider a basic job system τ with $k+1$ different release dates.

For this proof, we need to introduce some additional notations such as:

- $r = \max\{r_j | j \in \tau\}$ the last release date in the job system τ,
- $\tau_r = \{j \in \tau | r_j = r\}$ the set of all jobs with the last release date,
- $\tau_k = \{j \in \tau | r_j < r\} = \tau \backslash \tau_r$ the set of all other jobs,
- $t_S = \max\{r, t_b(S)\}$. Where $t_b(S)$ is the end of the last max-full interval in S,
- $t_\sigma = \max\{r, t_b(\sigma)\}$. Similarly, $t_b(\sigma)$ is the end of the last max-full interval in σ.

Note that τ_k is a basic job system with k different release dates and that every job $j \in \tau_k$ starts before time r in the basic nondelay schedule S. Schedules σ and σ_k are the optimal schedules for job systems τ and τ_k, respectively, while S_k is the basic nondelay schedule for the job system τ_k. Clearly, schedules σ and σ_k are identical in the time interval $[0, r)$ as no job $j \in \tau_r$ can start before time r in any schedule and all long jobs start at their release dates in both schedules. Similarly, schedules S and S_k are identical for all jobs that belong to job system τ_k.

In the first step, we prove that it is sufficient to determine $D_{t_\sigma}(S,\tau)$. Due to our induction assumption, and as schedules S and S_k are identical for all jobs in the job system τ_k, there is

$$D_t(S,\tau) = D_t(S,\tau_k) = D_t(S_k,\tau_k) \leq \frac{1}{4}U^*_{[0,t)}(\tau_k) = \frac{1}{4}U^*_{[0,t)}(\tau) \quad \text{for all } \; t \leq r. \tag{3.2}$$

As no additional jobs are released after time r and S is a nondelay schedule, there are at least as many machines idle at time t_2 as at time t_1 in schedule S with $r \leq t_1 < t_2$.

Let $[t_1, t_2)$ be a subinterval of $[r, t_\sigma)$ such that no machine becomes idle in (t_1, t_2) of schedule S and let the number of busy machines in $[t_1, t_2)$ be m_{t_1}. Note that $t_2 \leq t_\sigma$. Then we have

$$\begin{aligned} U_{[0,t_2)}(\sigma,\tau) &= U_{[0,t_1)}(\sigma,\tau) + U_{[t_1,t_2)}(\sigma,\tau) \\ &= U_{[0,t_1)}(\sigma,\tau) + m(t_2 - t_1), \end{aligned}$$

Similarly, and from Equation 3.1 we get

$$\begin{aligned} D_{t_2}(S,\tau) &= D_{t_1}(S,\tau) + U_{[t_1,t_2)}(\sigma,\tau) - U_{[t_1,t_2)}(S,\tau) \\ &= D_{t_1}(S,\tau) + m(t_2 - t_1) - m_{t_1}(t_2 - t_1). \end{aligned}$$

This leads to

$$\frac{D_{t_2}(S,\tau)}{U_{[0,t_2)}(\sigma,\tau)} = \frac{D_{t_1}(S,\tau) + (m - m_{t_1})(t_2 - t_1)}{U_{[0,t_1)}(\sigma,\tau) + m(t_2 - t_1)}.$$

Therefore, exactly one of the two following sequences of inequalities is valid:

$$\frac{D_{t_1}(S,\tau)}{U_{[0,t_1)}(\sigma,\tau)} < \frac{D_{t_2}(S,\tau)}{U_{[0,t_2)}(\sigma,\tau)} < 1 - \frac{m_{t_1}}{m}$$

or

$$\frac{D_{t_1}(S,\tau)}{U_{[0,t_1)}(\sigma,\tau)} \geq \frac{D_{t_2}(S,\tau)}{U_{[0,t_2)}(\sigma,\tau)} \geq 1 - \frac{m_{t_1}}{m}$$

As the value m_{t_1} is decreasing monotonically with growing time t_1 in the interval $[r, t_\sigma)$, this results in

$$\max_{t \in [r,t_\sigma)} \left\{ \frac{D_t(S,\tau)}{U_{[0,t)}(\sigma,\tau)} \right\} = \max \left\{ \frac{D_r(S,\tau)}{U_{[0,r)}(\sigma,\tau)}, \frac{D_{t_\sigma}(S,\tau)}{U_{[0,t_\sigma)}(\sigma,\tau)} \right\}.$$

Further, we have $U_{[t_\sigma,t)}(S,\tau) \geq U_{[t_\sigma,t)}(\sigma,\tau)$ with $t > t_\sigma$ as no long job can complete earlier in schedule S than in the optimal schedule σ and no job starts at time t_σ or later in schedule σ. This with the Equation 3.1 leads to

$$D_t(S,\tau) = D_{t_\sigma}(S,\tau) + U_{[t_\sigma,t)}(\sigma,\tau) - U_{[t_\sigma,t)}(S,\tau) \leq D_{t_\sigma}(S,\tau) \quad \text{for all } t > t_\sigma.$$

Therefore, it is sufficient to determine $\frac{D_{t_\sigma}(S,\tau)}{U_{[0,t_\sigma)}(\sigma,\tau)}$.

Next, we address two cases:

1. $t_b(\sigma_k) \leq r$.

   Due to $t_\sigma \leq t_S$, the interval $[r, t_\sigma)$ is full interval in both schedules S and σ. Hence, we have $U_{[r,t_\sigma)}(\sigma, \tau) = U_{[r,t_\sigma)}(S, \tau)$. This with Equation 3.1 results in

   $$D_{t_\sigma}(S, \tau) = D_r(S, \tau) + U_{[r,t_\sigma)}(\sigma, \tau) - U_{[r,t_\sigma)}(S, \tau) = D_r(S, \tau).$$

   Clearly, $U_{[0,t_\sigma)}(\sigma, \tau) \geq U_{[0,r)}(\sigma, \tau)$. Therefore, we have

   $$\frac{D_{t_\sigma}(S, \tau)}{U_{[0,t_\sigma)}(\sigma, \tau)} \leq \frac{D_r(S, \tau)}{U_{[0,r)}(\sigma, \tau)} = \frac{D_r(S_k, \tau_k)}{U_{[0,r)}(\sigma_k, \tau)}$$

   Next, we assume that $t_\sigma > t_S$ holds and have

   $$U_{[t_1,t_2)}(S, \tau_k) = U_{[t_1,t_2)}(S_k, \tau_k) \geq U_{[t_1,t_2)}(\sigma_k, \tau_k) = U_{[t_1,t_2)}(\sigma, \tau_k)$$

   for all $r \leq t_1 < t_2$ as no short job from τ_k completes after time r in schedules σ and σ_k.

   We define

   $$m_\sigma = \frac{U_{[t_\sigma,T)}(S, \tau_r) - U_{[t_\sigma,T)}(\sigma, \tau_r)}{t_S - r}.$$

   For each long job $j \in \tau_r$, its contribution to the term $U_{[t_\sigma,T)}(S, \tau_r) - U_{[t_\sigma,T)}(\sigma, \tau_r)$ is upper bounded by $t_S - r$ as all those jobs start at time t_S in the basic schedule S and at their release date r in the optimal schedule σ, respectively. Therefore, the value m_σ is at most the number of long jobs from job system τ_r that complete after time t_σ in schedule S. Hence, at most the machine product $(t_\sigma - t_S)(m - m_\sigma)$ can be idle in time interval $[r, t_\sigma)$ in schedule S. This results in

   $$\begin{aligned} m(t_\sigma - r) &= U_{[r,t_\sigma)}(\sigma, \tau) \\ &= U_{[r,t_\sigma)}(\sigma, \tau_k) + U_{[r,t_\sigma)}(\sigma, \tau_r) \\ &= U_{[r,t_\sigma)}(\sigma, \tau_k) + \sum_{j \in \tau_r} p_j - U_{[t_\sigma,T)}(\sigma, \tau_r) \qquad \text{and} \end{aligned}$$

   $$m(t_\sigma - r) \leq U_{[r,t_\sigma)}(S, \tau_k) + \sum_{j \in \tau_r} p_j - U_{[t_\sigma,T)}(S, \tau_r) + (t_\sigma - t_S)(m - m_\sigma).$$

Therefore, we have

$$U_{[r,t_\sigma)}(\sigma,\tau_k)+U_{[t_\sigma,T)}(S,\tau_r)-U_{[t_\sigma,T)}(\sigma,\tau_r)\leq U_{[r,t_\sigma)}(S,\tau_k)+(t_\sigma-t_S)(m-m_\sigma).$$

With the definition of m_σ, this leads to the inequality

$$\begin{aligned} U_{[r,t_\sigma)}(\sigma,\tau_k)+(t_S-r)m_\sigma &= U_{[r,t_\sigma)}(\sigma,\tau_k)+U_{[t_\sigma,T)}(S,\tau_r)-U_{[t_\sigma,T)}(\sigma,\tau_r) \\ &\leq U_{[r,t_\sigma)}(S,\tau_k)+(t_\sigma-t_S)(m-m_\sigma) \end{aligned}$$

Then, we get

$$m_\sigma(t_\sigma-r)\leq U_{[r,t_\sigma)}(S,\tau_k)-U_{[r,t_\sigma)}(\sigma,\tau_k)+(t_\sigma-t_S)m$$

and this yields

$$m_\sigma \leq \frac{(t_\sigma-t_S)\,m+\Delta}{t_\sigma-r} \qquad \text{with} \tag{3.3}$$

$$\Delta = U_{[r,t_\sigma)}(S,\tau_k)-U_{[r,t_\sigma)}(\sigma,\tau_k)=D_r(S,\tau_k)-D_{t_\sigma}(S,\tau_k), \tag{3.4}$$

see Equation 3.1. Remember that all jobs from τ_k must start before r in schedules σ and S. Therefore, we have

$$\Delta = U_{[r,t_\sigma)}(S,\tau_k)-U_{[r,t_\sigma)}(\sigma,\tau_k)\geq 0.$$

Now, we obtain a simple optimization of Inequality 3.3. From this inequality, we have

$$(t_S-r)m_\sigma\leq(t_S-r)\left(\frac{(t_\sigma-t_S)\,m+\Delta}{t_\sigma-r}\right) \tag{3.5}$$

By Partially differentiating the R.H.S with respect to t_S we obtain

$$\frac{1}{t_\sigma-r}\cdot\Big((t_\sigma-t_S)m+\Delta-(t_S-r)\,m\Big)=0$$

Therefore, the R.H.S of Inequality 3.5 is maximized for $t_S=\frac{t_\sigma+r}{2}+\frac{\Delta}{2m}$. This leads to

$$t_S-r=\frac{t_\sigma-r}{2}+\frac{\Delta}{2m} \quad \text{and} \quad t_\sigma-t_S=\frac{t_\sigma-r}{2}-\frac{\Delta}{2m}$$

By substituting these values into the Inequality 3.5, we obtain

$$\begin{aligned}
(t_S - r)m_\sigma &\leq \left(\frac{t_\sigma - r}{2} + \frac{\Delta}{2m}\right)\left(\frac{\left(\frac{t_\sigma - r}{2} - \frac{\Delta}{2m}\right)m + \Delta}{t_\sigma - r}\right) \\
&= \left(\frac{t_\sigma - r}{2} + \frac{\Delta}{2m}\right)\left(\frac{m}{2} + \frac{\Delta}{2(t_\sigma - r)}\right) \\
&= \frac{m\,(t_\sigma - r)}{4} + \frac{\Delta}{4} + \frac{\Delta}{4} + \frac{\Delta^2}{4m(t_\sigma - r)} \\
&= \frac{m\,(t_\sigma - r)}{4} + \frac{\Delta}{2}\left(1 + \frac{\Delta}{2m\,(t_\sigma - r)}\right) \\
&= \frac{1}{4}U_{[r,t_\sigma)}(\sigma,\tau) + \frac{\Delta}{2}\left(1 + \frac{\Delta}{2m\,(t_\sigma - r)}\right).
\end{aligned}$$

With this result and the definition of m_σ (3.3) and Equation (3.4), we can obtain

$$\begin{aligned}
D_{t_\sigma}(S,\tau) &= U_{[t_\sigma,T)}(S,\tau) - U_{[t_\sigma,T)}(\sigma,\tau) \\
&= U_{[t_\sigma,T)}(S,\tau_k) + U_{[t_\sigma,T)}(S,\tau_r) - U_{[t_\sigma,T)}(\sigma,\tau_k) - U_{[t_\sigma,T)}(\sigma,\tau_r) \\
&= D_{t_\sigma}(S,\tau_k) + U_{[t_\sigma,T)}(S,\tau_r) - U_{[t_\sigma,T)}(\sigma,\tau_r) \\
&= D_{t_\sigma}(S,\tau_k) + (t_S - r)m_\sigma \\
&= D_r(S,\tau_k) - \Delta + (t_S - r)m_\sigma \\
&\leq \frac{1}{4}U^*_{[0,r)}(\tau) - \Delta + \frac{1}{4}U_{[r,t_\sigma)}(\sigma,\tau) + \frac{\Delta}{2}\left(1 + \frac{\Delta}{2m\,(t_\sigma - r)}\right) \\
&= \frac{1}{4}U^*_{[0,t_\sigma)}(\tau) - \frac{\Delta}{2}\left(1 - \frac{\Delta}{2m\,(t_\sigma - r)}\right).
\end{aligned}$$

As at most $(m-1)$ long jobs from τ_k execute concurrently in schedule S, we have $U_{[r,t_\sigma)}(S,\tau_k) < m(t_\sigma - r)$. This yields

$$\Delta = U_{[r,t_\sigma)}(S,\tau_k) - U_{[r,t_\sigma)}(\sigma,\tau_k) \leq U_{[r,t_\sigma)}(S,\tau_k) < 2m(t_\sigma - r).$$

Finally, with this result and the last result, we get

$$D_{t_\sigma}(S,\tau) \leq \frac{1}{4}U^*_{[0,t_\sigma)}(\tau).$$

This completes the proof of the first case $t_b(\sigma_k) \leq r$. Next we start to consider the second case:

2. $t_b(\sigma_k) > r$.

   In this case, we transform the basic job system τ into another basic job system τ' with the basic nondelay schedule S' such that we have

   - $D_{t_\sigma}(S, \tau) \leq D_{t_\sigma}(S', \tau')$ and $U^*_{[0,t_\sigma)}(\tau) \geq U^*_{[0,t_\sigma)}(\tau')$
   - τ' either has only k different release dates or Case 1 applies to τ'.

   To this end, we again distinguish four cases:

   (a) There is no long job in τ_r or $t_S = r$ holds, that is, all long jobs with release date r start at time r in schedule S. Then we have $D_{t_\sigma}(S, \tau) = D_{t_\sigma}(S, \tau_k)$ and $U^*_{[0,t_\sigma)}(\tau) \geq U^*_{[0,t_\sigma)}(\tau_k)$ as no short job completes after t_σ in both schedules S and σ and $C_j(S) = C_j(\sigma)$ holds for all long jobs $j \in \tau_r$. This leads to

   $$\frac{D_{t_\sigma}(S, \tau)}{U^*_{[0,t_\sigma)}(\tau)} \leq \frac{D_{t_\sigma}(S, \tau_k)}{U^*_{[0,t_\sigma)}(\tau_k)}.$$

   (b) $C_j(S) \leq t_\sigma$ holds for a long job $j \in \tau_r$. Then job j is split into short jobs with the same release date r. This transformation results in the new job system τ' and schedules S' and σ' with $t_{S'} \geq t_S$ and $t_{\sigma'} = t_\sigma$ as the number of short jobs that are released at time r increased and the interval $[r, t_\sigma)$ is a full interval in schedule σ. This results in $C_{j'}(S') \geq C_j(S)$ and $C_{j'}(\sigma') = C_j(\sigma)$ for each long job $j \in \tau_r$ and its corresponding long job $j' \in \tau'_r$. Note that this splitting operation does not change the completion time of any long job from τ_k. This leads to

   $$\begin{aligned} U_{[t_\sigma,T)}(S', \tau') &\geq U_{[t_\sigma,T)}(S, \tau) \quad \text{and} \\ U_{[t_\sigma,T)}(\sigma', \tau') &= U_{[t_\sigma,T)}(\sigma, \tau). \end{aligned}$$

   Hence, we have

   $$\frac{D_{t_\sigma}(S', \tau')}{U^*_{[0,t_\sigma)}(\tau')} \geq \frac{D_{t_\sigma}(S, \tau)}{U^*_{[0,t_\sigma)}(\tau)}.$$

(c) $t_S > r$ holds and there are long jobs $j_1 \in \tau_r$ with $C_{j_1}(S) > t_\sigma$ and $j_2 \in \tau_k$ with $t_S \leq C_{j_2}(S) \leq t_\sigma$. Then we create job system τ' from the job system τ by replacing j_1 and j_2 with jobs j_1' and j_2' such that

$$r_{j_1'} = r_{j_1} = r\,, \qquad p_{j_1'} = \max\{p_{j_2} + r_{j_2} - r, 0\} \qquad \text{and}$$
$$r_{j_2'} = r_{j_2}\,, \qquad p_{j_2'} = p_{j_1} + p_{j_2} - p_{j_1'} = \min\{r - r_{j_2}, p_{j_2}\} + p_{j_1}.$$

Figures (3.1) and (3.2) illustrate the transformation of the schedules S and σ of τ into the new schedules S' and σ' of τ' when $p_{j_2} > r - r_{j_2}$. While Figures (3.3) and (3.4) show the same transformation with $p_{j_2} < r - r_{j_2}$ *i.e.* when $p_{j_1'} = 0$. Observe that each long job that belongs to a basic job system starts at its release date in the optimal schedule. Hence, in the new optimal schedule σ', we can obtain

$$\begin{aligned} C_{j_1'}(\sigma') &= r_{j_1'} + p_{j_1'} = r + \max\Big\{p_{j_2} + r_{j_2} - r, 0\Big\} \\ &= \max\Big\{p_{j_2} + r_{j_2}, r\Big\} = \max\Big\{C_{j_2}(\sigma), r\Big\} \end{aligned}$$

and

$$\begin{aligned} C_{j_2'}(\sigma') &= r_{j_2'} + p_{j_2'} = r_{j_2} + \min\Big\{r - r_{j_2}, p_{j_2}\Big\} + p_{j_1} \\ &= \min\Big\{r, r_{j_2} + p_{j_2}\Big\} + p_{j_1} \\ &= \min\Big\{C_{j_1}(\sigma), C_{j_2}(\sigma) + p_{j_1}\Big\} \\ &\leq C_{j_1}(\sigma). \end{aligned}$$

which lead to

$$U_{[t_\sigma,T)}(\sigma', \tau') \leq U_{[t_\sigma,T)}(\sigma, \tau) \tag{3.6}$$

due to $C_{j_2}(\sigma) \leq C_{j_2}(S) \leq t_\sigma$. Note that this transformation can not increase t_σ as the completion times of jobs j_1' and j_2' in the new schedule σ' are at most the completion times of j_2 and j_1 in schedule σ respectively, see Figures (3.1) and (3.3). For the basic schedules S and S', we need to compare the completion times $C_{j_2'}(S')$ and $C_{j_1'}(S')$ with $C_{j_1}(S)$ and $C_{j_2}(S)$, respectively.

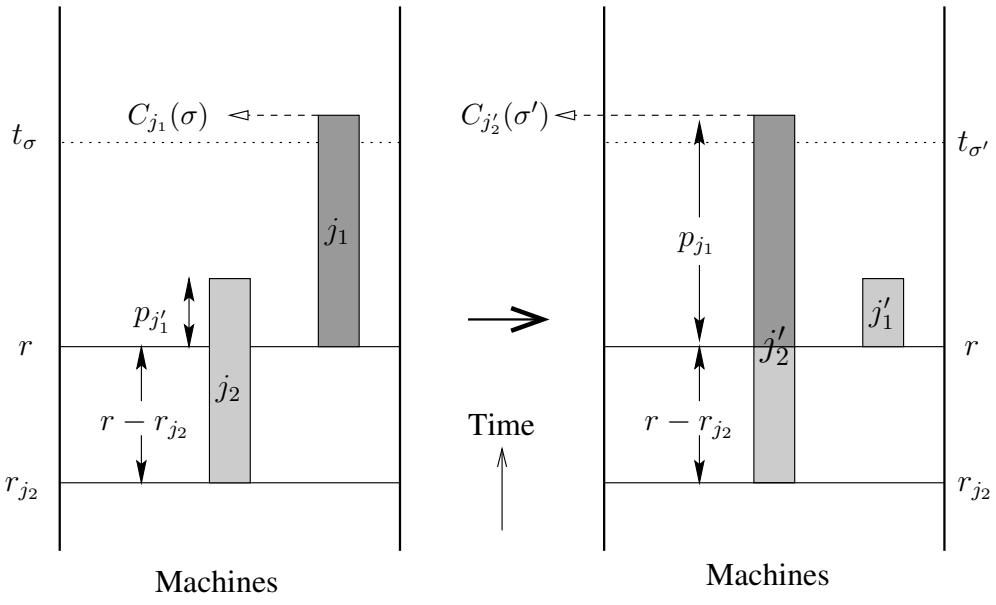

Figure 3.1: *Optimal Schedule σ of τ (left) and new optimal schedule σ' of τ' (right) when $p_{j'_2} > r - r_{j_2}$.*

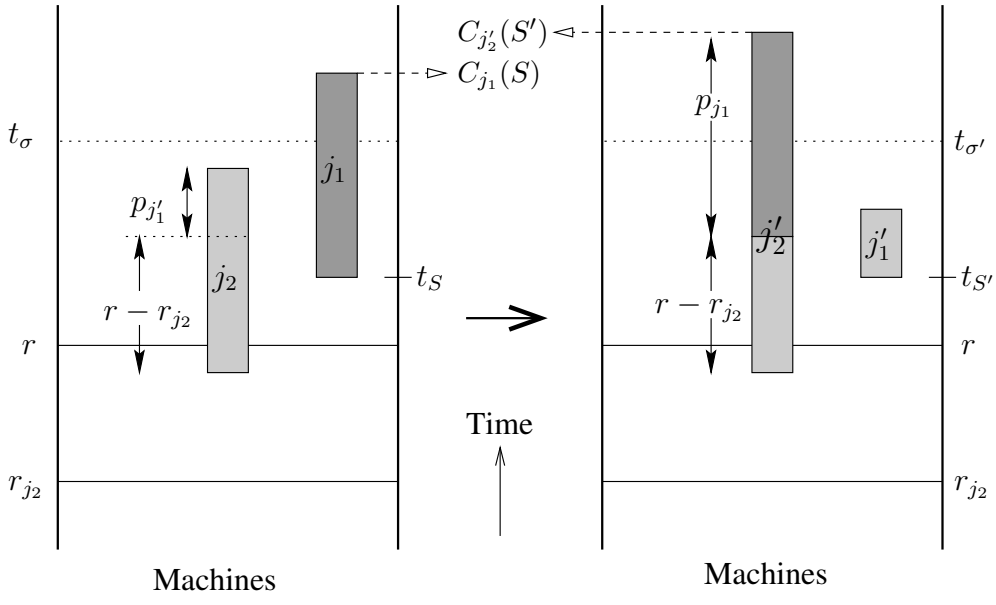

Figure 3.2: *Basic Schedule S of τ (left) and new basic S' of τ' (right) when $p_{j'_2} > r - r_{j_2}$*

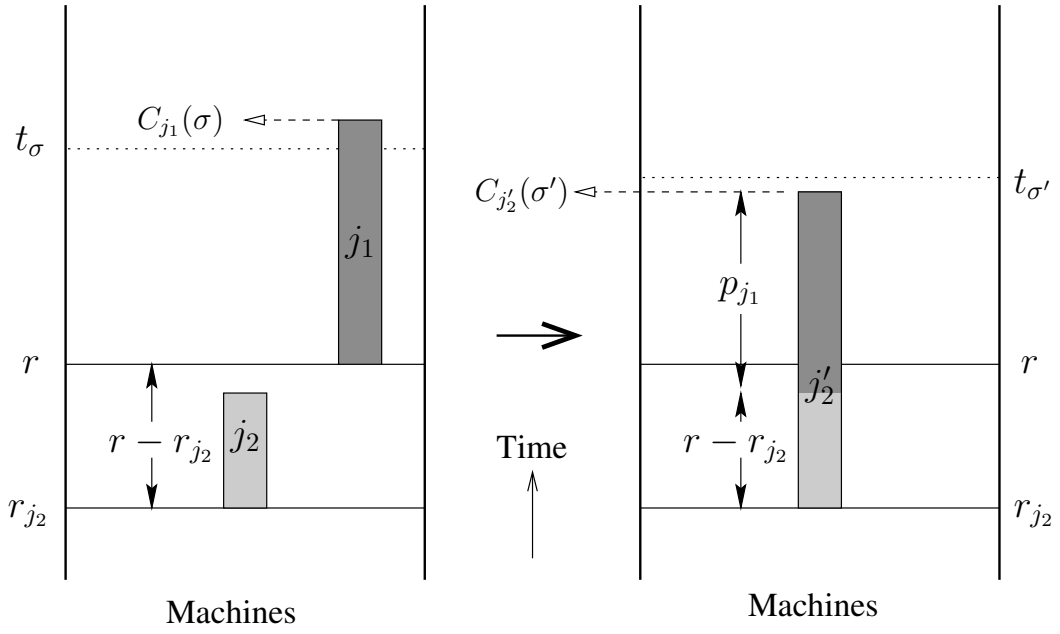

Figure 3.3: *Optimal Schedule σ of τ (left) and new optimal schedule σ' of τ' (right) when* $p_{j'_1} = 0$.

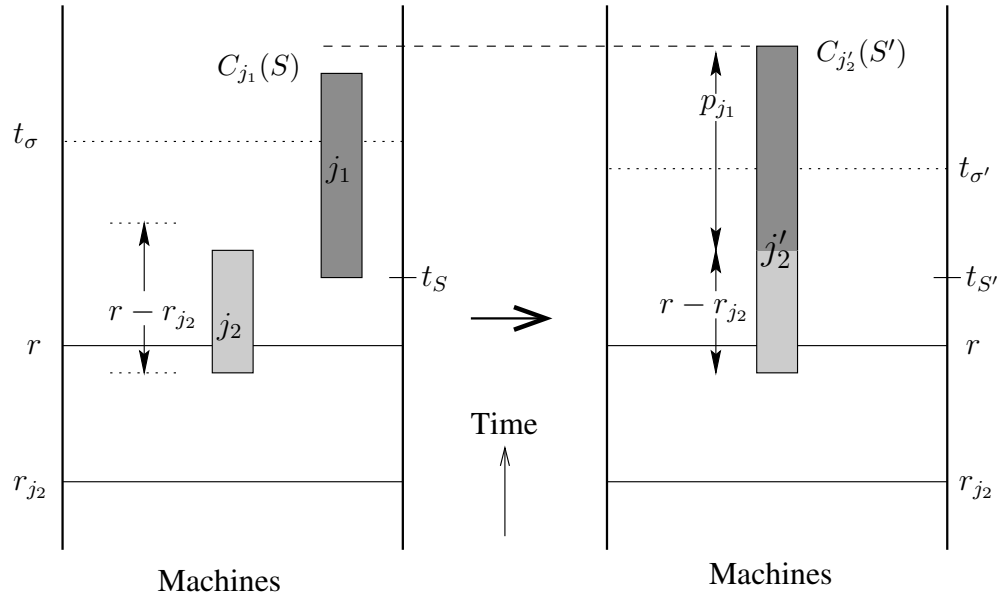

Figure 3.4: *Basic Schedule S of τ (left) and new basic schedule S' of τ' (right) when* $p_{j'_1} = 0$.

Remember that all long jobs with the same release date start at the end of a max-full interval in any basic schedule. Therefore, the starting times of job j_2 in schedule S and job j_2' in the new basic nondelay schedule S' are identical, that is, we have

$$C_{j_2}(S) - p_{j_2} = C_{j_2'}(S') - p_{j_2'},$$

similarly, for jobs j_1 and j_1' there is

$$C_{j_1}(S) - p_{j_1} = C_{j_1'}(S') - p_{j_1'} = t_S.$$

Further remember that $t_S - r < C_{j_2}(S) - p_{j_2} - r_{j_2}$ follows directly from the third property of Definition 2.3. This results in

$$\begin{aligned} C_{j_2'}(S') &= C_{j_2}(S) - p_{j_2} + p_{j_2'} \\ &= C_{j_2}(S) + p_{j_1} + \min\Big\{r - r_{j_2} - p_{j_2}, 0\Big\} \\ &\geq C_{j_2}(S) + r - r_{j_2} - p_{j_2} + p_{j_1} \\ &> t_S + p_{j_1} = C_{j_1}(S). \end{aligned}$$

Similarly, we have

$$\begin{aligned} C_{j_1'}(S') &= C_{j_1}(S) - p_{j_1} + p_{j_1'} \\ &= C_{j_1}(S) + p_{j_2} - p_{j_2'} \\ &= C_{j_1}(S) + C_{j_2}(S) - C_{j_2'}(S') \\ &\leq C_{j_2}(S). \end{aligned}$$

Therefore, we obtain

$$U_{[t_\sigma,T)}(S', \tau') \geq U_{[t_\sigma,T)}(S, \tau) \tag{3.7}$$

From Equations (3.6) and (3.7), we obtain

$$\begin{aligned} D_{t_\sigma}(S', \tau') &= U_{[t_\sigma,T)}(S', \tau') - U_{[t_\sigma,T)}(\sigma', \tau') \\ &\geq U_{[t_\sigma,T)}(S, \tau) - U_{[t_\sigma,T)}(\sigma, \tau) \\ &= D_{t_\sigma}(S, \tau) \end{aligned}$$

This results in

$$\frac{D_{t_\sigma}(S', \tau')}{U^*_{[0,t_\sigma)}(\tau')} \geq \frac{D_{t_\sigma}(S, \tau)}{U^*_{[0,t_\sigma)}(\tau)}.$$

(d) $t_S > r$ holds and there is no long job $j \in \tau$ with $t_S \leq C_j(S) \leq t_\sigma$. In this case, let m_S be the number of machines that are idle in the interval $[t_S, t_\sigma)$ of schedule S, and let m_r be the number of short jobs from job system τ_r that start at time r in the same schedule S. Hence, we cover the following two possibilities:

i. $m_S \leq m_r$: We split any long job $j \in \tau_r$ into a long job j' with processing time $p_{j'} = p_j - \epsilon$ and an additional short job. Both jobs have the same new release date $r' = r + \epsilon$. Then we increase the release date of all short jobs from job system τ_r to the new release date r' as well. This process results in the new job system τ', the basic nondelay schedule S' with $t_{S'} = t_b(S')$ and the optimal schedule σ' with $t_{\sigma'} = t_b(\sigma')$. This transformation process is illustrated in Figures (3.5) and (3.6) of the basic schedule S and the optimal schedule σ, respectively. Assume that all new short jobs that are produced from the splitting process start at the same time t_S in the new schedule S'. Due to $m_S \leq m_r$, some of the short jobs from τ_r that execute in the interval $[r, r')$ of schedule S will execute on m_S machines during the interval $[t_S, t_S + \epsilon)$ in schedule S'. While the remaining jobs will move to the interval $[t_S + \epsilon, t_{S'})$. That is, we have $t_{S'} \geq t_S + \epsilon$. This results in $C_j(S) \leq C_{j'}(S')$, see Figure 3.5. Hence, we have

$$U_{[t_\sigma,T)}(S, \tau) \leq U_{[t_\sigma,T)}(S', \tau').$$

Note that the long job $j \in \tau$ starts at time r in the optimal schedule σ while the resulting new long job $j' \in \tau'$ starts at time r' in the new optimal schedule σ'. This results in $C_j(\sigma) = C_{j'}(\sigma')$ as $r' = r + \epsilon$. Further, $t_\sigma = t_{\sigma'}$ as the interval $[r, t_\sigma)$ in schedule σ is a full interval. However, this transformation process may decrease $t_b(\sigma_k)$, see Figure 3.6. There-

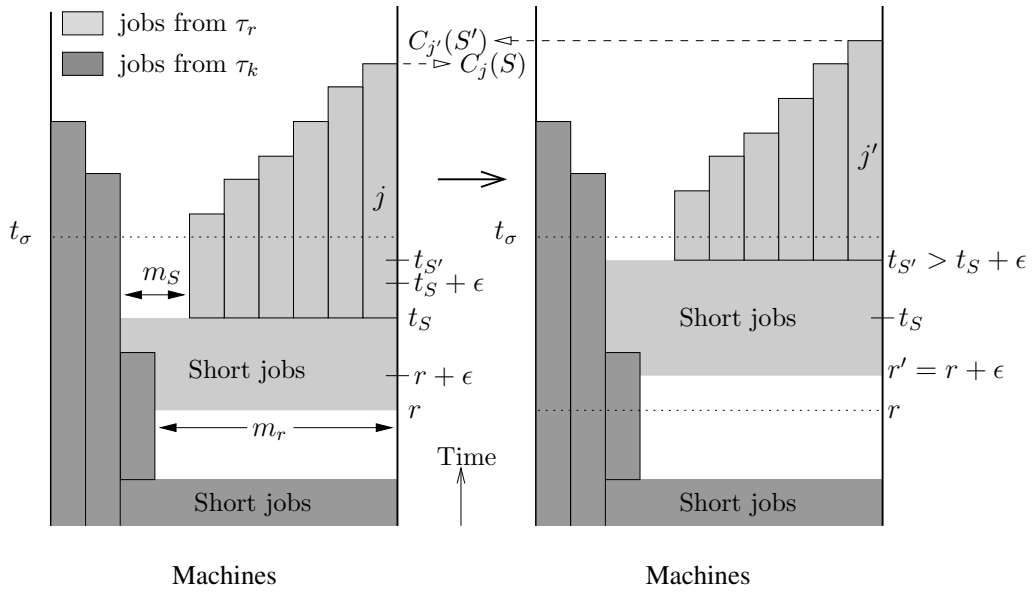

Figure 3.5: *The transformation process when $m_S \leq m_r$. Left: Basic schedule S of job system τ. Right: A resulting basic nondelay schedule S' of the new job system τ'.*

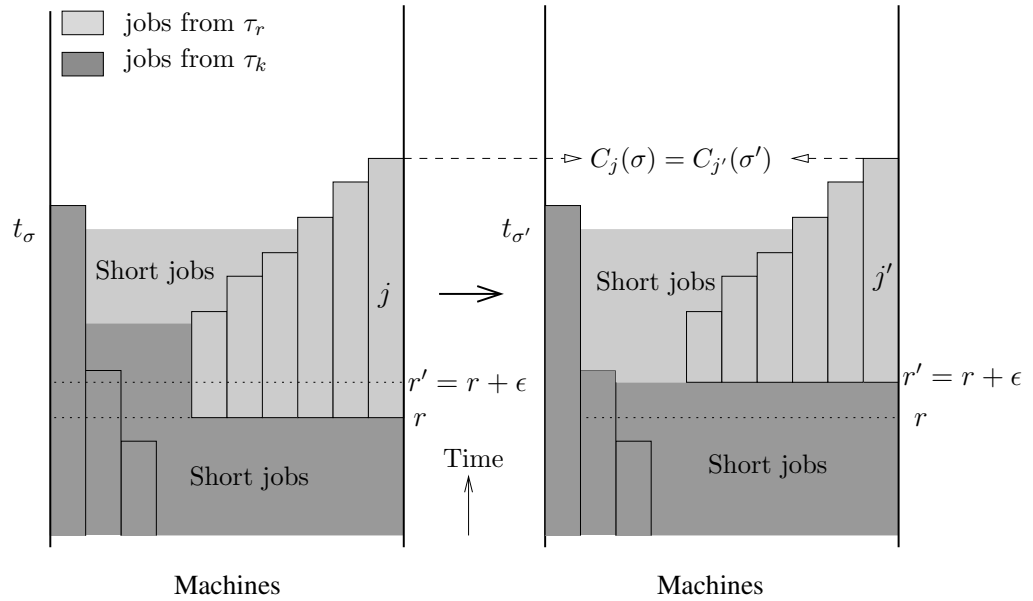

Figure 3.6: *Transformation process from the optimal schedule σ (left) of job system τ into the new optimal schedule σ' (right) of job system τ' when $m_S \leq m_r$.*

fore, we have

$$U_{[t_\sigma,T)}(\sigma,\tau) = U_{[t_\sigma,T)}(\sigma',\tau').$$

This results yield

$$\frac{D_{t_\sigma}(S',\tau')}{U^*_{[0,t_\sigma)}(\tau')} \geq \frac{D_{t_\sigma}(S,\tau)}{U^*_{[0,t_\sigma)}(\tau)}.$$

This process is repeated until $r' = t_b(\sigma_k)$ holds (Case 1).

ii. $m_S > m_r$: We combine every long job j from job system τ_r with a short job from τ_r with processing time ϵ. That is, this combination process produces a new long job j' with processing time $p_{j'} = p_j + \epsilon$. The resulting long jobs all have a new release date r'. Further, we decrease the release date of all other short jobs from job system τ_r to the release date r' as well. We choose this new release date r' such that $t_{S'} = t_S - \epsilon$ holds for the new basic job system τ' and its basic nondelay schedule S'.
Figures (3.7) and (3.8) illustrate this transformation process of schedules S and σ, respectively. Clearly, in the new basic schedule S', we have $C_j(S) = C_{j'}(S')$ as $t_{S'} = t_S - \epsilon$. This results in

$$U_{[t_\sigma,T)}(S',\tau') = U_{[t_\sigma,T)}(S,\tau).$$

Due to $m_S > m_r$, we have $r' < r - \epsilon$, see Figure 3.7. Again remember that in the optimal schedules σ and σ', each long job starts at its release date. Therefore, we have $C_{j'}(\sigma') < C_j(\sigma)$ as $r' < r - \epsilon$. Further, $t_{\sigma'} > t_\sigma$ holds for the new optimal schedule σ'. Consequently, we obtain

$$U_{[t_\sigma,T)}(\sigma',\tau') \leq U_{[t_\sigma,T)}(\sigma,\tau)$$

for the new optimal schedule σ'. Note that t_σ corresponds to the original optimal schedule σ. These results yield

$$\frac{D_{t_\sigma}(S',\tau')}{U^*_{[0,t_\sigma)}(\tau')} \geq \frac{D_{t_\sigma}(S,\tau)}{U^*_{[0,t_\sigma)}(\tau)}.$$

Note that, in schedule S', this process may result in a long job $i \in \tau$ with $t_{S'} \leq C_i(S') \leq t_\sigma$, see such job i in Figure 3.7. Together with the

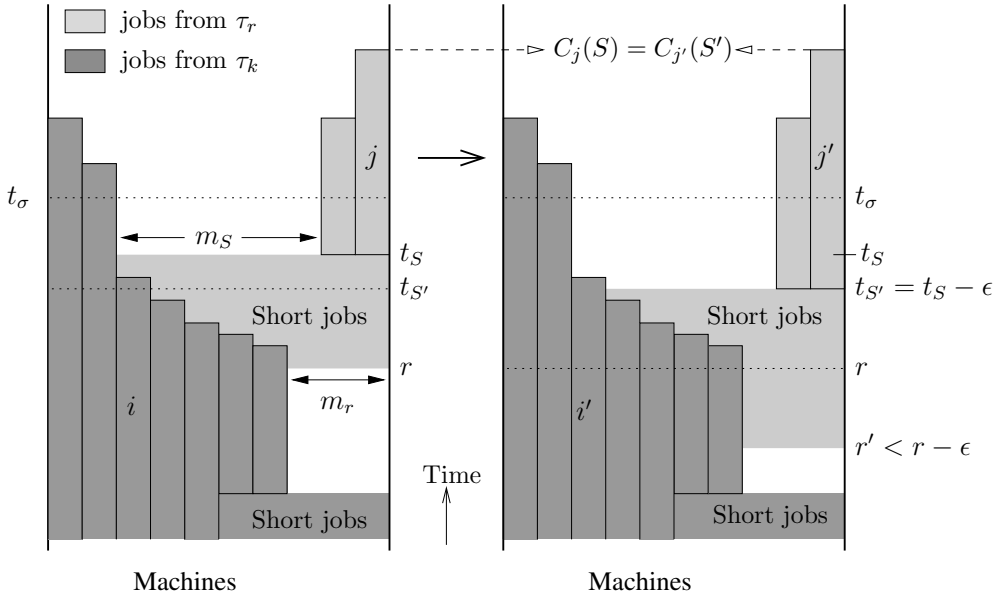

Figure 3.7: *An illustration of the transformation process when $m_S > m_r$. Left: Basic schedule S of job system τ. Right: A new basic nondelay schedule S' of job system τ'.*

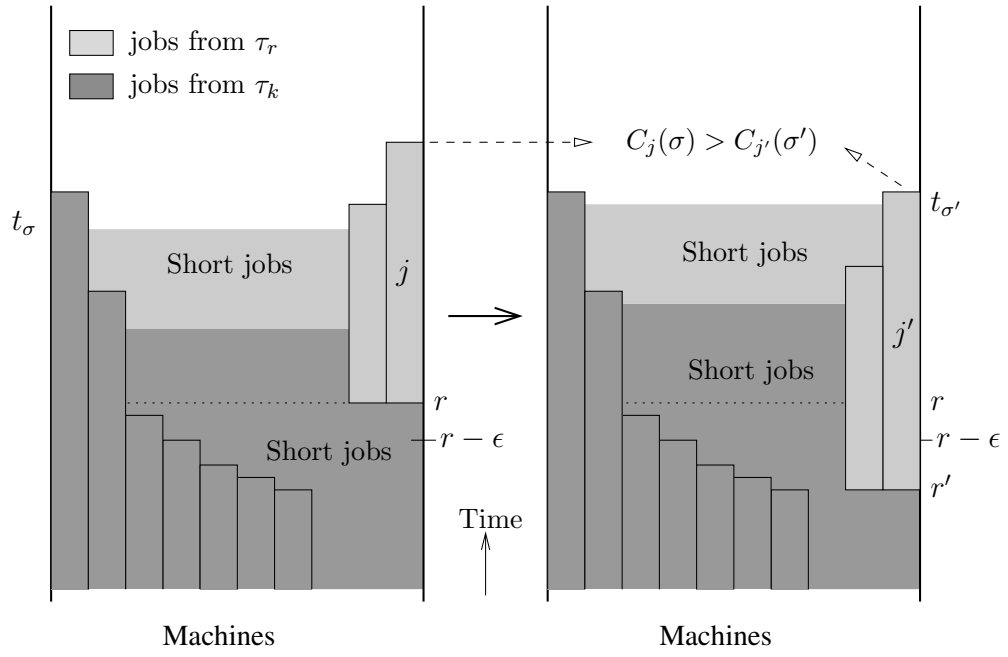

Figure 3.8: *Transformation process from the optimal schedule σ (left) of job system τ into the new optimal schedule σ' (right) of job system τ' in case $m_S > m_r$.*

transformation of Case 2c, this process is repeated until we do not have long job belongs to $\tau'_{r'}$ any more due to Case 2c or the basic job system τ' has only k different release dates.

■

With this result, we can finally determine an upper bound for $\frac{U^*_{[0,t)}(\tau)}{U_{[0,t)}(S,\tau)}$ if schedule S is a nondelay schedule.

Theorem 3.1 *For any job system τ and a nondelay schedule S for τ on m identical machines, the inequality $\frac{U^*_{[0,t)}(\tau)}{U_{[0,t)}(S,\tau)} \leq \frac{4}{3}$ holds for all $t \geq 0$. This bound is tight.*

Proof: Due to Corollary 2.2, it is sufficient to consider only basic job systems with $\epsilon \to 0$ and their basic nondelay schedules. Let τ be such a basic job system and S be its basic nondelay schedule.

Lemma 3.1 yields $D_t(S,\tau) = U^*_{[0,t)}(\tau) - U_{[0,t)}(S,\tau) \leq \frac{1}{4}U^*_{[0,t)}(\tau)$ for all $t \geq 0$. This immediately results in

$$\frac{U^*_{[0,t)}(\tau)}{U_{[0,t)}(S,\tau)} \leq \frac{4}{3}.$$

Finally assume $m > 1$ machines with m being even. Our job system τ contains $\frac{m}{2}$ jobs of processing time 2 and m jobs of processing time 1 all with release date 0. In schedule S, all m jobs with processing time 1 start concurrently at time 0 while the longer jobs all start at time 1. Clearly, S is a nondelay schedule and $U_{[0,2)}(S,\tau) = 1.5m$ holds.

In the optimal schedule σ, all $\frac{m}{2}$ jobs of processing time 2 and $\frac{m}{2}$ of the other jobs start concurrently at time 0 while the remaining jobs of processing time 1 start at time 1. This results in $U_{[0,2)}(\sigma,\tau) = U^*_{[0,2)}(\tau) = 2m$ and $\frac{U^*_{[0,2)}(\tau)}{U_{[0,2)}(S,\tau)} = \frac{4}{3}$. ■

Chapter 4

Online scheduling to minimize equal priority completion time

4.1 introduction

Often in practical scheduling systems, not all jobs are treated equally. Some jobs might be more important than others. Some operations inherently receive a higher priority as compared with other operations. Perhaps the simplest and most natural way to formalize setting with different priority levels is to assign weights to jobs and then consider some weighted function of the completion time or a related measure that one is interested in. For example, when the scheduling objective is to find schedules that minimize $\sum_{j\in\tau} w_j\, C_j$ (total weighted completion time) where C_j is the completion time of job j in the schedule and w_j is the weight of such job. We are primarily interested in non-preemptive schedules. The simplest variant of total weighted completion time minimization problems is when all jobs have the same release date and the goal is to schedule them on a single machine. For this case, the total weighted completion time problem can be solved optimally in polynomial time [76] using the *Weighted Shortest Processing Time first* (*WSPT*) rule (called also SMITH's rule). SMITH's rule schedules the jobs in non-increasing order of the ratio $\frac{w_j}{p_j}$. This rule minimizes the total weighted completion time only in the single machine case and unfortunately, it cannot be generalized to the parallel machines case even in the single release date case. Further, SMITH's rule cannot be extended to the multi-release date case even if we schedule

jobs on only a single machine. The offline version of the identical parallel machines problem is already $\mathcal{NP}$-hard [48]. In fact, our scenario has a more general and realistic setting of which the jobs have different release dates (online version). Clearly, this realistic restriction make the problem more harder.

As we have shown in Chapter 2, our second new criterion *equal priority completion time* $\sum_{j\in\tau} p_j\, C_j$ is adequate to quantitatively represent system utilization. Therefore, we devote this chapter to address the analysis of the nondelay schedules for our nonclairvoyant online scheduling model, with the goal of minimizing such criterion. As we pointed out previously, this criterion can be obtained from the total weighted completion time criterion by demanding that $w_j = p_j$ holds for all jobs in job system τ. Further, we will use the abbreviation C_{equ} to identify such criterion. Clearly, this weight selection guarantees that all jobs have the same SMITH's ratio. Consequently, it enable us to overcome the problem of a job's priority according to SMITH's rule. In this chapter, we will derive an upper bound for the competitive ratio of the *equal priority completion time*. Again, we need only to analyze basic job systems with $\epsilon \to 0$ that are described in Chapter2, their basic nondelay schedules, and their optimal schedules in order to determine a worst case deviation of equal priority completion time. Note that in any basic nondelay schedule, jobs can be arranged such that they are scheduled in non decreasing order of their release dates. However, this assumption is not obligatory for the optimal schedule because all long jobs start at their release dates as we have shown previously in Corollary 2.4.

The remainder of the chapter is organized as follows. Section 4.2 discusses some previous work related to the identical parallel-machine problems in which the objective is to minimize the weighted sum of the jobs completion times. In this section, we present results corresponding to both off-line and on-line versions of such problem. Moreover, we review some developments of the stochastic variant of such problem beside the deterministic models. In Section 4.3, we address the general problem in which the jobs have arbitrary weights, thus have different priorities. In this case, we will show that analyzing nondelay schedules yields an unbounded competitive ratio of the total weighted completion time criterion. As we will see later, the main result

of this chapter depends on the off-line result that has been presented by KAWAGUCHI and KYAN [44]. Therefore, we shall make use of such off-line result in our analysis of the online model. For the sake of completeness, we devote Section 4.4 to repeat KAWAGUCHI and KYAN's significant and tight bound to use it in the next section. In Section 4.5, we derive an upper bound of the competitive ratio $\frac{C_{equ}(S)}{C^*_{equ}(\tau)}$ for our online scenario where $C^*_{equ}(\tau)$ is the smallest attainable value of the equal priority completion time for job system τ. Finally, in Section 4.6, we give some remarks and observations about the possibility to apply another criterion related to flow time for the purpose of evaluating system utilization.

4.2 Related Results

Scheduling to minimize total weighted completion time is one of the best studied class of problems in scheduling theory. Most variants of scheduling to minimize such criterion are strongly $\mathcal{NP}$-hard including preemptive problems [51]. In the last few years, considerable progress has been made in understanding the approximability of many of these $\mathcal{NP}$-hard problems. Constant and logarithmic ratio approximations were found for several variants. See [1, 29] for more details on the history of these developments. In this section, we present only some of those previous results that are related to the total weighted completion time criterion where we restrict our review on the models with identical parallel machine environment.

Starting with the off-line variant, the total weighted completion time minimization problem with identical parallel machines and single release date, $P||\sum_j w_j C_j$, has been determined to be $\mathcal{NP}$-hard in the strong sense [20]. However, KAWAGUCHI and KYAN [44] establish that the *Weighted Shortest Processing Time* approach achieves an $\frac{1}{2}(1+\sqrt{2}) \approx 1.21$ approximation ratio for such problem. The authors analyzed a list scheduling algorithm in which jobs are ordered according to non-increasing ratios of weight to processing time $\frac{w_j}{p_j}$. Further, they proved the tightness of their bound. This result was the best for a long time. ALON *et al.* [4] gave a polynomial-time approximation scheme (PTAS) for the problem of minimizing $\sum_{i=1}^{m} M_i^2$ where M_i denotes the

completion time of machine i. Recently, SKUTELLA and WOEGINGER [74] realized that ALON's result implies also a PTAS for the problem of minimizing the total weighted completion time criterion if all job ratios $\frac{w_j}{p_j}$ are equal. In a first step, the authors generalized this to a PTAS for $\frac{w_j}{p_j}$ ratios within a constant range. In a second step, they constructed their PTAS for the general weight problem $P||\sum_j w_j C_j$. Their result improves upon the previous best known bound by KAWAGUCHI and KYAN [44]. In fact, this result constitutes the first known polynomial-time approximation scheme for a strongly $\mathcal{NP}$-hard problem with the Min-Sum objective. Their algorithm is based on ratio-partitioning. The basic idea was to partition the jobs into groups according to their $\frac{w_j}{p_j}$ ratios such that the ratios of all jobs in one group are within a constant range. Then, using the first step, compute near optimum schedules for each group separately. Finally, the schedules for different groups could be concatenated together on the same machine according to SMITH's rule because there are no different release dates. However, this technique cannot be easily generalized to the scheduling problems involving multiple release dates, *i.e.* $P|r_j|\sum_j w_j C_j$. For this multi-release date case, that is, the problem is still off-line as release dates are known in advance, the time partitioning technique was recalled by several researchers [1] and it was proven powerful enough to yield PTASs in the presence of these release dates both with and without preemptions allowed. Several other algorithms use a linear programming relaxation to obtain constant approximation factors for the off-line problem $P|r_j|\sum_j w_j C_j$. PHILLIPS *et al.* [62] gave the first such algorithms, a $(24+\epsilon)$-approximation algorithm. This result has been greatly improved to $4+\epsilon$ and $4-\frac{1}{m}$ by HALL *et al.* in [28] and its journal version by HALL *et al.* in [29]. Subsequently, there has been an explosion of research for such problem with successively smaller constant ratios algorithms. Most of theses algorithms follow the general and successful relaxation approach: first an optimal solution to a relaxation of the original problem is polynomially obtained, then this solution is rounded, either to order the jobs in time or to assign the jobs to machines, to obtain a near-optimum optimum solution of the original problem. The large body of the algorithms within this approach can be classified according to the type of relaxation. Types of relaxations include preemptive schedule relaxation and linear programming

relaxation. CHAKRABARTI *et al.* [13] obtained a 3.5-approximation algorithm by using a conversion technique from preemptive to non-preemptive schedules. By using a general on-line framework [13], one can derive an algorithm with an approximation ratio of $2.89 + \epsilon$. More recently, the best current approximation factor for this problem has been derived by SCHULZ and SKUTELLA [69]. The authors provided a randomized algorithm that has running time $O(n \log n)$ while the running time of the best previous algorithm was $\Theta((m+1)^{poly(1/\epsilon)}n + n \log n)$ by AFRATI *et al.* [1]. Their randomized algorithm has a performance guarantee of 2. Interestingly, They stated that their algorithm can be applied to the on-line setting in which jobs arrive over time as well, with no difference in such performance guarantee. Their main idea was to assign jobs randomly to machines with probabilities derived from an optimal solution to a linear programming relaxation in time-indexed variables. In addition, it is worth to mention that their work includes new results for models with more general machine environment both with and without release dates, *i.e.* $R||\sum_j w_j C_j$ and $R|r_j|\sum_j w_j C_j$.

So far, all results presented above are treating the off-line version for the identical parallel machines scheduling problem with the total weighted completion time objective. Now, we turn our attention to the on-line version of such problem which is more closely to our scenario, that is, the situations in which the existence of a job is unknown until the arrival time. For the deterministic work, there are only two recent papers that addressed this kind of online problem. However, the authors in both papers restrict their considerations with the assumption that all job characteristics are known as soon as a job arrives. This assumption is slightly different from our scenario which impose that the algorithm learns the processing time of a job only once it is completed. The first paper by HALL *et al.* [29] presented a deterministic online algorithm which has the competitive ratio $4 + \epsilon$. This algorithm was given as part of a more general on-line framework. The authors introduced a general technique producing on-line algorithms that yield constant performance guarantees for a variety of scheduling models in which the objective is to minimize the weighted sum of the job completion times. Further, their paper gives comprehensive reviews of the development of both off-line and on-line algorithms for the total weighted completion time minimization

problem with various scheduling environments. In the second paper, the currently best competitive ratio of 3.28 is achieved by the *Shifted WSPT* algorithm of MEGOW and SCHULZ [54]. The authors used the straightforward extension of SMITH's rule to the parallel machines with the idea of delaying the release dates. They modify the release date of each job such that it is equal to a certain fraction of its processing time. The tightness of their bound has not been proven, but they conjecture that the remaining gap is at most 0.5.

In a special case $w_j = 1$ for all jobs that arrive over time, PHILLIPS *et al.* [64] presented online algorithm for $P|r_j|\sum_j C_j$, which converts a preemptive schedule into a non-preemptive one. This algorithm achieves a conversion factor of $3 - \frac{1}{m}$. Subsequently, CHEKURI *et al.* [14] showed that by sequencing jobs nonpreemptively in the order of their completion times in the optimal preemptive schedule on a single machine of speed m times as fast as that of any one of the m identical machines, one can obtain a $(3-\frac{1}{m})$-competitive algorithm for the online variant of the problem $P|r_j|\sum_j C_j$. More recently, the same online problem is addressed and this result is improved considerably by LU et al. [52]. The authors gave a 2α-competitive online algorithm, where α denotes the performance ratio of the *Shortest Remaining Processing Time* first rule for the preemptive relaxation of the problem. Moreover, this rule is known to achieve a worst-case performance ratio of 2, as was shown by PHILLIPS *et al.* [64].

Let us eventually present some stochastic work for the online problem $P|r_j|\sum_j w_j C_j$. The only online characteristic of the model of stochastic scheduling is the fact that the actual job processing times become known only upon completion. However, their respective probability distributions are assumed to be given beforehand. Further, the aim is to find a scheduling policy that minimizes the expected total weighted completion time. By using an approach based upon the solution of linear programming relaxations, MÖHRING *et al.* [57] have derived an LP-based priority policy with a performance guarantee of $3 - \frac{1}{m} + \max\left\{1, \frac{m-1}{m}\Delta\right\}$ for such problem, where Δ is an upper bound on the squared coefficients of variation of the occurring probability distributions. Afterwards, CHAKRABARTI *et al.* [13] presented a randomized online algorithm with a performance guarantee of $2.89 + \epsilon$. Recently, this result has been improved to

2 by a $\Theta(n \log n)$ randomized algorithm that is capable of working in an on-line context in which jobs are randomly assigned to machines (*i.e.* jobs arrive over time), as was shown by SCHULZ and SKUTELLA [69]. Concurrently with writing this thesis, the same online model $P|r_j|\mathbb{E}\left[\sum_j w_j C_j\right]$ has been addressed by MEGOW *et al.* [55] who obtained a performance bound strictly less than $\left(\frac{5+\sqrt{5}}{2} - \frac{1}{2m}\right)$ for a specific distribution called NBUE. This result improved upon the previously best known performance guarantee of $4 - \frac{1}{m}$ for the same NBUE distributions, which was derived for an LP-based list scheduling policy [57]. Further, the authors showed that their improved bound holds even though we apply a restricted policy that first has to assign jobs to machines on-line, without knowledge of the jobs to come. However, they assumed that jobs appear one by one. Finally, the best known result of the off-line version of the stochastic problem $P||\mathbb{E}\left[\sum_j w_j C_j\right]$ has been obtained very recently by SOUZA and STEGER [77]. The authors provided a general bound on the expected competitive ratio for list scheduling algorithms. Their bound depends on the probability of any pair of jobs being in the wrong order in the list of an arbitrary list scheduling algorithm, compared to an optimum list OPT. For a special case, they show that WSEPT (Weighted Shortest Expected Processing Time) algorithm achieves $\mathbb{E}\left[\frac{WSEPT}{OPT}\right] \leq 3 - \frac{1}{m}$ for exponential distributed processing times. Moreover, they provided empirical simulations which demonstrate the tightness of this bound.

4.3 Jobs with Arbitrary Priority

In this section, we address the general case of the total weighted completion time minimization problem, that is, when the jobs have arbitrary weights. We will show that there is no upper bound for the competitive factor $\frac{C(S)}{C^*(\tau)}$ of any nondelay schedule for such online problem in its full generality of the job weights.

Lemma 4.1 *The competitive factor for the general online total weighted completion time scheduling problem for m identical machines is unbounded.*

Proof: There are job systems τ with $|\tau| = n \leq m$ such that $\frac{C(S)}{C^*(\tau)} = \Theta(k)$ for any $k > 0$ unless all jobs of job system τ are executed concurrently at a time instant in schedule S. This statement is proved by induction on the number n of jobs in any job system τ. For $n = 1$, the starting of the single job clearly cannot be postponed forever to prevent. Therefore, we assume that the statement is true for some number of jobs $n - 1 < m$. Hence, there are a job system τ' with $|\tau'| = n - 1$ and $q = \max_{j \in \tau'} \left(\frac{w_j}{p_j}\right)$, a schedule S', and a time instant t such that all $n - 1$ jobs of job system τ' execute concurrently at such time t in schedule S'. Note that, $w_j \leq (q\ p_j)$ holds for each job $j \in \tau'$. In our online model we are free to increase the processing times of the jobs that are executing at time t. Therefore, we require the processing time p_j to be sufficiently large such that $C_j(S') = t\ (k+1)$ for all jobs $j \in \tau'$. Hence, we have

$$\begin{aligned} C^*(\tau') &= \sum_{j \in \tau'} w_j\ C_j^* \\ &\leq \sum_{j \in \tau'} w_j\ C_j(S') \leq \sum_{j \in \tau'} q\ p_j\ C_j(S') \\ &\leq q \sum_{j \in \tau'} \Big(C_j(S')\Big)^2 = (n-1)\ q\ \Big(t\ (k+1)\Big)^2 \end{aligned}$$

Next, let us presume that an additional job j_n is released with $r_{j_n} = p_{j_n} = t$ and $w_{j_n} = (n\ q\ t\ k^2)$. This additive process generates a new job system τ with $\tau = \tau' \cup \{j_n\}$ and a new schedule S based on schedule S'. Assume that the new job j_n is not started before time $t\ (k+1)$ in schedule S to prevent n jobs being executed concurrently in schedule S at any time instant. Consequently, we have $C_{j_n}(S) \geq t(k+2)$ and $C_{j_n}^* = 2t$. This results

in

$$\begin{aligned}
\frac{C(S)}{C^*(\tau)} &= \frac{C(S') + \left(w_{j_n}\, C_{j_n}(S)\right)}{C^*(\tau') + \left(w_{j_n}\, C^*_{j_n}\right)} \\
&\geq \frac{C(S') + n\, q\, k^2\, \left(t^2(k+2)\right)}{C^*(\tau') + n\, q\, k^2\, (2t^2)} \\
&\geq \frac{C^*(\tau') + n\, q\, k^2\, \left(t^2(k+2)\right)}{C^*(\tau') + n\, q\, k^2\, (2t^2)} \\
&\geq \frac{n\, q\, \left(t\, (k+1)\right)^2 + n\, q\, k^2\, \left(t^2\, (k+2)\right)}{n\, q\, \left(t\, (k+1)\right)^2 + n\, q\, k^2\, (2t^2)} \\
&= \frac{k^3 + 3k^2 + 2k + 1}{3k^2 + 2k + 1} = \Theta(k).
\end{aligned}$$

Next, we assume a job system τ' with $|\tau'| = m$ and $q = \max_{j \in \tau'} \left(\frac{w_j}{p_j}\right)$, a schedule S', and a time instant t such that all m jobs of job system τ' execute concurrently at time t in schedule S'. Again we require the processing time p_j to be sufficiently large such that $C_j(S') = t\ (k+1)$ for all jobs $j \in \tau'$ and release an additional job j_n with $r_{j_n} = p_{j_n} = t$ and $w_{j_n} = (m\, q\, t\, k^2)$ to generate a new job system $\tau = \tau' \cup \{j_n\}$ and a new schedule S. Note that, schedules S' and S are identical for all jobs of the job system τ'. As in the previous case we have

$$C^*(\tau') \leq C(S') = \sum_{j \in \tau'} w_j\, C_j(S') \leq m\, q\, \left(t(k+1)\right)^2$$

In schedule S, the new job j_n cannot be started before time $t\ (k+1)$. However, in the optimal schedule, job j_n is started at its release time t while the start of one job from job system τ', say job i, is delayed until time $t + p_{j_n} = 2t$. That is, the delaying time of such job is at most $2t$. Therefore, we obtain

$$\begin{aligned}
C^*(\tau) &\leq w_{j_n}\, C^*_{j_n} + C^*(\tau') + w_i \cdot 2t \\
&\leq (m\, q\, t\, k^2) \cdot 2t + C^*(\tau') + (q\, p_i) \cdot 2t \\
&\leq m\, q\, k^2\, (2t^2) + C^*(\tau') + 2q\, t^2(k+1).
\end{aligned}$$

Clearly, all jobs from τ' are scheduled in schedules S' and S in the same fashion. This results in

$$\begin{aligned}
\frac{C(S)}{C^*(\tau)} &\geq \frac{m\,q\,k^2\left(t^2(k+2)\right)+C(S')}{m\,q\,k^2\,(2t^2)+C^*(\tau')+2q\,t^2(k+1)} \\
&\geq \frac{m\,q\,k^2\left(t^2(k+2)\right)+C(S')}{m\,q\,k^2\,(2t^2)+C(S')+2q\,t^2(k+1)} \\
&= \frac{m\,q\,k^2\left(t^2(k+2)\right)+m\,q\left(t\,(k+1)\right)^2}{m\,q\,k^2\,(2t^2)+m\,q\left(t\,(k+1)\right)^2+2q\,t^2\,(k+1)} \\
&= \frac{m\left(k^3+3k^2+2k+1\right)}{m\left(3k^2+2k+1\right)+2k+2} \\
&\geq \frac{k^3+3k^2+2k+1}{3k^2+4k+3}=\Theta(k).
\end{aligned}$$

This concludes the proof. ∎

After we have shown that there is no constant competitive factor for the general online problem we will address in the remainder of this chapter a special variant of such online problem where all jobs have equal priority, that is, when $w_j = p_j$ holds for all jobs $j \in \tau$. In other words, we are going to apply our second new criterion *equal priority completion time* C_{equ} which has been already introduced formally in Section 2.2. In the next section, we reobtain the competitive factor of the off-line version of our problem which was provided already by KAWAGUCHI and KYAN [44].

4.4 The upper bound of the off-line problem

To our knowledge our online scenario of minimizing equal priority completion time has not been addressed earlier. However, from the results of KAWAGUCHI and KYAN [44] it follows immediately that all list schedules with equal priority jobs are $\left(\frac{\sqrt{2}+1}{2}\right)$-competitive if all jobs are released at time 0. Further, it can easily be seen that this

factor is tight. In Section 4.5, we will use this competitive factor of the off-line problem to obtain our bound of the online version of such problem. For the sake for completeness, we give the result from KAWAGUCHI and KYAN [44] and briefly repeat the proof using the notations and corollaries of this thesis. To do so, we start with the following corollary which holds for any job system where all jobs have equal priorities.

Corollary 4.1 *Let τ be a job system with equal priority jobs. Then any legal schedule S with $C_{max}(S) = \sum_{j\in\tau} p_j$ is optimal for any order of the jobs in S and there is*

$$C(S) = C^*(\tau) = \frac{1}{2}\left(\sum_{j\in\tau} p_j\right)^2 + \frac{1}{2}\sum_{j\in\tau} p_j^2.$$

Proof: Note that Smith's rule [76] guarantees that the job order of schedule S does not affect the total weighted completion time cost. Hence, any schedule S with no intermediate idle periods is optimal.

Now, we use induction to proof such bound. As the last job in schedule S always completes at time $\sum_{j\in\tau} p_j$ the bound is clearly true for any job system with $|\tau| = 1$. Next assume that the bound holds for all job systems τ' with $|\tau'| = n - 1$ and any schedule S' for such job system τ' with $C_{max}(S) = \sum_{j\in\tau} p_j$. Adding a new job j_n to the job system τ' and starting this job time immediately after the completion of the last job in schedule S' produces a new job system τ with $|\tau| = n$ and a new schedule S with

$$\begin{aligned}
C(S) &= C(S') + w_{j_n} p_{j_n} + w_{j_n} \sum_{j\in\tau} p_j \\
&= C(S') + p_{j_n}^2 + p_{j_n} \sum_{j\in\tau} p_j \\
&= \frac{1}{2}\left(\sum_{j\in\tau'} p_j\right)^2 + \frac{1}{2}\sum_{j\in\tau'} p_j^2 + p_{j_n}^2 + p_{j_n} \sum_{j\in\tau} p_j \\
&= \frac{1}{2}\left(\sum_{j\in\tau} p_j\right)^2 + \frac{1}{2}\sum_{j\in\tau} p_j^2 \,.
\end{aligned}$$

■

Note that all long jobs in a basic job system can have different processing times. In order to simplify our further analysis we define a rectangular job system $\bar{\tau}$:

Definition 4.1 *A job system $\bar{\tau}$ with equal priority jobs is called* **rectangular** *if it consists of $m - m_t$ jobs with processing time $p_b \geq 0$ and m_t jobs with processing time $p_t > p_b$. All jobs have the same release date r.*

Clearly, $\sum_{j\in\bar{\tau}} p_j = m_t p_t + (m - m_t)p_b$ holds for any rectangular job system $\bar{\tau}$. Therefore, the following equation holds for a rectangular job system $\bar{\tau}$ as well:

$$\begin{aligned} C^*(\bar{\tau}) &= r\sum_{j\in\bar{\tau}} p_j + \sum_{j\in\bar{\tau}} p_j^2 \\ &= r\sum_{j\in\bar{\tau}} p_j + m_t\, p_t^2 + (m - m_t)\, p_b^2 \end{aligned}$$

Further, we want to mention a simple convexity relation:

$$\begin{aligned} \sum_{j\in\tau} p_j^2 &= \sum_{j\in\tau}\left(p_j - \frac{1}{|\tau|}\sum_{j\in\tau} p_j + \frac{1}{|\tau|}\sum_{j\in\tau} p_j\right)^2 \\ &= \sum_{j\in\tau}\left(\frac{1}{|\tau|}\sum_{j\in\tau} p_j\right)^2 + \sum_{j\in\tau}\left[2\left(p_j - \frac{1}{|\tau|}\sum_{j\in\tau} p_j\right)\frac{1}{|\tau|}\sum_{j\in\tau} p_j\right] + \sum_{j\in\tau}\left(p_j - \frac{1}{|\tau|}\sum_{j\in\tau} p_j\right)^2 \\ &\geq \frac{1}{|\tau|}\left(\sum_{j\in\tau} p_j\right)^2 + \sum_{j\in\tau}\left(p_j - \frac{1}{|\tau|}\sum_{j\in\tau} p_j\right)^2 \\ &\geq \frac{1}{|\tau|}\left(\sum_{j\in\tau} p_j\right)^2 \end{aligned} \tag{4.1}$$

It is easy to see that any upper bound of $\frac{C(S)}{C^*(\tau)}$ that is valid for $2m$ machines also holds for m machines. Therefore, we base our worst case analysis on systems with a very large number of machines and assume that all relevant values for m_t with $0 < m_t \leq m$ are integer.

Next, we consider a simple job system τ with $n \leq m$ jobs such that all jobs have the same release date. For such a job system, we can find a rectangular job system that has the same optimal total weighted completion time and the same sum of processing times, see Figure 4.1.

Corollary 4.2 *Assume a job system τ consisting of $n \leq m$ long jobs with equal priorities and the same release date r. For each value m_t with $0 < m_t \leq \left(\left(\sum_{j\in\tau} p_j\right)^2 / \sum_{j\in\tau} p_j^2\right)$ there is a*

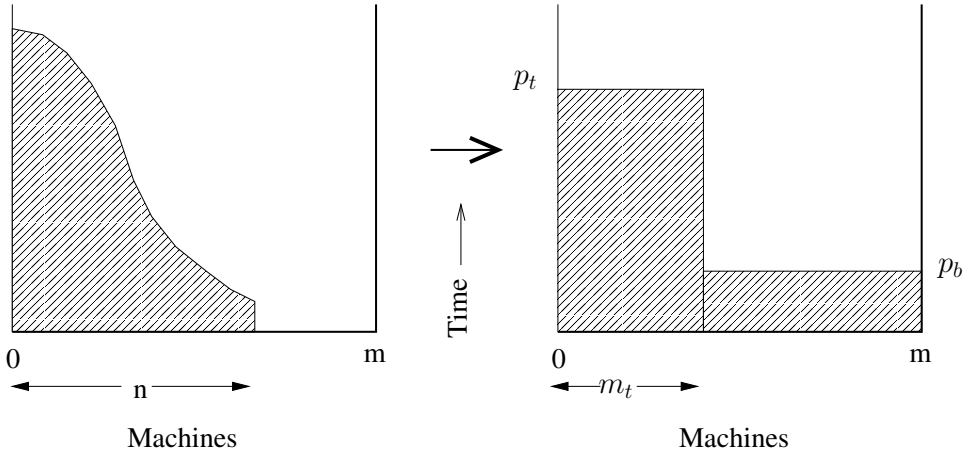

Figure 4.1: *The transformation of τ to rectangular job system $\bar{\tau}(m_t)$.*

rectangular job system $\bar{\tau}(m_t)$ such that

$$\sum_{j\in\{\bar{\tau}(m_t)\}} p_j = \sum_{j\in\tau} p_j \qquad \textit{and} \qquad C^*(\tau) = C^*\Big(\bar{\tau}(m_t)\Big) \qquad \textit{hold.}$$

Proof: Due to Inequality 4.1 and the inequality $|\tau| = n \leq m$, we have $\sum_{j\in\tau} p_j^2 \geq \frac{1}{m}\Big(\sum_{j\in\tau} p_j\Big)^2$.

Then we use the following values of p_t and p_b which always exist:

$$p_t = \frac{\sum_{j\in\tau} p_j}{m} + \sqrt{\frac{m-m_t}{m\,m_t}\left(\sum_{j\in\tau} p_j^2 - \frac{\Big(\sum_{j\in\tau} p_j\Big)^2}{m}\right)}$$

$$p_b = \frac{\sum_{j\in\tau} p_j}{m} - \sqrt{\frac{m_t}{m\,(m-m_t)}\left(\sum_{j\in\tau} p_j^2 - \frac{\Big(\sum_{j\in\tau} p_j\Big)^2}{m}\right)}$$

These values satisfy the following equation:

$$\sum_{j\in\{\bar{\tau}(m_t)\}} p_j = m_t\,p_t + (m-m_t)\,p_b = \sum_{j\in\tau} p_j$$

By using this result and the values of p_t and p_b we can obtain

$$\begin{aligned} C^*\Big(\bar{\tau}(m_t)\Big) &= m_t\, p_t^2 + (m - m_t)\, p_b^2 + r \sum_{j \in \{\bar{\tau}(m_t)\}} p_j \\ &= \sum_{j \in \tau} p_j^2 + r \sum_{j \in \tau} p_j \\ &= C^*(\tau) \end{aligned}$$

Finally, we show that $p_b \geq 0$ holds for $0 < m_t \leq \frac{\left(\sum_{j \in \tau} p_j\right)^2}{\sum_{j \in \tau} p_j^2}$ as we have:

$$\begin{aligned} p_b \geq 0 \quad &\Longleftrightarrow \quad \frac{\left(\sum_{j \in \tau} p_j\right)^2}{m^2} \geq \frac{m_t}{m\,(m - m_t)} \left(\sum_{j \in \tau} p_j^2 - \frac{\left(\sum_{j \in \tau} p_j\right)^2}{m} \right) \\ &\Longleftrightarrow \quad m \Big(\sum_{j \in \tau} p_j \Big)^2 \geq m\, m_t \sum_{j \in \tau} p_j^2 \\ &\Longleftrightarrow \quad \frac{\left(\sum_{j \in \tau} p_j\right)^2}{\sum_{j \in \tau} p_j^2} \geq m_t. \end{aligned}$$

■

Note that there is a specific solution with $p_t = \frac{\sum_{j \in \tau} p_j^2}{\sum_{j \in \tau} p_j}$, $p_b = 0$ and $m_t = \frac{\left(\sum_{j \in \tau} p_j\right)^2}{\sum_{j \in \tau} p_j^2}$.

Also assume that the processing time of the shortest job of the rectangular job system $\bar{\tau}$ with processing time $p_{\min} \geq 0$ is increased by a very small amount x. In order to determine the influence of this modification on the ratio $\frac{\left(\sum_{j \in \tau} p_j\right)^2}{\sum_{j \in \tau} p_j^2}$ and consequently on the value m_t, we determine

$$\begin{aligned} \lim_{x \longrightarrow 0} \frac{\partial}{\partial x} \left(\frac{\left(x + \sum_{j \in \tau} p_j\right)^2}{\sum_{j \in \tau \setminus j_{min}} p_j^2 + (p_{min} + x)^2} \right) &= \lim_{x \longrightarrow 0} \frac{\partial}{\partial x} \left(\frac{\left(x + \sum_{j \in \tau} p_j\right)^2}{x^2 + 2x p_{\min} + \sum_{j \in \tau} p_j^2} \right) \\ &= \frac{2\Big(\sum_{j \in \tau} p_j\Big)\Big(\sum_{j \in \tau} p_j^2 - p_{\min} \sum_{j \in \tau} p_j\Big)}{\Big(\sum_{j \in \tau} p_j^2\Big)^2} \\ &\geq 0\,. \end{aligned} \tag{4.2}$$

Therefore, such a modification cannot decrease the ratio $\frac{\left(\sum_{j \in \tau} p_j\right)^2}{\sum_{j \in \tau} p_j^2}$.

Next, we compare two rectangular job systems $\bar{\tau}$ and $\bar{\tau}'$ with the same value m_t, the same release date, $p_t > p'_t$, and $\sum_{j\in\bar{\tau}} p_j = \sum_{j\in\bar{\tau}'} p_j$, that is $p'_b = p_b + (p_t - p'_t)\frac{m_t}{m-m_t}$. Then we have

$$\begin{aligned} C^*(\bar{\tau}) - C^*(\bar{\tau}') &= m_t\, p_t^2 + (m-m_t)\, p_b^2 - m_t\, {p'}_t^2 - (m-m_t)\, {p'}_b^2 \\ &= m_t \left(p_t^2 - {p'}_t^2\right) + (m-m_t)\left[p_b^2 - \left(p_b + (p_t - {p'}_t)\frac{m_t}{m-m_t}\right)^2\right] \\ &= m_t \left(p_t^2 - {p'}_t^2\right) - 2m_t\, p_b(p_t - {p'}_t) - \left(p_t - p'_t\right)^2 \frac{{m_t}^2}{m-m_t} \\ &= m_t \left(p_t - {p'}_t\right)\left(p_t + {p'}_t - 2p_b - (p_t - {p'}_t)\frac{m_t}{m-m_t}\right) \\ &= m_t\, (p_t - p'_t)\Big(p_t + p'_t - p_b - p'_b\Big) > 0 \end{aligned} \tag{4.3}$$

For a given (integer) value m_t, let us denote the set of the jobs with the m_t largest processing times by τ_t. Due to the convexity relation (4.1) and the relation between two rectangular systems we obtain

$$m_t\, p_t \geq \sum_{j\in\tau_t} p_j \quad \text{and} \quad (m-m_t)p_b \leq \sum_{j\in\tau\setminus\tau_t} p_j. \tag{4.4}$$

We consider the optimal schedule of a job system $\tau = \tau_{long} \cup \tau_{short}$ which contains $n \leq m$ long jobs and additional short jobs. The next corollary shows that for the purpose of worst case analysis we do not need to use all possible job systems τ_{long} but that we can restrict ourselves to rectangular job systems instead, see Figure 4.2.

Corollary 4.3 *Job system $\tau = \tau_{long} \cup \tau_{short}$ consists of $n \leq m$ long jobs (τ_{long}) and additional short jobs (τ_{short}) with all jobs $j \in \tau$ having the same release date r. There is another job system $\tilde{\tau} = \bar{\tau}(m_t) \cup \tau_{short}$ consisting of the same set τ_{short} and a rectangular job system $\bar{\tau}(m_t)$ corresponding to job system τ_{long} with the processing times p_t and p_b being chosen according to Corollary 4.2. Then $C^*(\tilde{\tau}) \leq C^*(\tau)$ holds.*

Proof: Without restriction of generality we set $r = 0$. Let t_σ be the end of the full interval $[0, t_\sigma)$ in the optimal schedule σ of job system τ. We define $\tau_l = \{j \in \tau_{long} | p_j \geq t_\sigma\}$ and $m_l = |\tau_l|$. Clearly, all jobs from τ_{short} are scheduled on $m - m_l$ machines in schedule σ and we have $(m-m_l)\, t_\sigma = \sum_{j\in\tau_{short}} p_j + \sum_{j\in\tau_{long}\setminus\tau_l} p_j$.

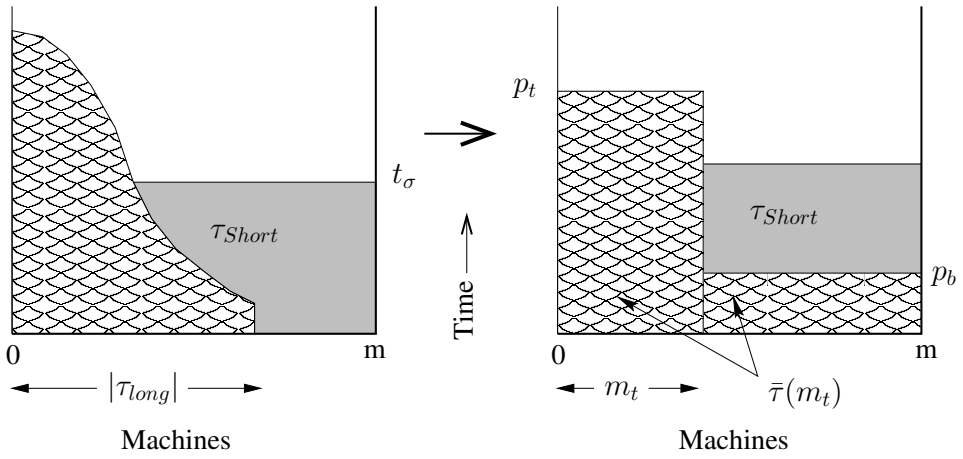

Figure 4.2: *The transformation of τ with a large number of short jobs.*

Assume that all jobs from τ_{long} are assigned to machines in decreasing order of their processing times, that is, all jobs from set τ_l are executed on machines 1 to m_l. Further, we set $m_t = \frac{\left(\sum_{j\in\tau_{long}} p_j\right)^2}{\sum_{j\in\tau_{long}} p_j^2}$. Remember that in this case $p_b = 0$ and $p_t = \frac{\sum_{j\in\tau} p_j^2}{\sum_{j\in\tau} p_j}$. Next, we consider the following two cases:

1. $(m - m_t)\, t_\sigma \geq \sum_{j\in\tau_{short}} p_j$. In the optimal schedule $\tilde{\sigma}$ of job system $\tilde{\tau}$, all jobs from $\bar{\tau}(m_t)$ are started at time 0 on the first m_t machines while all jobs from the set τ_{short} are executed on the other $m - m_t$ machines. Further, in the optimal schedule σ of τ, less resources are used for executing jobs from the set τ_{short} on machines $m_l + 1$ to m_t than resources are needed for processing jobs from the set τ_{long} on machines $m_t + 1$ to m. Hence, there is a bijective mapping $\left\{\mathcal{G} : \tau_{short} \rightarrow \tau_{short}\right\}$ such that $C_j(\sigma) \geq C_{\mathcal{G}(j)}(\tilde{\sigma})$ holds for all jobs $j \in \{\tau_{short}\}$. That is, the completion time of each short job in the schedule $\tilde{\sigma}$ cannot be larger than the completion time of its corresponding job in the schedule σ. Therefore, the total weighted sum of completion times of all jobs from τ_{short} is not more in schedule $\tilde{\sigma}$ than in schedule σ. This results in $C^*(\tilde{\tau}) = C(\tilde{\sigma}) \leq C(\sigma) = C^*(\tau)$.

2. $(m - m_t)t_\sigma < \sum_{j\in\tau_{short}} p_j$. Due to $\epsilon \to 0$ we obtain with Corollary 4.1:

$$C^*(\tau) = \sum_{j\in\tau_l} p_j^2 + \frac{1}{2}\left(\sum_{j\in\tau_{long}\setminus\tau_l} p_j^2 + (m - m_l)\, t_\sigma^2\right)$$

Now, let us define another job system τ' consisting of job system τ_l and $m - m_l$ jobs with release date 0 and processing time t_σ. Note that we have $\frac{\left(\sum_{j\in\tau_{long}} p_j\right)^2}{\sum_{j\in\{\tau_{long}\}} p_j^2} \leq \frac{\left(\sum_{j\in\tau'} p_j\right)^2}{\sum_{j\in\tau'} p_j^2}$ due to Inequality 4.2. Then job system $\bar{\tau}'(m_t)$ is a rectangular job system corresponding to τ' with p'_t and p'_b being chosen according to Corollary 4.2. This leads to

$$\begin{aligned} C^*(\tau) &= \frac{1}{2}\sum_{j\in\tau_{long}} p_j^2 + \frac{1}{2}\left(\sum_{j\in\tau_l} p_j^2 + (m - m_l)t_\sigma^2\right) \\ &= \frac{1}{2}\sum_{j\in\tau_{long}} p_j^2 + \frac{1}{2}\sum_{j\in\tau'} p_j^2 \\ &= \frac{m_t}{2}p_t^2 + \frac{1}{2}\left(m_t\, {p'_t}^2 + (m - m_t){p'_b}^2\right) \\ &= \frac{m_t}{2}\left(p_t^2 + {p'_t}^2\right) + \frac{m - m_t}{2}\, {p'_b}^2. \end{aligned}$$

If the interval $[0, p_t]$ is a full interval in the optimal schedule of job system $\tilde{\tau}$ then $C^*(\tilde{\tau}) \leq C^*(\tau)$ clearly holds. Otherwise, we have

$$C^*(\tilde{\tau}) = m_t\, p_t^2 + \frac{\left(\sum_{j\in\{\tau_{short}\}} p_j\right)^2}{2(m - m_t)}.$$

Therefore, we obtain

$$C^*(\tau) - C^*(\tilde{\tau}) = \frac{m_t}{2}\left({p'_t}^2 - p_t^2\right) + \frac{m - m_t}{2}\left({p'_b}^2 - \left(\frac{\sum_{j\in\{\tau_{short}\}} p_j}{m - m_t}\right)^2\right).$$

From Inequality (4.4), we obtain

$$(m - m_t)\, p'_b \leq \sum_{j\in\{\tau'\setminus\tau'_t\}} p_j = (m - m_t)\, t_\sigma$$

However, in this case, we have $(m - m_t)\, t_\sigma < \sum_{j\in\{\tau_{short}\}} p_j$. This leads to

$$p'_b \leq t_\sigma < \frac{\sum_{j\in\{\tau_{short}\}} p_j}{m - m_t} \iff p'_t > p_t.$$

From Inequality (4.3), we know that there is $C^*(\tau) - C^*(\tilde{\tau}) > 0$ if $p'_t > p_t$ hold.

■

Therefore, we are ready to give the result of KAWAGUCHI and KYAN [44] in the following lemma.

Lemma 4.2 *For any job system τ with equal priority jobs and a list schedule S we have $\frac{C(S)}{C^*(\tau)} \leq \frac{\sqrt{2}+1}{2}$ if all jobs are released at time 0. This bound is tight.*

Proof: Due to Corollaries 2.2 and 4.3 we can assume that $\tau = \tau_{short} \cup \bar{\tau}(m_t)$ where τ_{short} is a set of jobs that contains only short jobs and $\bar{\tau}(m_t)$ is a rectangular job system with m_t long jobs. Without restriction of generality, let further be $\sum_{j \in \{\tau_{short}\}} p_j = m$. Then we have

$$\begin{aligned} C(S) &= \frac{m}{2} + m_t\, p_t\, (p_t + 1) \qquad \text{and} \\ C^*(\tau) &= m_t\, p_t^2 + (m - m_t) \cdot \frac{1}{2}\left(\frac{m}{m - m_t}\right)^2 \end{aligned}$$

if $p_t \geq \frac{m}{m-m_t}$ holds. It is easy to see that the case $p_t < \frac{m}{m-m_t}$ can be ignored. A simple optimization leads to

$$\max_{m_t \geq 0,\, p_t > 0} \left\{ \frac{C(S)}{C^*(\tau)} \right\} = \frac{\sqrt{2}+1}{2}.$$

The maximum value is obtained for $m_t = m\,(1 - \frac{\sqrt{2}}{2})$ and $p_t = 1 + \sqrt{2}$. ■

4.5 An upper bound of equal priority completion time

Finally, we are ready to prove our main theorem. We use Lemma 3.1 to address the equal priority completion time criterion.

Theorem 4.1 $\frac{C_{equ}(S)}{C^*_{equ}(\tau)} \leq 1.25$ *holds for any job system τ and any nondelay schedule S for τ on m identical machines.*

Proof: Due to Corollary 2.2, it is sufficient to consider only basic job systems with $\epsilon \to 0$ and their basic nondelay schedules. We assume that the first release date is always 0 even if there are no jobs that are released at this time.

At first we define r, τ_r, t_S, t_σ, and σ as in Lemma 3.1. Time instant $\tilde{r}$ is the highest release date of any job in job system τ such that no short job $j \in \tau$ with $r_j < \tilde{r}$ completes after time $\tilde{r}$ in the optimal schedule σ of job system τ. Further, we introduce $\tilde{\tau} = \{j \in \tau | r_j \geq \tilde{r}\}$.

As in Lemma 3.1, we prove this theorem by induction on the number k of different release dates. We assume that $\frac{C_{equ}(S)}{C^*_{equ}(\tau)} \leq 1.25$ holds for all considered basic job systems with at most k different release dates. From KAWAGUCHI's and KYAN's result [44], we know that the assumption is valid for $k = 1$ as we have $\frac{1+\sqrt{2}}{2} < 1.25$.

Again, we need to address two cases:

1. $\tilde{r} > 0$. Assume a full interval $[t_a, t_b)$ in S with $t_a \geq \tilde{r}$. From the third property of Definition 2.3, we know that $C_j(S) - (p_j + r_j) > (t_b - t_a)$ holds for all jobs $j \in \tau \backslash \tilde{\tau}$ with $C_j(\sigma) = r_j + p_j > t_a$. Together with Lemma 3.1, this results in

$$\begin{aligned} U_{[t_a,t_b)}(S, \tau\backslash\tilde{\tau}) &\leq \sum_{j \in \tau\backslash\tilde{\tau} \wedge C_j(S) > t_a} \Big(C_j(S) - \max\{t_a, p_j + r_j\} \Big) \\ &\leq \sum_{j \in \tau\backslash\tilde{\tau} \wedge C_j(S) > \tilde{r}} \Big(C_j(S) - \max\{\tilde{r}, p_j + r_j\} \Big) \\ &= D_{\tilde{r}}(S, \tau\backslash\tilde{\tau}) \leq \frac{1}{4} U^*_{[0,\tilde{r})}(\tau\backslash\tilde{\tau}) \leq \frac{1}{4}\tilde{r}m \; . \end{aligned}$$

   Let $\tilde{S}$ and $\tilde{\sigma}$ be the basic schedule and the optimal schedule for job system $\tilde{\tau}$, respectively. If the same scheduling order is used for all short jobs of job system $\tilde{\tau}$ in schedules S, $\tilde{S}$, σ, and $\tilde{\sigma}$ then we have

$$\begin{aligned} C_j(S) &\leq C_j(\tilde{S}) + \frac{1}{4}\tilde{r} \quad \text{and} \\ C_j(\sigma) &\geq C_j(\tilde{\sigma}) \end{aligned}$$

   for all jobs $j \in \tilde{\tau}$. Remember that $\tilde{\tau}$ has at most k different release dates that are greater than 0.

By using the induction assumption twice, we obtain

$$\begin{aligned}
\frac{C_{equ}(S)}{C^*_{equ}(\tau)} &= \frac{\sum_{j\in\{\tau\setminus\tilde{\tau}\}} p_j\, C_j(S) + \sum_{j\in\tilde{\tau}} p_j\, C_j(S)}{\sum_{j\in\{\tau\setminus\tilde{\tau}\}} p_j\, C_j(\sigma) + \sum_{j\in\tilde{\tau}} p_j\, C_j(\sigma)} \\
&\leq \frac{1.25\, C^*_{equ}(\tau\setminus\tilde{\tau}) + C_{equ}(\tilde{S}) + 0.25\,\tilde{r}\sum_{j\in\tilde{\tau}} p_j}{C^*_{equ}(\tau\setminus\tilde{\tau}) + C^*_{equ}(\tilde{\tau})} \\
&\leq \frac{1.25\, C^*_{equ}(\tau\setminus\tilde{\tau}) + \Big(C_{equ}(\tilde{S}) - \tilde{r}\sum_{j\in\tilde{\tau}} p_j\Big) + 1.25\,\tilde{r}\sum_{j\in\tilde{\tau}} p_j}{C^*_{equ}(\tau\setminus\tilde{\tau}) + C^*_{equ}(\tilde{\tau})} \\
&< \frac{1.25\, C^*_{equ}(\tau\setminus\tilde{\tau}) + 1.25\Big(C^*_{equ}(\tilde{\tau}) - \tilde{r}\sum_{j\in\tau_r} p_j\Big) + 1.25\,\tilde{r}\sum_{j\in\tilde{\tau}} p_j}{C^*_{equ}(\tau\setminus\tilde{\tau}) + C^*_{equ}(\tilde{\tau})} \\
&= 1.25.
\end{aligned}$$

2. $\tilde{r} = 0$. First, we introduce our so called primary transformation. To this end, we consider a long job $j_0 \in \tau$ with $t_S \geq C_{j_0}(S) > r$. We generate job system τ' by splitting this long job into another long job j_1 with $p_{j_1} = p_{j_0} - \epsilon$, $r_{j_1} = r_{j_0}$ and a short job j_2 with $p_{j_2} = \epsilon$, $r_{j_2} = r$. The resulting schedule S' is a basic nondelay schedule for the new job system τ' with $t_{S'} = t_S$. Figure 4.3 illustrates this primary transformation process of the basic nondelay schedule S of τ into the new schedule S' of the generated job system τ'. Corollary 2.1 yields

$$\frac{\sum_{j\in\tau'} p_j\, C_j(S')}{\sum_{j\in\tau'} p_j\, C_j(\sigma')} \geq \frac{\sum_{j\in\tau} p_j\, C_j(S)}{\sum_{j\in\tau} p_j\, C_j(\sigma)}\,.$$

Although $t_\sigma = t_{\sigma'}$ is still valid, we may have $\tilde{r}' > 0$ as $p_{j_0} + r_{j_0} < r$ leads to $C_{j_0}(\sigma) < C_{j_2}(\sigma')$. Then, we are back to Case 1. The primary transformation process of the optimal schedule σ into its corresponding optimal schedule σ' of job system τ' is illustrated in Figure 4.4. Let $\tau'_r \subset \tau'$ is the set of jobs that are released at time r.

If we still have $\tilde{r}' = 0$ and there is no long job $j \in \tau'$ with $t_{S'} \geq C_j(S') > r$, we split each long job $j' \in \tau'_r$ into a long job j'' with $p_{j''} = p_{j'} - \epsilon$ and a short job. Both jobs have release date $r'' = r + \epsilon$. Further, we increase the release date of all short jobs

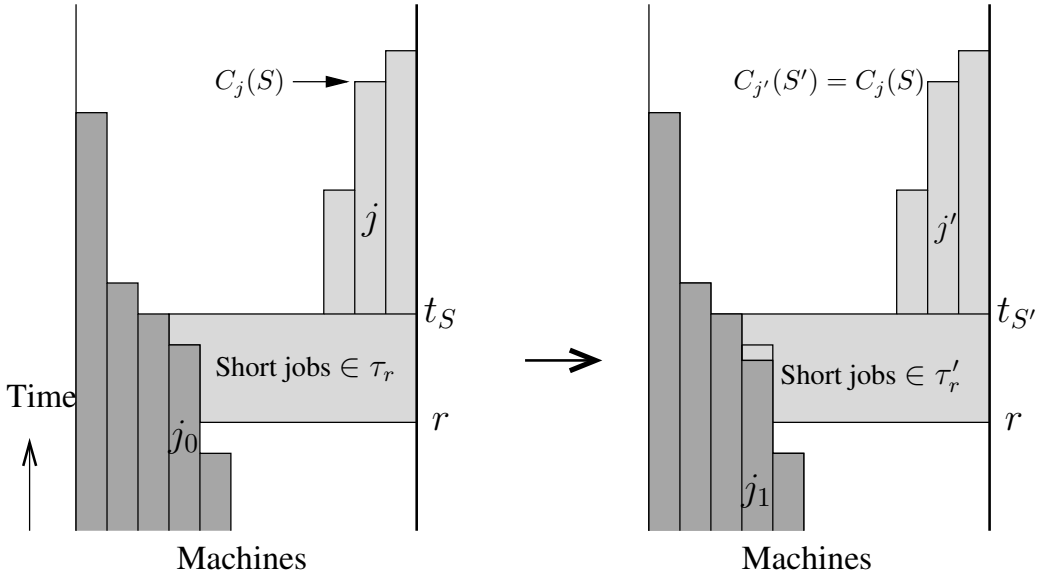

Figure 4.3: *Illustration of the primary transformation process from schedule S (left) of τ into schedule S' (right)of τ'*

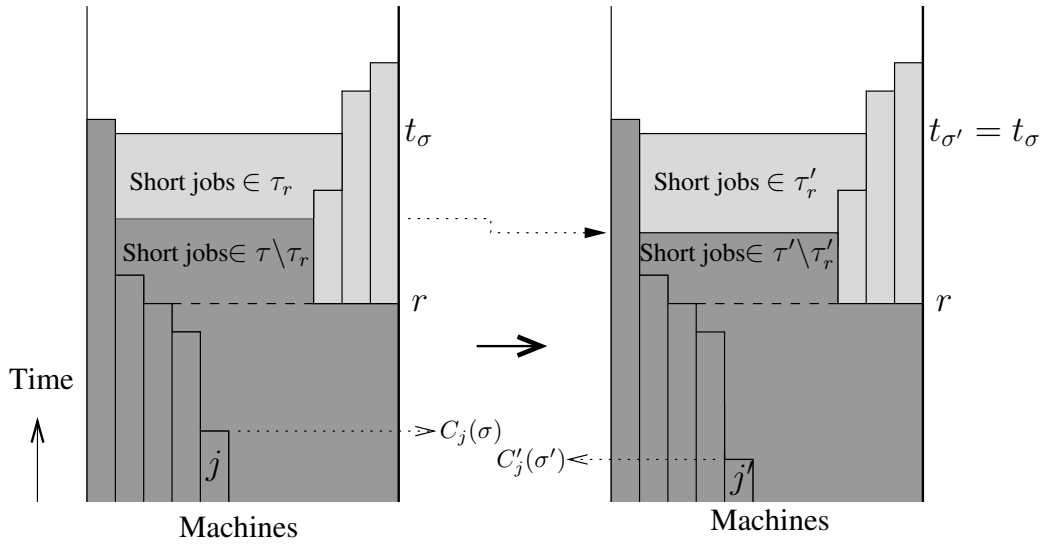

Figure 4.4: *Illustration of the primary transformation process from the optimal schedule σ of τ into the optimal schedule σ' of τ'.*

from τ'_r to r''. This transformation yields job system τ'', basic nondelay schedule S'', and optimal schedule σ''. The transformation process from schedules S or σ of job system τ' into schedules S' or σ' of the produced job system τ'' is illustrated in Figures 4.5 and 4.6. It is very similar to the transformation in Case 2(d)i of the proof for Lemma 3.1. In the produced basic schedule S'', each long job $j'' \in \tau''_r$ will complete later than the corresponding long job $j' \in \tau'_r$ in the basic schedule S' as we have $t_{S''} > t_{S'} + \epsilon$. This is due to that no long job from τ' completes within the interval $(r, t_{S'}]$ in schedule S', that is, the number of idle machines at $t_{S'}$ in schedule S' is less than the number of short jobs that start at r in the same schedule S'. On the other hand, the completion times of job j'' in the optimal schedule σ'' and of job j' in the optimal schedule σ' are identical. Further, there is $t_{\sigma'} = t_{\sigma''}$ as the interval $(\tilde{r}'', t''_\sigma]$ is a full interval in optimal schedule σ''. Together with Corollary 2.1, this results in

$$\frac{\sum_{j\in\tau''} p_j\, C_j(S'')}{\sum_{j\in\tau''} p_j\, C_j(\sigma'')} \geq \frac{\sum_{j\in\tau'} p_j\, C_j(S')}{\sum_{j\in\tau'} p_j\, C_j(\sigma')}\,.$$

The repeated application of this transformation will result either in $\tilde{r}'' = r''$ (Case 1) or lead back to the beginning of this case if there is a long job $j \in \tau''\backslash\tau''_r$ with $C_j(S'') = t_{S''}$.

■

Note that the result of Theorem 4.1 is not tight. However, the gap is very small as Kawaguchi and Kyan have shown that there are job systems with a single release date that come arbitrary close to the bound $\frac{1+\sqrt{2}}{2} \approx 1.207 < 1.25$. Therefore, we conjecture that Kawaguchi's and Kyan's bound is also a tight upper bound for the ratio $\frac{C_{equ}(S)}{C^*_{equ}(\tau)}$ in the multi-release date case.

Moreover, this worst case bound for the equal priority completion time is slightly smaller than the worst case bound for the utilization that has been derived in the previous chapter. This confirms the observations from Table 2.1.

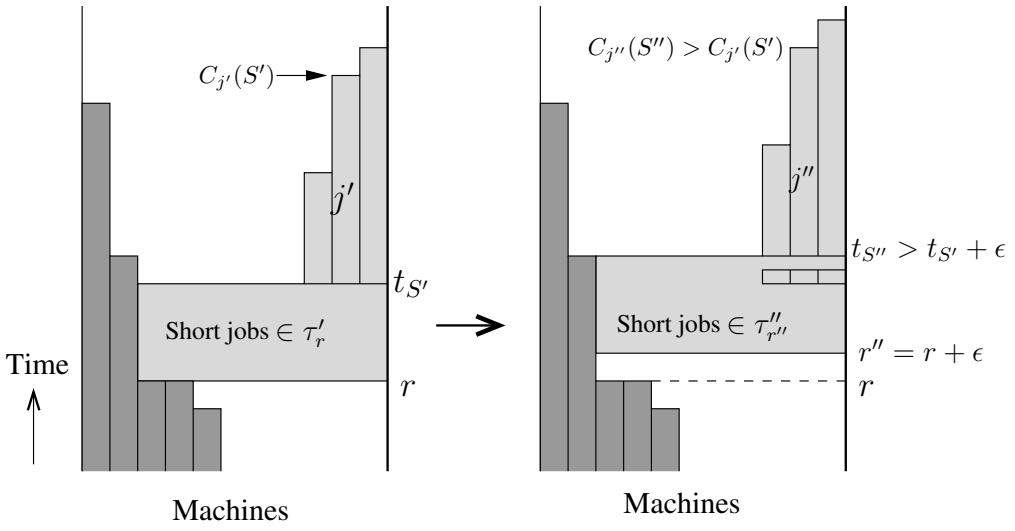

Figure 4.5: *Illustration of the generation process of schedule S'' (right) of the new job system τ'' from schedule S' (left) of job system τ'.*

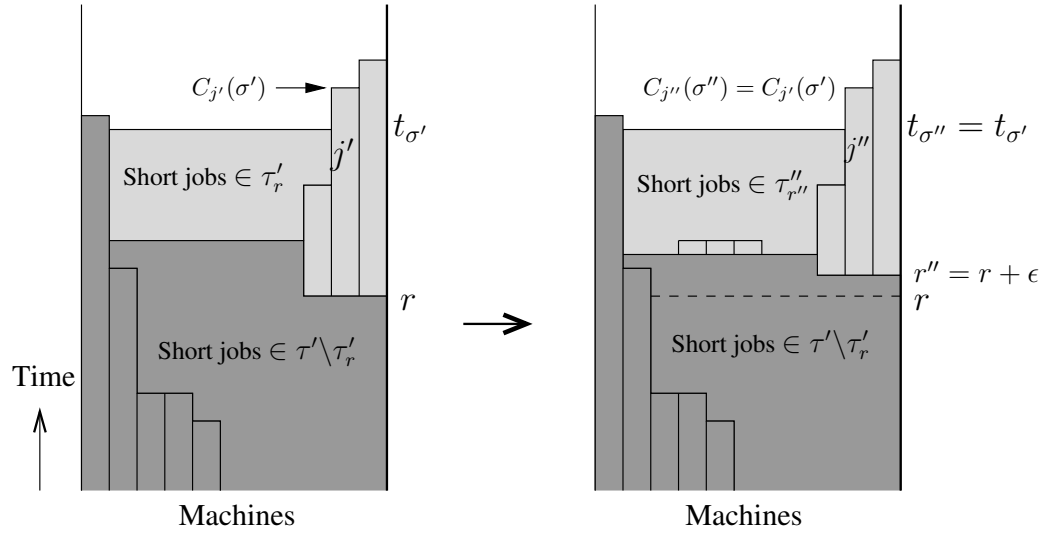

Figure 4.6: *Illustration of the generation process of the optimal schedule σ'' (right) of the new job system τ'' from the optimal schedule σ' (left) of job system τ'.*

4.6 The applicability of equal priority flow time

For the online (nonclairvoyant) scheduling problems, a widely used criterion to measure the quality of service (QoS) provided to users (i.e. the responsiveness of the system) is the average (weighted) flow time of the jobs, that is the average (weighted) time spent by jobs in the system between release and completion. Therefore, in this section, we are interested to discuss whether it is appropriate to use an equal priority flow time criterion $F_{equ}(S) = \sum_{j\in\tau} p_j\Big(C_j(S) - r_j\Big)$ to quantitatively represent the utilization for our kind of online scheduling problems. Note that the equal priority flow time criterion $F_{equ}(S)$ is modelled in the same fashion as the equal priority completion time.

To do so, we consider a specific basic job system τ. In τ, there are $m-1$ long jobs with processing time m and short jobs with a total resource consumption (machine time product) of m. All those jobs are released at time 0. In addition, $m-1$ long jobs with processing time 1 and short jobs with a total resource consumption of 1 are released at time $m+h$ for all integer h from 0 to $k-1$. Let schedule S be the basic non-delay schedule and schedule σ be the optimal schedule for τ, respectively. In schedule S all short jobs with release time 0 are executed in interval $[0, 1)$ on m machines while all long jobs with release time 0 are started at time 1. Further, the short jobs that are released at time $m+h$ are executed in the interval $[m+h,\ m+h+1)$ while all long jobs that are released at $m+h$ are started at time $m+h+1$. Note that, during the interval $[0,\ m+k+1)$ in such schedule S one of the m machines must be idle in the interval $[1,\ m)$ and one machine must be idle in the interval $m+k,\ m+k+1$. However, in the optimal schedule σ, there are no any intermediate idle times during the whole interval $[0,\ m+k)$ and all long jobs are started at their release dates. Now, we are going to calculate the utilization, the equal priority completion time, and the equal priority flow time. For the utilization, we have

$$U_{[0,m+k)}(S,\tau) = m(m-1) + mk + 1 \qquad \text{and} \qquad U^*_{[0,m+k)}(\tau) = m(m+k)$$

Then we have

$$\begin{aligned}\frac{U^*_{[0,m+k)}(\tau)}{U_{[0,m+k)}(S,\tau)} &= \frac{m(m-1)+m\,k+1+(m-1)}{m(m-1)+m\,k+1}\\ &= 1+\frac{m-1}{m(m-1)+m\,k+1}\,.\end{aligned}$$

For the equal priority completion time criterion, we have

$$\begin{aligned}\sum_{j\in\{longjobs\}} p_j\,C_j(S) &= m(m-1)(m+1)+(m-1)\Big((m+2)+(m+3)+\cdots+(m+k+1)\Big)\\ &= m(m^2-1)+(m-1)\Big(m\,k+\frac{k+3}{2}\cdot k\Big) \qquad \text{and}\\ \sum_{j\in\{shortjobs\}} p_j\,C_j(S) &= \frac{m}{2}+m\,k+\frac{k^2}{2}\end{aligned}$$

Note that the second sum can be obtain by using Corollary 4.1. This results in

$$\begin{aligned}C_{equ}(S) &= \sum_{j\in\{longjobs\}} p_j\,C_j(S)+\sum_{j\in\{shortjobs\}} p_j\,C_j(S)\\ &= m^3+k\,m^2-\frac{m}{2}+\frac{mk^2}{2}+\frac{3m\,k}{2}-\frac{3k}{2}\\ &= k\Big(\frac{m(k-1)-1}{2}+m^2+m\Big)+m^3-\frac{m^2}{2}+(m-1)(k+\frac{m}{2}).\end{aligned}$$

Similarly,

$$\begin{aligned}C^*_{equ}(\tau) &= \sum_{j\in\{longjobs\}} p_j\,C^*_j+\sum_{j\in\{shortjobs\}} p_j\,C^*_j\\ &= (m-1)m^2+(m-1)\Big((m+1)+(m+2)+\cdots+(m+k)\Big)+\frac{(m+k)^2}{2}\\ &= (m-1)m^2+(m-1)\Big(m\,k+\frac{k(k+1)}{2}\Big)+\frac{(m+k)^2}{2}\\ &= k\Big(\frac{m(k-1)-1}{2}+m^2+m\Big)+m^3-\frac{m^2}{2}.\end{aligned}$$

Therefore, we have

$$\frac{C_{equ}(S)}{C^*_{equ}(\tau)} = 1+\frac{(k+\frac{m}{2})(m-1)}{k\Big(\frac{m(k-1)-1}{2}+m^2+m\Big)+m^3-\frac{m^2}{2}}$$

	$\frac{U^*_{[0,m+k)}(\tau)}{U_{[0,m+k)}(S,\tau)}$	$\frac{C_{equ}(S)}{C^*_{equ}(\tau)}$	$\frac{F_{equ}(S)}{F^*_{equ}(\tau)}$
$k = 0$	$1 + \frac{m-1}{m^2-m+1}$	$1 + \frac{m-1}{2m^2-m}$	$1 + \frac{m-1}{2m^2-m}$
$k >> m^2 >> 1$	$1 + \frac{1}{k}$	$1 + \frac{2}{k}$	2

Table 4.1: Comparison between Utilization, Equal Priority Completion Time, and Equal Priority Flow Time for Selected Schedules

Finally, the ratio of the equal priority flow time can be derived as follows with the help of the above results of the equal priority completion time and the processing time of any job is at most m

$$\begin{aligned}
\frac{F_{equ}(S)}{F^*_{equ}(\tau)} &= \frac{\sum_{j\in\tau} p_j\, C_j(S) - \sum_{j\in\tau} p_j\, r_j}{\sum_{j\in\tau} p_j\, C^*_j - \sum_{j\in\tau} p_j\, r_j} \\
&= \frac{C_{equ}(S) - \sum\limits_{h=0}^{k-1}\left((m+h)\cdot \sum\limits_{i\in\tau|r_i=m+h} p_i\right)}{C^*_{equ}(\tau) - \sum\limits_{h=0}^{k-1}\left((m+h)\cdot \sum\limits_{i\in\tau|r_i=m+h} p_i\right)} \\
&= \frac{C_{equ}(S) - m\sum\limits_{h=0}^{k-1}(m+h)}{C^*_{equ}(\tau) - m\sum\limits_{h=0}^{k-1}(m+h)} \\
&= \frac{C_{equ}(S) - m\Big(m\,k + \frac{(k-1)k}{2}\Big)}{C^*_{equ}(\tau) - m\Big(m\,k + \frac{(k-1)k}{2}\Big)} \\
&= 1 + \frac{(k+\frac{m}{2})(m-1)}{(k+m^2)(m-\frac{1}{2})}.
\end{aligned}$$

Observe that the term $\sum_{j\in\tau} p_j\, r_j$ is independent of any schedule.

The comparison results between the criteria utilization, equal priority completion time, and equal priority flow time are displayed in Table 4.1 for two different cases

$k = 0$ and $k >> m^2 >> 1$. While utilization and equal priority completion time criteria show a very similar behavior the criterion equal priority flow time deviates significantly. Although the interval $[m, m + k)$ is a full interval in schedule S, the ratio $\frac{F_{equ}(S)}{F^*_{equ}(\tau)}$ increases with growing k. Therefore, equal priority flow time is not suited to describe utilization in our online model.

Chapter 5

Experimental Study

In this chapter, we experimentally analyze the performance of the nondelay online algorithm with respect to our new criteria *utilization* and *equal priority completion time*. That is, the study is devoted to confirm experimentally our upper bounds which have been derived theoretically in Chapters 3 and 4 for our new criteria. Therefore, we have conducted extensive computational experiments to acquaint the agreement between the theoretical and the experimental results. Moreover, we want to consider the effect of the size of the problem on the competitive ratios as well as the effect of larger machine numbers. To achieve these goals, a broad range of sets of problem instances are designed to provide a rich test set for investigating. The experimental results have been obtained according to job instances generated by using fundamental probability distributions. A detailed description of the experimental design is given in the following section. Then, in section 5.2, we provide an approach that is used to obtain near-optimal solutions of both criteria. Finally, in Section 5.3 we discuss the obtained results and report the analysis of the experiments.

5.1 Experimental Design

5.1.1 Computing Environment

The experiments were conducted on a Pentium(R) 4 PC with 2.6 GHz clock rate and 0.99 GB of memory, operating under Linux (Debian 3.3). The scheduling program is

coded in the C++ computer language, which reads the problem parameters from an input file and generates the desired schedules. The program has been compiled with the GNU g^{++} compiler Version 3.3.5 using the -O2 optimization option.

5.1.2 Benchmark Instances

In this section, we present the data generation scheme to create test problems. The benchmark instances were randomly generated and primarily consist of 9 sets with 3, 5, 10, 15, 20, 50, 100, 200, and 500 machines. For each selected value of m, various distribution functions have been considered to produce several different problem instances in a subset.

Number of jobs: The number of jobs at each release date is created by using a random generation for a *Poisson* distribution $P(\lambda)$. This distribution is commonly used to model the number of events occurring within a given time interval. The parameter λ is the shape parameter which indicates the average number of events (the positive mean). Clearly, the number of jobs designates the size of the problem instance. As the number of jobs increases, computational burden and hence the time needed to find the solution increases. For a small number of machines (up to 20 machines), we considered 25 different values of λ between $1.5m$ and $40m$, results in 125 problem instances. For larger machine numbers (50-500 machines), four different values of $\lambda = 1.5m,\ 2m,\ 2.5m,$ and $3m$ have been considered resulting in another 16 problem instances. In addition, for each selected number of jobs and fixed number of machines, several individual problem instances have been generated from different distribution functions.

Processing time: We tested many different probability distribution functions to model the processing times of the jobs. We found that most of these distributions, like Gamma, Poisson, Weibull and a lot of others, give small competitive ratios. That is, the nondelay online algorithm performs well with these distributions as the competitive ratios were close to 1. Since we want to evaluate the performance of the nondelay online algorithm with respect to our new criteria from the worst case point of view, we need come as close as possible to the worst (largest) ratios. We found three differ-

ent distribution functions which can capture a bad behavior of the algorithm. Therefore, the processing times of the jobs have been generated randomly according to those following probability distributions: *exponential*, *chi-square*, and *log-normal* distribution. Each distribution is characterized by corresponding parameters that have been properly set in order to get realistic job instances. Therefore, when choosing the parameters of the distributions from which the processing times were generated we tried on the one hand to cover a wide range of values and on the other hand to create different degrees of variability in the job processing times.

For the exponential distribution exp(β), whose location parameter is equal to zero, β is often referred as scale parameter which equals $\frac{1}{mean}$. We considered a total of 5 β values $1,\ \frac{1}{2},\ \frac{1}{3},\ \frac{1}{4}$, and $\frac{1}{5}$. This yields 705 problem instances generated from such distribution. For the log-normal distribution $L(\mu,\ \sigma)$, where μ and σ are the mean (scale parameter) and the standard deviation (shape parameter) of the distribution on the log scale, we fixed $\sigma = 1$. Then, we considered four values of $\mu = 0,\ 0.5,\ 0.8,$ and 1. Consequently, the number of problem instances which were generated from this distribution is 564. Finally, the processing times are created by using a random generation for the Chi-Square distribution which has a shape parameter ν_1 and a scale parameter ν_2. We considered ν_1 values of 2 and 3. For each value of ν_1, the corresponding four values of $\nu_2 = 0,\ 0.5,\ 0.8$, and 1 have been considered to create 8 problem instances from such distribution for each fixed number of machines and given number of jobs (size of the instance). This results in 1128 problem instances that have been generated from such distribution. Therefore, we considered total 2397 problem instances.

Release dates (r_j): Job arrival times are determined during the computational process. After all jobs with the current release date are scheduled, the next release date is selected to be one of all possible integer values between the smallest and the largest completion time of all machines. By this way we make sure that: (i) There is idle time between any two release dates. (ii) There is at least one scheduled job that prevents a new job to be scheduled at its release date. In each test problem, we considered 6 different release dates.

After generating the job instances, we ran the nondelay online algorithm on each

job instance and computed the competitive ratio of the utilization and equal priority completion time at each release date. Hence, the largest ratios are picked to represent the worst competitive ratios for such instance.

5.2 An approach for optimal solution

As we know, the competitive analysis is typically used to capture the worst case behavior of the algorithm. Therefore, the competitive ratio quantifies by how much, in the worst case, the online schedule deviates from the optimal schedule. Clearly, such competitive ratio requires an optimal solution. Consequently, in an experimental study it is necessary to find a solution which is as close as possible to the optimal solution to determine correctly such competitive ratio, thus to capture precisely the worst case. To achieve such goal, we proposed an approach to compute near optimal solution. In this approach, we are interested only to find the slots of time in which the machine is busy regardless of which or how many jobs are executed in this busy interval. Before we explain the algorithm we describe how one can calculate the equal priority completion time from the produced schedule. Let $[A_{i(k)}, B_{i(k)}]$ denotes to the i^{th} busy interval of machine M_k. Assume that l jobs with equal processing times, $p_j = p$ and $p \to 0$, are executed during this busy interval $[A_{i(k)}, B_{i(k)}]$. Therefore, we have $l \cdot p = \Big(B_{i(k)} - A_{i(k)}\Big)$. The equal priority completion time can be obtained as follows:

$$\begin{aligned}
\sum_j p_j\, C_j &= p \sum_j C_j \\
&= p\left(l \cdot A_{i(k)} + p\,(1 + 2 + \cdots + l)\right) \\
&= p\left(l \cdot A_{i(k)} + p\,\frac{l(l+1)}{2}\right) \\
&= p \cdot l\left(A_{i(k)} + \frac{1}{2}\,l \cdot p\right) + \frac{1}{2}\,l \cdot p^2 \\
&= \left(B_{i(k)} - A_{i(k)}\right)\left(A_{i(k)} + \frac{1}{2}\Big(B_{i(k)} - A_{i(k)}\Big)\right) + \frac{1}{2}\sum_j p_j^2 \\
&= \frac{1}{2}\left(B_{i(k)}^2 - A_{i(k)}^2\right) + \frac{1}{2}\sum_j p_j^2
\end{aligned}$$

Next, we describe our approach. Firstly, we sort the jobs in a list L according to the non-decreasing order of their release dates such that jobs with the same release date are sorted according to the non-increasing order of their processing times. The algorithm is as the following:

Step 1 Set $r = 0$ and $h(k) = 1,\ A_{h(k)} = B_{h(k)} = 0$ for each machine M_k.

Step 2 For every machine M_k, if $B_{h(k)} < r$ then increase $h(k)$ by one and let $A_{h(k)} = r$. That is, we start a new busy interval for this machine.

Step 3 For every idle machine M_k at the release date r, schedule the next job in the list L on such machine. Devote the completion time of this job to the value $B_{h(k)}$. Then re-index the machines such that $B_{h(1)} \geq B_{h(2)} \geq \cdots \geq B_{h(m)}$.

Step 4 let $k = \lambda_p = \lambda_m = 0$.

Step 5 Compute a time instant

$$TargetLine = \frac{\left(\sum\limits_{r_j<r<C_j}(C_j - r) + \sum\limits_{r_j=r} p_j\right) - \lambda_p}{m - \lambda_m}$$

increase k by 1;

Step 6 For the machine M_k, if $B_{h(k)} > TargetLine$ then put $\lambda_p = \lambda_p + (B_{h(k)} - r)$ and decrease λ_m by 1 and go to Step 5.
Otherwise, put $B_{h(k)} = TargetLine$ for all remaining machines.

Step 7 Remove all remaining jobs with release date r from the list L.
If $L \neq \phi$, let r be the next release date and go to Step 2.

Comment So far, we have determined all busy intervals of each machine. Now, we are ready to compute the utilization (U) and the equal priority completion time ($EPCT$) at each release date r.

Step 8 At each release date r, calculate the following for every machine M_k:

$$U = \sum_{i=1}^{h} \min\{B_{i(k)}, r\} - A_{i(k)} \quad \text{and}$$

$$EPCT = \frac{1}{2}\sum_{r_j \leq r} p_j^2 + \frac{1}{2}\sum_{i=1}^{h}\left(B_{i(k)}^2 - A_{i(k)}^2\right)$$

A detailed description of such algorithm is presented in AppendixA .

5.3 Analysis of the Results

In this section we present the experimental results of our tests and give a detailed analysis of the performance of the nondelay online algorithm with respect to our new criteria which have been introduced formally in the theoretical part. This performance is analyzed for instances in which the processing times of the jobs were generated according to the Exponential, Chi-Square, and Lognormal distribution.

Let us first discuss the computational results obtained for a small number of machines. In the following figures, we give the experiments for $m = 3,\ 5,\ 10,\ 15,$ and 20 a number of machines. For every number of machines, we present two figures. One describes the results for the utilization criterion and the other for the results of the equal priority completion time criterion. In each figure, the behavior of the (maximum) competitive ratios for various instances with different sizes is depicted. The size of any instance is described by the total number of jobs in such instance. Moreover, there are three different curves in every figure. Each one is corresponding to the probability distribution which is used to model the job processing times.

Figures 5.1-5.5(top) illustrate the results for the utilization criterion when the number of machines m is $3,\ 5,\ 10,\ 15,$ and 20, respectively. Figures 5.1-5.5(bottom) illustrate the results for equal priority completion time criterion with the same numbers of machines. As we observed from our theoretical investigation, we found from these experimental results that the ratios of the utilization criterion follow the same trend of the equal priority completion time ratios even with different machine numbers. One

can easily note this observation by comparing the curves of the utilization with the corresponding curves of the equal priority completion time for each number of machines. The experimental results which show this similarity of the trends of both criteria are reported in Appendix B. Moreover, in all cases of machine numbers, we find another observation that the competitive ratios decrease generally with increasing size of the instance. Therefore, one can achieve the worst ratio only when the considered instance has a small number of jobs.

For the utilization, the largest obtained ratio is 1.31 which has been achieved from the Chi-Square distribution when $m = 15$, see Figure 5.4(top). Clearly, this value is very close to our tight bound $\frac{4}{3} \approx 1.33$ which we have derived theoretically in Chapter 3. Consequently, these experimental results are coincident with our theoretical results for the utilization criterion.

In fact, in practice the behavior of the algorithm is frequently much better than the theoretical estimation. For the equal priority completion time criterion, no ratio could reach our upper bound 1.25 which we have obtained theoretically for such criterion in Chapter 4. Moreover, the experimental results could not reach even the value $\frac{1+\sqrt{2}}{2} \approx 1.207$ in any case. The largest ratio 1.65 is obtained from the Chi-Square distribution when the number of machines $m = 3$ and from the exponentially distributed job sizes when the number of machines $m = 20$, see Figures 5.1(bottom) and 5.5(bottom) respectively. These experimental results confirm our conjecture in the sense that KAWAGUCHI's and KYAN's result for the single release date case $\frac{1+\sqrt{2}}{2}$ is a tight bound for the online case as well.

As we mentioned above, the competitive ratio, for both criteria, decreases in general with the instances that contain larger number of jobs. This general observation holds for all distribution. However, we observed that the behavior of the competitiveness depends on the type of the distribution used to generate the processing times of the jobs. Let us discuss that in more details.

The following description of the developments of the competitive ratios holds for both criteria (utilization and equal priority completion time) in all cases, where it turned out that their trends are almost similar. The developments of the ratios un-

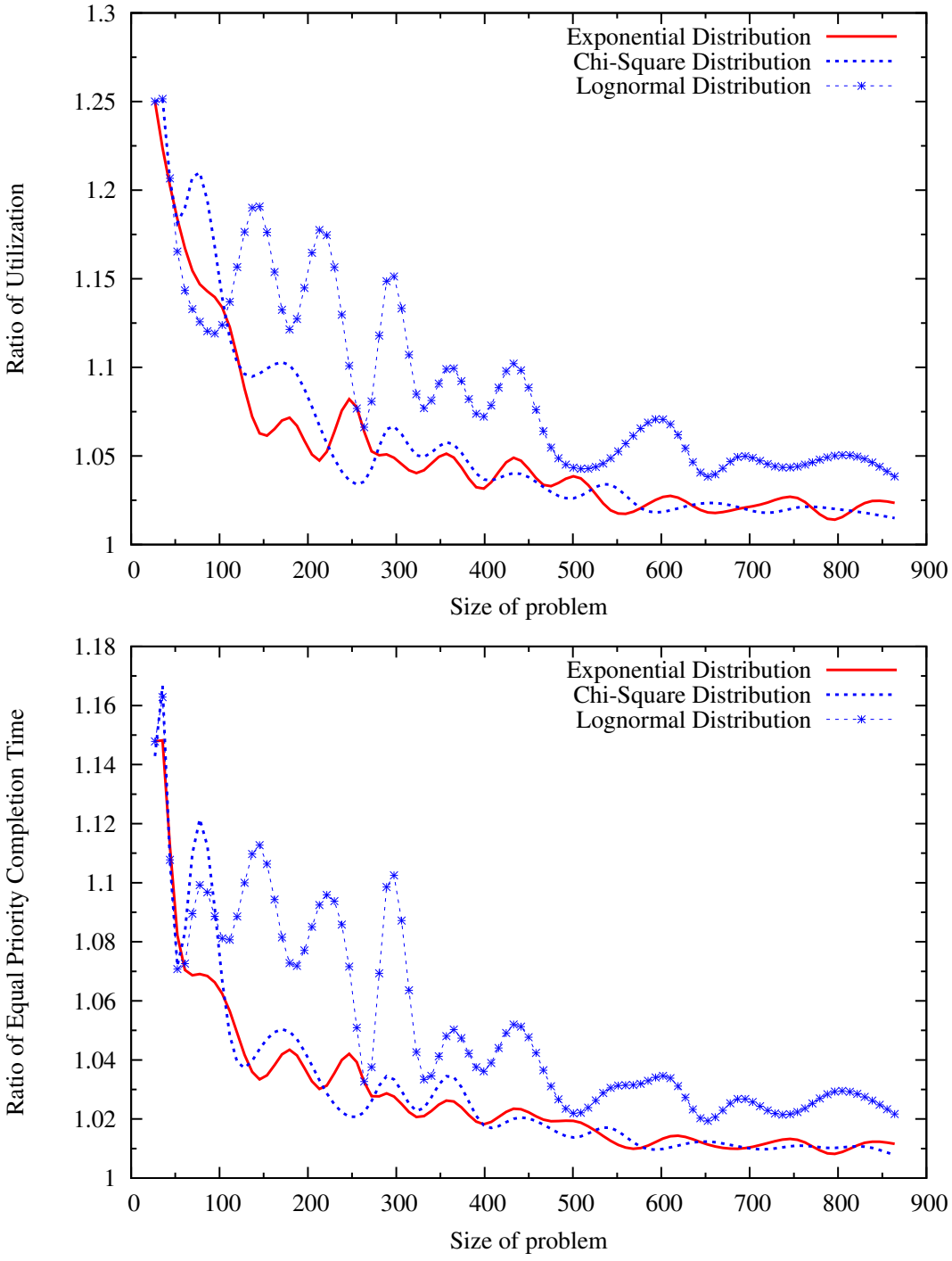

Figure 5.1: *The competitive ratio of the utilization and equal priority completion time with different number of jobs when* $m = 3$.

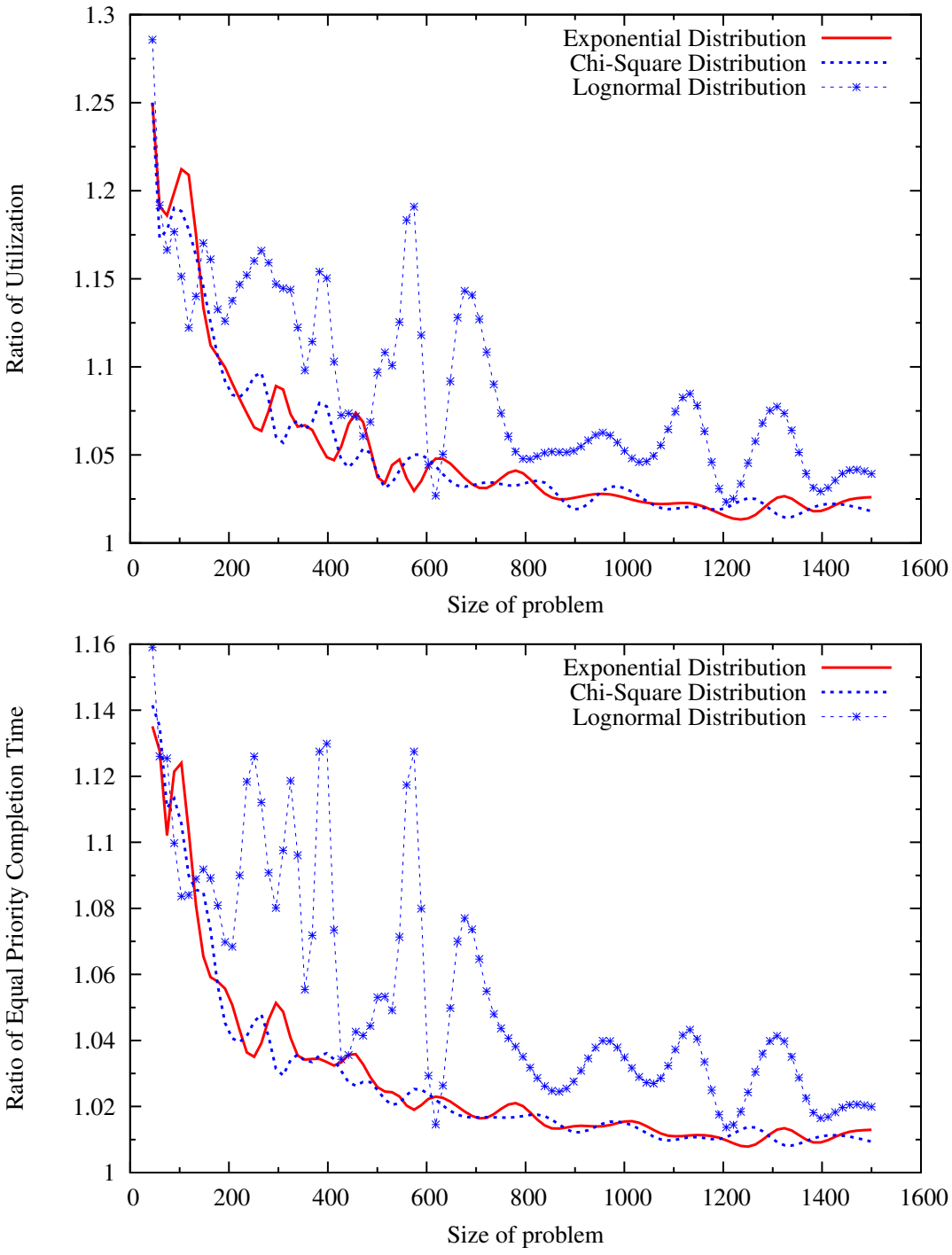

Figure 5.2: *The competitive ratio of the utilization and equal priority completion time with different number of jobs when* $m = 5$.

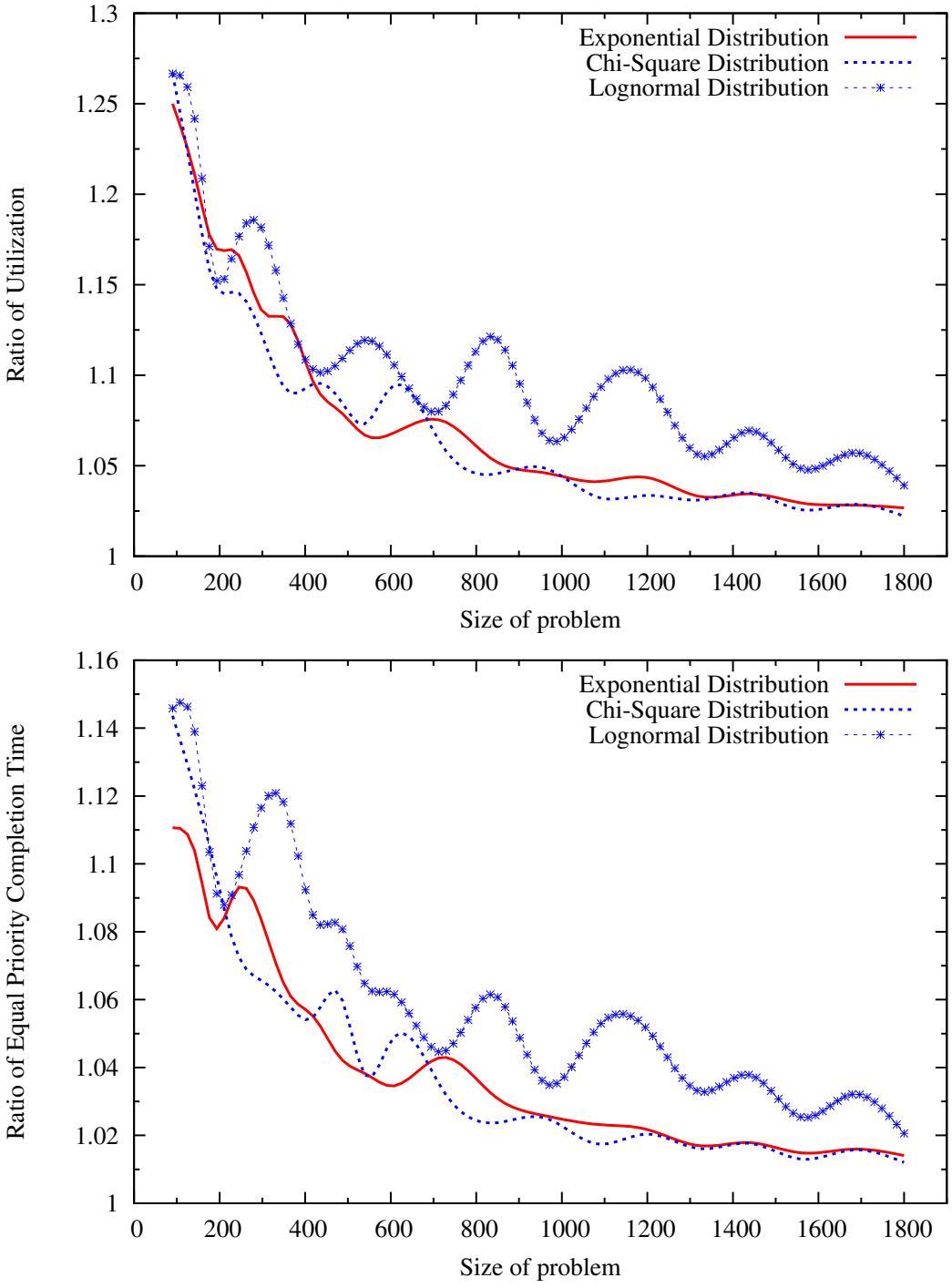

Figure 5.3: *The competitive ratio of the utilization and equal priority completion time with different number of jobs when* $m = 10$.

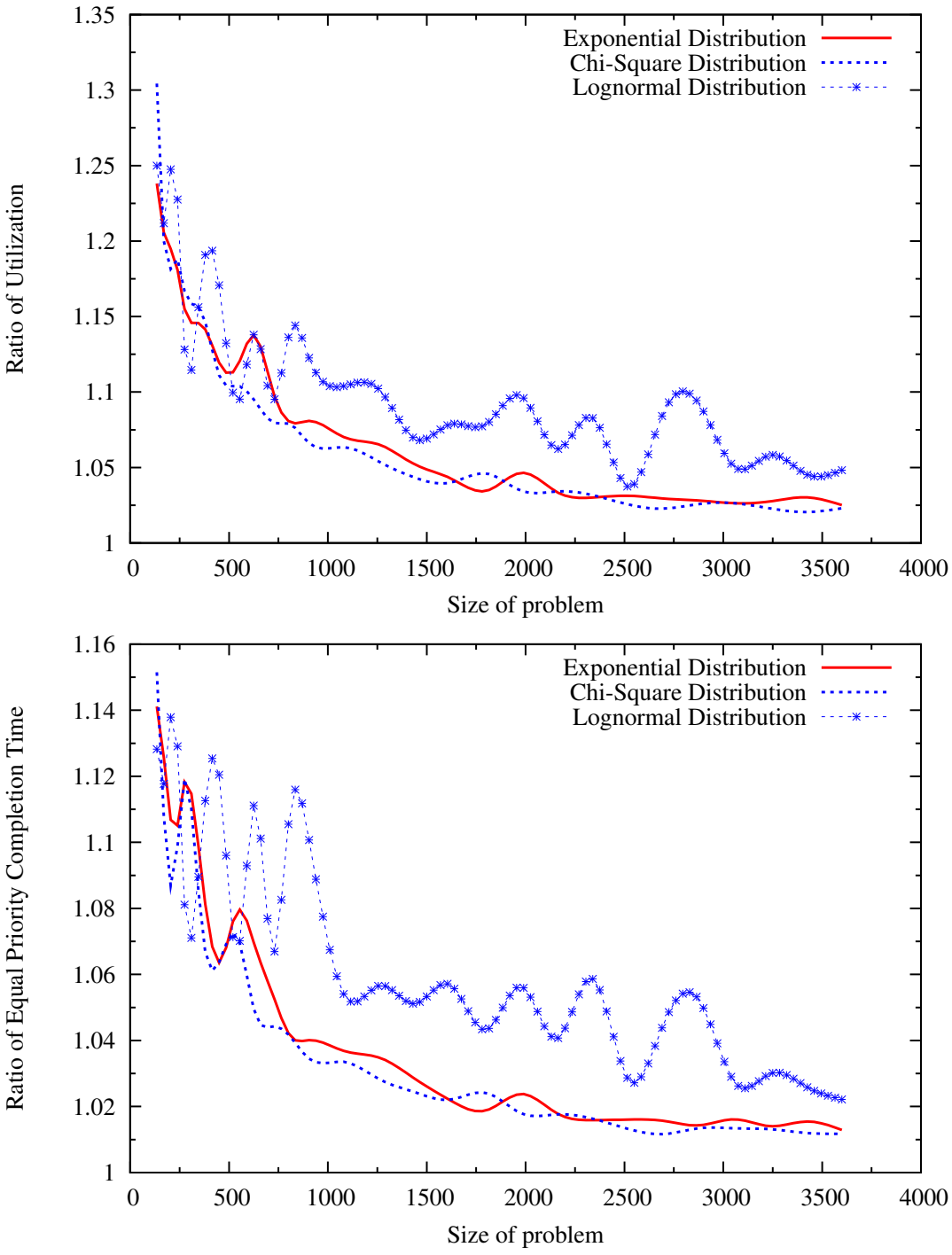

Figure 5.4: *The competitive ratio of the utilization and equal priority completion time with different number of jobs when* $m = 15$.

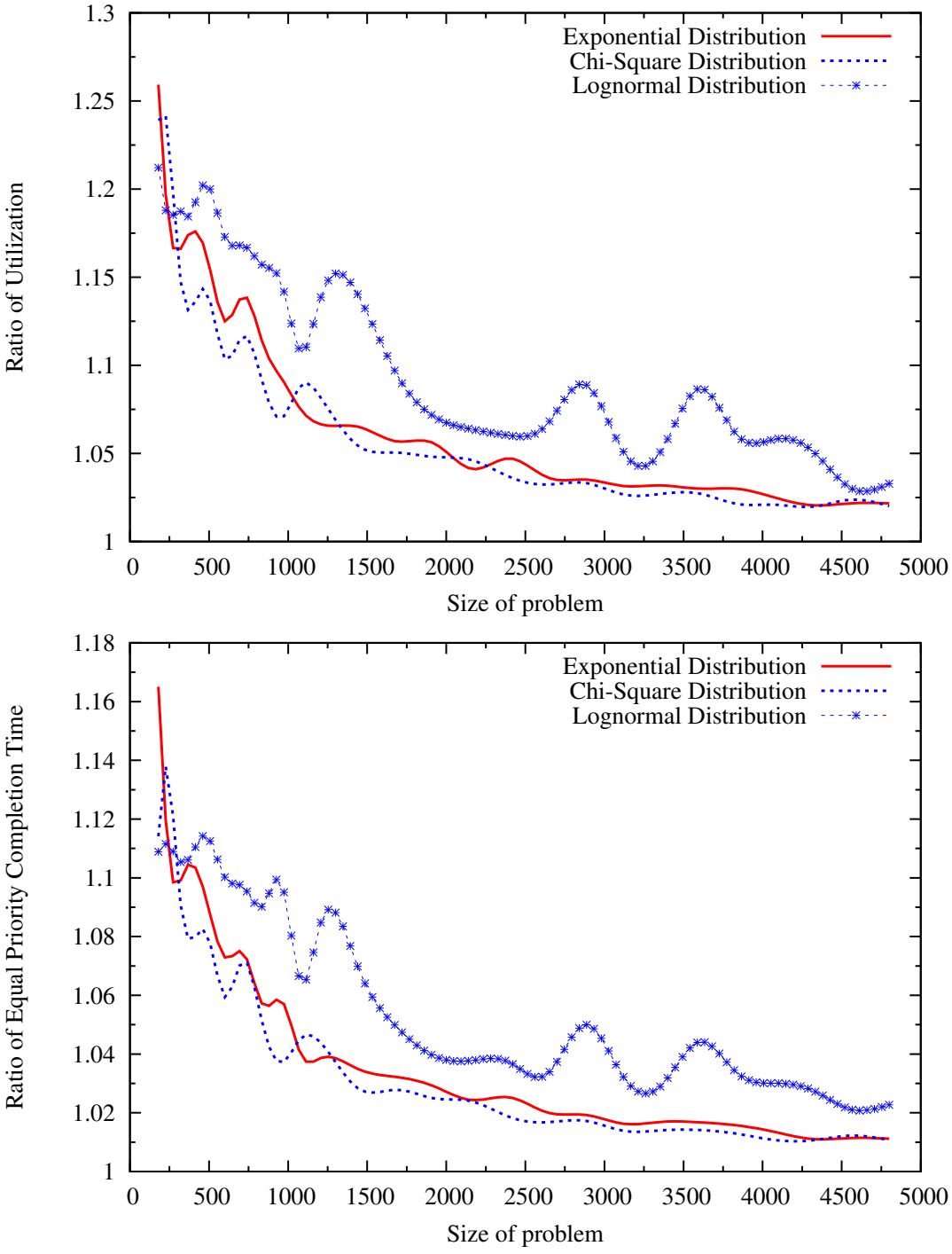

Figure 5.5: *The competitive ratio of the utilization and equal priority completion time with different number of jobs when* $m = 20$.

der the job instances generated by using exponential and chi-square distributions are very close to each other. That is, the trends of the ratios are almost coincident for these distributions. For the instances with a small size of up to about $120m$ jobs, the competitive ratio rapidly drops to reach a value around 1.05. After that the competitiveness tends to become more stable with very small oscillations until it converges toward 1.02 for the instances with a size of around $180m$ jobs.

For the Lognormally distributed job sizes, it is obvious that the competitive ratios oscillate greatly in the beginning. The magnitude of these oscillations tends to become smaller in the cases of larger machine numbers, compare Figure 5.1 with 5.5. This fluctuations of the competitive ratios converge gradually and slowly until the trend becomes stable around 1.05. It is possible to note a higher stability and faster convergence of the behavior of the competitive ratios under exponential and Chi-Square distribution in comparison to the corresponding behavior under the log-normal one.

Finally, it is also possible to note that the ratios with the log-normal distribution tend to be significantly higher than the ratios obtained by using the exponential and the chi-square distribution in all the different cases. This concludes that the nondelay online algorithm performs worse when the job instances are generated by using log-normal distribution.

The key point for the interpretation of these experimental results is the coefficient of variation of the job instances. It is well known that the variability in the job sizes is very low for the instances which were generated from either the exponential or the chi-square distribution. For example, the exponential distribution has a mean squared coefficient of variation of 1 independently of its mean. For such reason, these distributions generate job instances for which the competitive ratios of the algorithm are quite stable. On the other hand, by using log-normal distribution, it is possible to generate instances of jobs with extremely variable job sizes. That is, the variability in the job sizes is much higher for such distribution, thus obtaining bigger fluctuations and much slower convergence. This turned out that the variability in the job sizes is the crucial factor for the performance of the online algorithm. A similar experimental result has been obtained in [3] but for the makespan criterion.

m	Size of Instance			
	$9m$	$12m$	$15m$	$18m$
3	1.25	1.22222	1.2	1.2
5	1.25	1.19048	1.18644	1.2
10	1.25	1.16667	1.20301	1.175
15	1.2381	1.2	1.19008	1.15789
20	**1.25926**	1.18378	1.16418	1.173
50	1.24359	1.1631	1.20242	1.17021
100	1.23239	1.15916	1.18919	1.13314
200	1.21951	1.14314	1.16959	1.14041
500	1.20546	1.14779	1.19495	1.15543

Table 5.1: *The competitive ratios of the utilization criterion for exponentially distributed job processing times.*

All experimental results presented so far are for simulation with up to 20 machines. We obtained more experiments for job instances with 50, 100, 200, and 500 machines to consider the effect of larger machine numbers on the worst competitive ratios. As we mentioned above the the worst performance of the online algorithm can be obtained only with a small number of the jobs. Therefore, we considered job instances with four different number of jobs. For each number of machines (50, 100, 200, and 500), we generate job instances with $9m$, $12m$, $15m$, and $18m$ number of jobs by applying the same probability distributions used before. The results are reported in the Tables 5.1-5.6. The results for the small number of machines are presented in these tables as well to compare them with the results obtained for larger number of of machines. Tables 5.1-5.3 present the competitive ratios of the utilization criterion under different machine numbers for job instances generated by using exponential, chi-square, and log-normal distribution respectively. From these results, we found that the competitive ratios for larger machine numbers are close to the competitive ratios under small number of machines. For the exponentially distributed job sizes, the largest competitive ratio is 1.259 which has been attained when the number of machines $m = 20$ and the average num-

m	Size of Instance			
	$9m$	$12m$	$15m$	$18m$
3	1.25	1.25	1.2	1.22727
5	1.25	1.17241	1.17857	1.19048
10	1.26667	1.2	1.19048	1.15517
15	**1.30435**	1.18421	1.19048	1.16949
20	1.23913	1.23457	1.16471	1.13208
50	1.21488	1.18836	1.17057	1.1254
100	1.21951	1.16032	1.17647	1.15268
200	1.21702	1.15226	1.17857	1.14971
500	1.20482	1.15364	1.19066	1.13723

Table 5.2: *The competitive ratios of the utilization criterion for job instances generated by chi-square distribution.*

m	Size of Instance			
	$9m$	$12m$	$15m$	$18m$
3	1.25	1.25	1.2	1.21053
5	1.28571	1.19048	1.16667	1.17647
10	1.26667	1.15385	1.22807	1.16505
15	1.25	1.21429	1.2561	1.13846
20	1.21212	1.18462	1.18812	1.18421
50	1.19048	1.17021	1.20365	1.20148
100	1.25949	1.15234	1.21359	1.1592
200	1.28205	1.15976	1.20395	1.16306
500	**1.31926**	1.16974	1.20643	1.15292

Table 5.3: *The competitive ratios of the utilization criterion for job instances generated by using log-normal distribution.*

m	Size of Instance			
	$9m$	$12m$	$15m$	$18m$
3	1.14783	1.14695	1.10714	1.12827
5	1.13514	1.12727	1.1102	1.12271
10	1.11064	1.104	1.09993	1.08279
15	1.14114	1.12038	1.10197	1.11712
20	**1.16503**	1.1103	1.09734	1.10398
50	1.1382	1.09542	1.1092	1.09379
100	**1.15646**	1.11631	1.11296	1.08232
200	1.13889	1.08392	1.1121	1.08266
500	1.1348	1.08488	1.1174	1.09121

Table 5.4: *The competitive ratios of the equal priority completion time criterion under exponentially distributed job processing times.*

ber of jobs equal to $9m$, see Table 5.1. Further, the largest competitive ratio for the job instances generated from chi-square distribution is 1.304 which has been gained when the number of machines $m = 15$ with $9m$ average number of jobs. An interesting result is achieved from the job instances generated by using log-normal distribution with $9m$ as average number of jobs and when the number of machines $m = 500$. Whereas the worst achievable competitive ratio among all considered cases has been gotten from this large number of machines. This worst competitive ratio is $1.319 \approx 1.32$ which is almost our theoretical bound 1.33, see Table 5.3. This turns out that the worst ratio can be obtained from larger machine numbers as well. As expected, the experimental results could not achieve ratio larger than our theoretical bound for the utilization criterion although we considered several cases of machine numbers. This again confirms the validation of our theoretical result for such criterion.

For equal priority completion time, the competitive ratios under different number of machines are presented in Tables 5.4-5.6 for job instances again generated by using exponential, chi-square, and log-normal distribution respectively. We can conclude that the number of machines has no effect on the worst competitive ratio. For most

m	Size of Instance			
	$9m$	$12m$	$15m$	$18m$
3	1.14286	**1.16522**	1.09901	1.14286
5	1.14152	1.13479	1.10992	1.11353
10	1.14362	1.10431	1.11821	1.10345
15	1.15152	1.09861	1.08986	1.11793
20	1.11416	1.13799	1.10156	1.07992
50	1.10295	1.10464	1.10145	1.08211
100	1.11015	1.09485	1.11356	1.09811
200	1.11109	1.085	1.10177	1.0848
500	1.1037	1.09533	1.11704	1.07541

Table 5.5: *The competitive ratios of the equal priority completion time criterion for job instances generated by chi-square distribution.*

m	Size of Instance			
	$9m$	$12m$	$15m$	$18m$
3	1.14783	**1.16129**	1.1	1.10497
5	1.15909	1.12579	1.125	1.09804
10	1.14581	1.09072	1.13256	1.10004
15	1.12822	1.12036	1.14239	1.08632
20	1.1089	1.11142	1.10646	1.1057
50	1.11467	1.10945	1.12246	1.11961
100	1.1028	1.09466	1.11762	1.10889
200	1.13093	1.08797	1.11459	1.09478
500	1.11562	1.08517	1.10869	1.09836

Table 5.6: *The competitive ratios of the equal priority completion time criterion for job instances generated by using log-normal distribution.*

cases, it is noticeable that the competitive ratios for small and large number of machines are not far from each other. Although the worst ratios for job instances generated from the chi-square and log-normal distributions are achieved under small number of machines, the corresponding ratios for exponentially distributed job sizes are obtained with larger machine numbers. The largest competitive ratios for chi-square and log-normal distribution are 1.165 and 1.161 respectively. Both results are attained from job instance with $12m$ average number of jobs and when the number of machines $m = 3$, see Tables 5.5 and 5.6. However, the worst two competitive ratios for exponentially distributed job processing times are 1.165 and 1.157 which are obtained with $9m$ average number of jobs. These values are reached when the number of machines $m = 20$ and 100 respectively, see Table 5.4. These experimental results for larger machine numbers again confirm our theoretical bound 1.25 for equal priority completion time and even confirm our conjecture that the bound $\frac{1+\sqrt{2}}{2}$ is a tight bound for the online problem as well.

Chapter 6

Conclusion

In this chapter, we summarize the main contributions and results presented in the thesis. This thesis has provided work and progress relevant to online scheduling and scheduling criteria. The work is motivated by some practical applications which may occur in the electronic commerce. We have investigated the non-clairvoyant online scheduling problem on identical parallel machines from the owner point of view. Therefore, the objective was to maximize system utilization. The main contribution of this work was a formal introduction of two new online scheduling criteria that more accurately capture system utilization. Further, the analysis of the worst case difference between any nondelay schedules with those criteria is presented. In fact, our study can help researchers and system owners to use our new criteria as a well alternative to the usually used makespan criterion for machine utilization.

Firstly, we have considered the problem to determine performance measures that are well suited to evaluate the utilization of identical machines in a specific time interval. For the parallel machine problems, there is a published fact which states that the makespan is closely related to the utilization. However, in this work we have shown that this relation does not always hold. As a consequence, the commonly used makespan criterion may not reflect the true utilization of the system. To vanquish this shortcoming of the makespan, we have introduced formally two new alternative criteria *utilization* and *equal priority completion time*. Further, a comparison of our criteria with the classic makespan criterion has been provided. As a result, we found that our

new criteria are well suited to quantitatively describe machine utilization particulary for online scheduling problems. Furthermore, we observed that while the utilization and equal priority completion time criteria seem to behave in a similar fashion they do not yield the same quantitative results.

After the formal introduction of our criteria, we considered the maximization problem of our first criterion (utilization). Further, the worst competitive factor of such criterion is derived. We obtained an upper bound of $\frac{4}{3}$ for such criterion. Moreover, we provided the proof of the tightness of such bound.

Next, the equal priority completion time minimization problem is addressed. We showed that the competitive ratio of the total weighted completion time criterion is unbounded when the jobs have arbitrary weights (*i.e.* jobs have different priorities). For equal priority jobs, that is, $w_j = p_j$ holds for all jobs, we have proven an almost tight competitive factor of our second criterion (equal priority completion time) for our online model. We derived an upper bound of 1.25 for such criterion. Moreover, we conjecture that the bound $\frac{1+\sqrt{2}}{2} \approx 1.207$, which has been derived by KAWAGUCHI and KYAN for single release date case, is the best possible factor for the multi-release date as well. At the end of the theoretical part, we have shown that it is not appropriate to use the equal priority flow time criterion which is modelled in the same fashion as the equal priority completion time to describe machine utilization.

Finally, we provided an experimental part of this research. In this part, we experimentally evaluate the performance of the nondelay online algorithm with respect to our new criteria which are provided formally before. The performance of such algorithm has been evaluated according to job instances in which the job processing times were generated by using various fundamental probability distributions. As a result of our experimental investigation, it is possible to conclude the following.

The performance of the nondelay online algorithm for our criteria depends heavily on the characteristics of the generated job instances. Thus, the results differ substantially depending on the type of the respective distribution which is used to model and generate the job instances. Whereas, we found that the ratios for the job instances generated by log-normal distribution are worse than the ratios for the job instances

generated by either exponential or chi-square distribution and this result holds for all considered cases. The reason is that the job instances which are modelled by log-normal distribution exhibit a higher variability in the job processing times and that the results mainly depend on how strong the effects of very long jobs are. Consequently, our computational results show the importance of selecting the right probability distribution when assessing online nondelay schedules experimentally.

The worst case ratio can be obtained only from job instances with a small number of jobs. Whenever the size of the instance increases the worst competitive ratio decreases in general gradually. In addition, we observed that the behavior of the competitive ratio differ according to the used distribution. It is noticeable for the job instances, which are generated by the exponential and chi-square distribution, that the competitive ratios become stable when the ratio $\frac{\text{number of jobs}}{m}$ becomes greater than 50. On the other hand, the experiments for the job instances, which are generated from log-normal distribution, showed that the competitive ratios are scientifically higher and fluctuate greatly. In this case, the ratios converge much slower whereas the competitive ratios stabilize only when the job instance contains a large number of jobs. We noticed that the competitive ratios become stable only when the ratio $\frac{\text{number of jobs}}{m}$ becomes greater that 200.

An important result of the experiments is that although the competitive ratios of the utilization and equal priority completion time are not quantitatively the same, their trends are almost similar for all considered cases. This experimental observation confirms our expectation in the sense that there may exist a close relationship between those criteria.

For larger machine numbers, we turned out from most considered cases that the performance of the online nondelay schedule can be predicted somewhat well from their performance under smaller machine numbers.

As a confirmation of our theoretical results, the worst attainable competitive ratio for utilization was 1.32. This experimental result is almost our bound $\frac{4}{3}$ which we have derived theoretically for such criterion. For equal priority completion time, the worst achievable competitive ratio was 1.165. This result insures and validates again

our theoretical bound 1.25 for such criterion. Moreover, this result even confirms our expectation that the bound $\frac{1+\sqrt{2}}{2}$ is the tight bound of the equal priority completion time for multi-release date case.

Appendix A

Near-Optimal Algorithm

The following is a detailed description of the algorithm which produces schedules with near-optimal solution. The approach of this algorithm is given in Section 5.2.

Algorithm: Near Optimal approach.

Input: Instance for $P_m|r_j|*$, the list L
Output: Optimum utilization U^r and equal priority completion time C^r_{equ}

forall *machines* **do**
  initialize $h(k) \leftarrow 1, A_{h(k)} \leftarrow 0, B_{h(k)} \leftarrow 0$, $l_k \leftarrow 0$, $\sum_2 \leftarrow 0$;
while $L \neq \emptyset$ **do**
  let r be the smallest release date in list L;
  compute $\sum_1 \leftarrow \sum p_j$, $\sum_2 \leftarrow \sum_2 + \sum p_j^2$ for all jobs released at r;
  foreach *machine k* **do**
    if $B_{h(k)} > r$ **then** $\sum_1 = \sum_1 + (B_{h(k)} - r)$;
    else if $B_{h(k)} < r$ **then**
      $h(k) + +$;
      $A_{h(k)} = r$;
  initialize $M \leftarrow m$, $k \leftarrow 1$, $M^* \leftarrow \{M_k | k = 1, \cdots, m\}$, $\sum_{equ} \leftarrow 0$;
  $TargetLine \leftarrow r + (\sum_1 / M)$;
1 **while** $k \leq m$ **do**
    while $M_k \in M^*$ **do**
      if $l_k > r$ **then**
        if $l_k >$*TargetLine* **then**
2 remove M_k from M^*;
          $B_{h(k)} \leftarrow l_k$;
          $\sum_1 \leftarrow \sum_1 - (B_{h(k)} - r)$;
          reduce M by one;
          calculate new: $TargetLine \leftarrow r + (\sum_1 / M)$;
          $k \leftarrow 1$;
          Go To Step 1;
3 $B_{h(k)} \leftarrow TargetLine$;
        increase k by one;
        Go To Step 1;
      else
        pick a first job $next$ from the list L;
        if $(r + p_{next}) >$*TargetLine* **then**
          $l_k \leftarrow r + p_{next}$;
          Go To Step 2;
        Go To Step 3;
    k++;
  foreach *machine k* **do**
    for $i \leftarrow 1$ ***to*** $h(k)$ **do**
      $\sum_{equ} \leftarrow \sum_{equ} + \frac{1}{2}(B_i - A_i)^2 + A_i(B_i - A_i)$;
      if $A_i < r$ **then** $\sum_U \leftarrow \sum_U + min\{r, B_i\} - A_i$;
    $C^r_{equ} \leftarrow \sum_{equ} + \frac{1}{2} \sum_2$;
    $U^r \leftarrow \sum_U$;
    if $B_{h(k)} =$*TargetLine* **then** $l_k = 0$;
  remove the remanning jobs with r from L;

Appendix B

Additional Experimental Results

The following figures represent the experimental results which are obtained to explain the close relation between the behavior of the competitive ratios of the utilization and the corresponding behavior of the competitive ratios of the equal priority completion time.

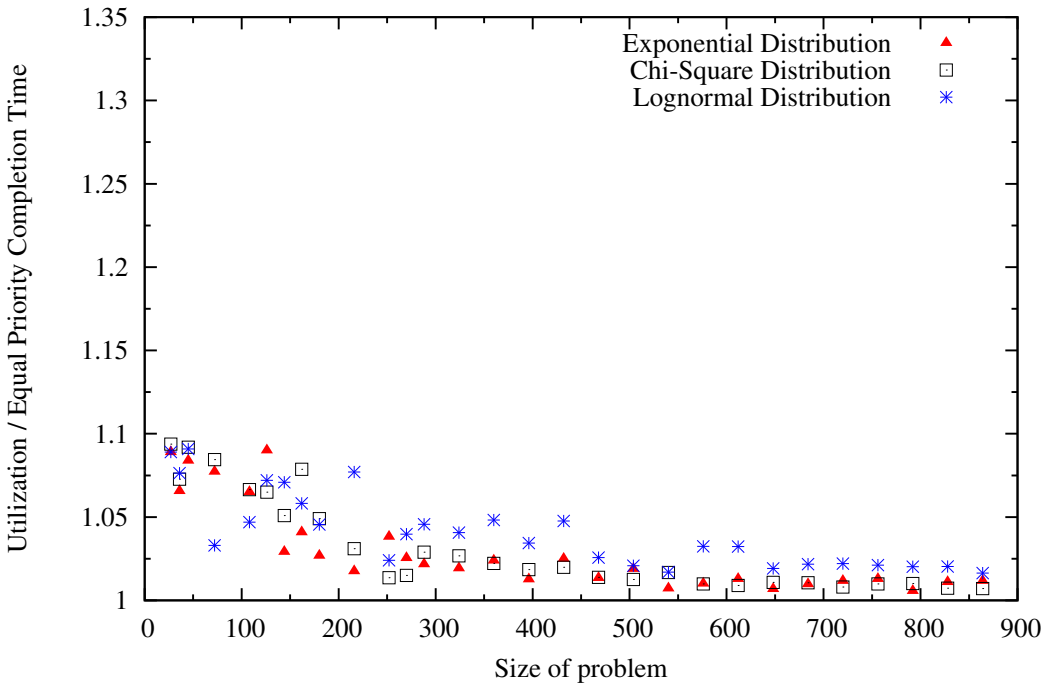

Figure B.1: *The ratios between the competitive ratio of the utilization and the competitive ratio of the equal priority completion time when* $m = 3$.

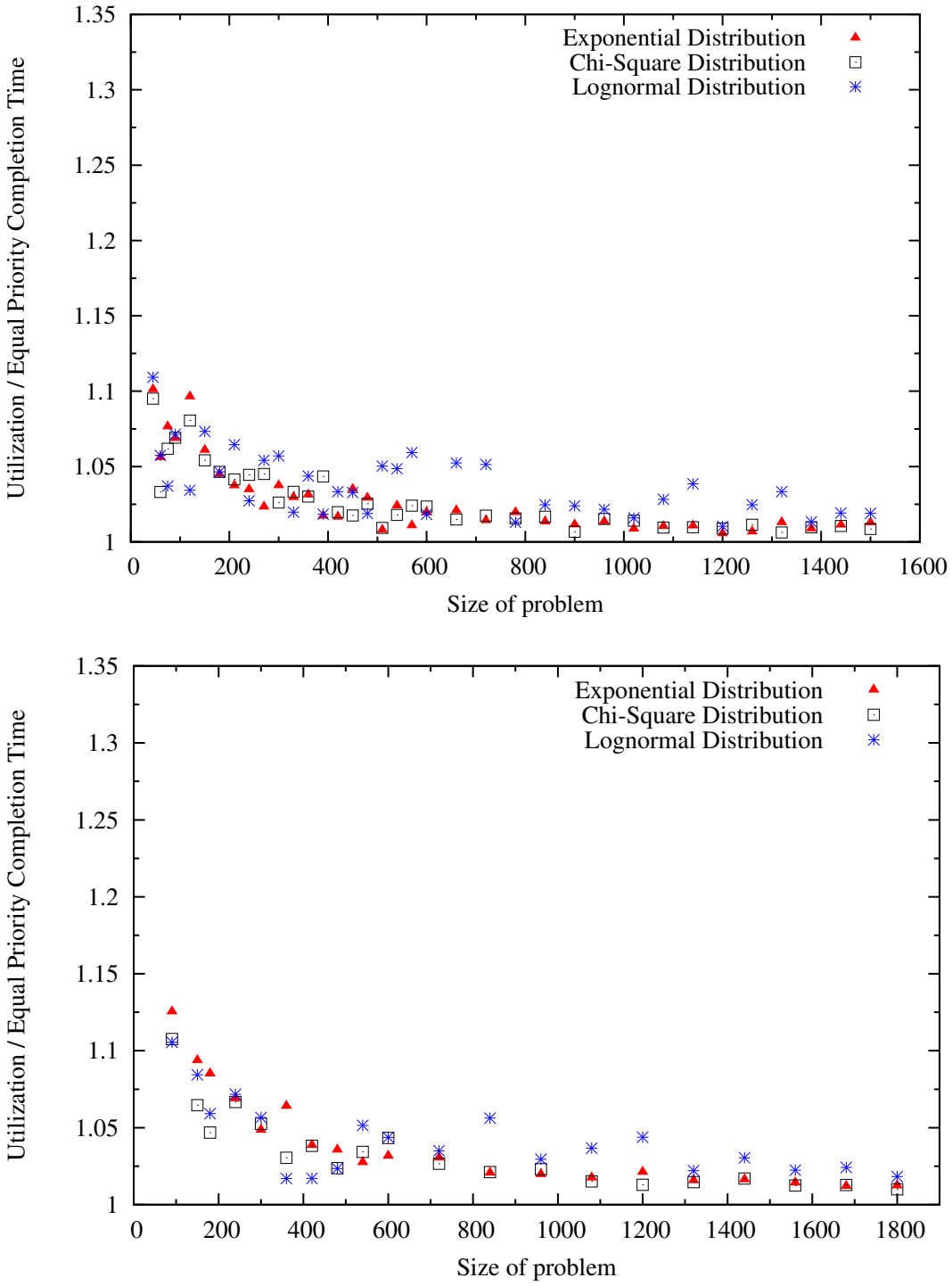

Figure B.2: *The ratios between the competitive ratio of the utilization and the competitive ratio of the equal priority completion time when* $m = 5$ *(top) and* $m = 10$ *(bottom).*

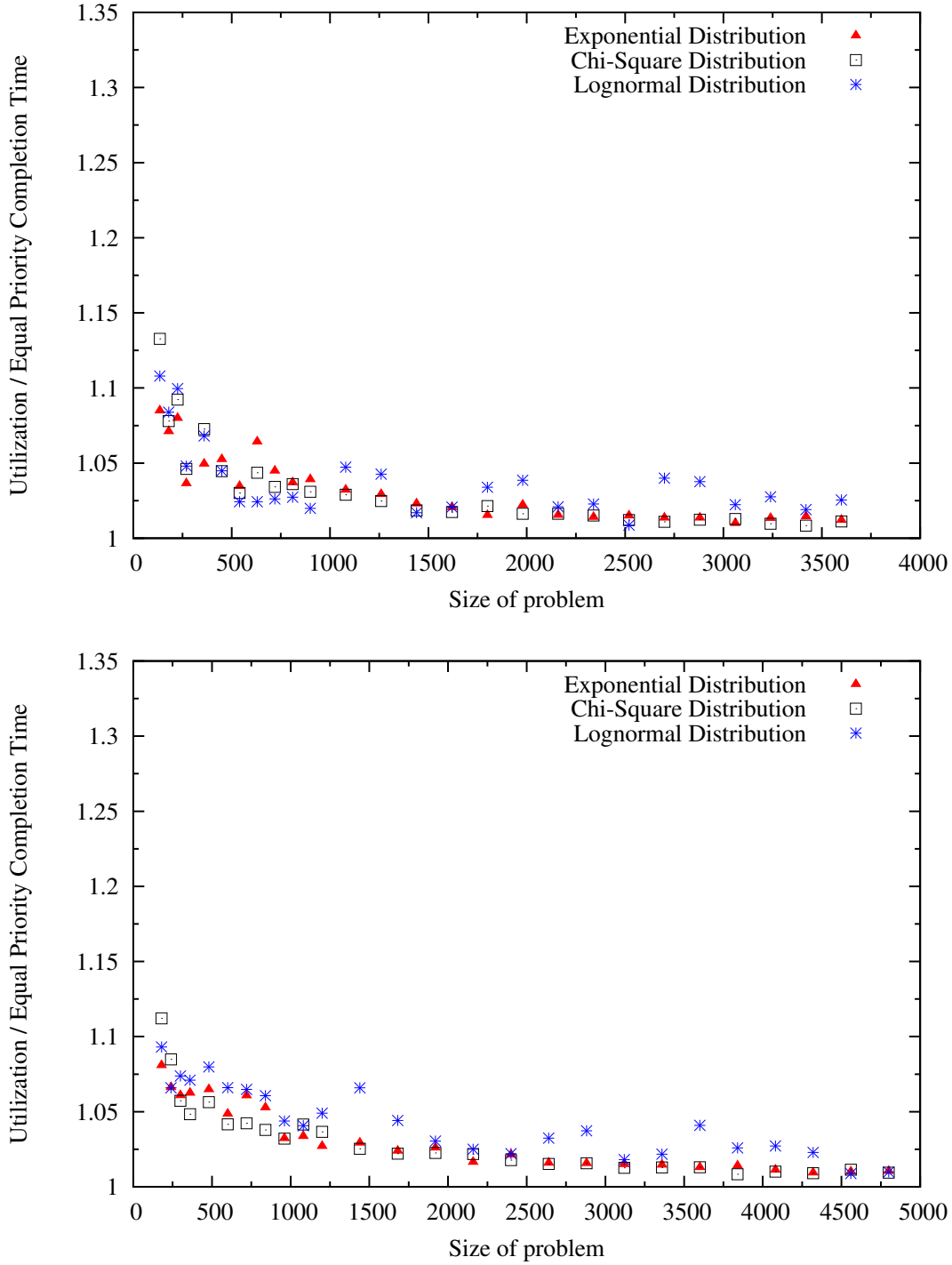

Figure B.3: *The ratios between the competitive ratio of the utilization and the competitive ratio of the equal priority completion time when* $m = 15$ *(top) and* $m = 20$ *(bottom).*

Bibliography

[1] F. Afrati, E. Bampis, C. Chekuri, D. Karger, C. Kenyon, S. Khanna, I. Milis, M. Queyranne, M. Skutella, C. Stein, and M. Sviridenko, *Approximation schemes for minimizing average weighted completion time with release dates*, Proceedings of the 40^{th} Annual IEEE Symposium on Foundations of Computer Science, 1999, pp. 32–43. (Cited on pages 79, 80 and 81.)

[2] S. Albers, *Better bounds for online scheduling*, SIAM Journal on Computing **29** (1999), no. 2, 459–473. (Cited on page 19.)

[3] S. Albers and B. Schröder, *An experimental study of online scheduling algorithms*, ACM Journal of Experimental Algorithms **7** (2002), 3. (Cited on page 117.)

[4] N. Alon, Y. Azar, G. J. Woeginger, and T. Yadid, *Approximation schemes for scheduling on parallel machines*, Journal of Scheduling **1** (1998), 55–66. (Cited on page 79.)

[5] K. R. Baker, *Introduction to Sequencing and Scheduling*, John Wiley & Sons Ltd. New York, 1974. (Cited on page 1.)

[6] Y. Bartal, S. Leonardi, A. Marchetti-Spaccamela, J. Sgall, and L. Stougie, *Multiprocessor scheduling with rejection*, SIAM Journal on Discrete Mathematics **13** (2000), no. 1, 64–78. (Cited on page 21.)

[7] L. Becchetti and S. Leonardi, *Nonclairvoyant scheduling to minimize the total flow time on single and parallel machines*, Journal of the ACM **51** (2004), no. 4, 517–539. (Cited on page 24.)

[8] L. Becchetti, S. Leonardi, A. Marchetti-Spaccamela, and K. Pruhs, *Semi-clairvoyant scheduling*, ESA 2003, LNCS 2832, 2003, pp. 67–77. (Cited on page 20.)

[9] R. Bellman, *Dynamic Programming*, Princeton University Press., 1957. (Cited on page 27.)

[10] M. Bender, S. Muthukrishnan, and R. Rajaraman, *Improved algorithms for stretch scheduling*, ACM/SIMA Symposium on Discrete Algorithms, 2002, pp. 762–771. (Cited on page 20.)

[11] A. Borodin and R. El-Yaniv, *Online Computation and Competitive Analysis*, Cambridge University Press, 1998. (Cited on pages 18, 23 and 24.)

[12] P. Brucker, *Scheduling Algorithms*, Springer Verlag, 2001. (Cited on page 3.)

[13] S. Chakrabarti, C. Phillips, A.S. Schulz, D.B. Shmoys, C. Stein, and J. Wein, *Improved scheduling algorithms for minsum criteria*, Proceedings of the 1996 International Colloquium on Automata, Languages and Programming (F. Meyer auf der Heide and B. Monien, eds.), Springer–Verlag, Lecture Notes in Computer Science LNCS 1099, 1996, pp. 646–657. (Cited on pages 81 and 82.)

[14] C. Chekuri, R. Motwani, B. Natarajan, and C. Stein, *Approximation techniques for average completion time scheduling*, SIAM Journal on Computing **31** (2001), no. 1, 146–166. (Cited on page 82.)

[15] E. Davis and J.M. Jaffe, *Algorithms for scheduling tasks on unrelatead processor*, Journal of the ACM **28** (1981), 721–736. (Cited on page 28.)

[16] M. A. Dempster, J. K. Lenstra, and A. Rinnooy Kan, *Deterministic and stochastic scheduling*, Kluwer Academic Publishers, 1982. (Cited on page 27.)

[17] L. W. Dowdy, E. Rosti, G. Serazzi, and E. Smirni, *Scheduling issues in high-performance computing*, ACM SIGMETRICS Performance Evaluation Review, New York, USA, March 1999, pp. 60–69. (Cited on page 55.)

[18] A. Fiat and editors G. J. Woeginger, *Online algorithms: The State of the Art*, Springer, 1998. (Cited on page 18.)

[19] S. French, *Sequencing and scheduling*, Ellis Horwood Limited, 1982. (Cited on page 3.)

[20] M. Garey and D. Johnson, *Computers and intractability: A guide to the theory of np-completeness*, W. H. Freeman, San Francisco, 1979. (Cited on pages 4 and 79.)

[21] E. Gelembe, V. Srinivasan, S. Seshadri, and N. Gautam, *Optimal policies for ATM-cell scheduling and rejection*, Telecommunication System **18** (2001), no. 4, 331–358. (Cited on page 21.)

[22] P. C. Gilmore and R. E. Gomory, *A linear programming approach to cutting stock problem*, Operations Research **9** (1961), 848–859. (Cited on page 27.)

[23] R. L. Graham, *Bounds for certain multiprocessor anomalies*, Bell System Technical Journal **45** (1966), 1563–1581. (Cited on page 27.)

[24] R.L. Graham, *Bounds on multiprocessor timing anomalies*, SIAM Journal of Applied Mathematics **17** (1969), 416–429. (Cited on page 28.)

[25] 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** (1979), 287–326. (Cited on page 11.)

[26] B. Das Gupta and M. A. Palis, *Online real-time preemptive scheduling of jobs with deadlines on multiple machines*, Journal of Scheduling **4** (2001), 297–312. (Cited on page 21.)

[27] S. K. Gupta and J. Kyparisis, *Single machine scheduling research*, OMEGA International Journal of Managemant Science **15** (1987), 207–227. (Cited on page 12.)

[28] L. Hall, D. Shmoys, and J. Wein, *Scheduling to minimize average completion time: Off-line and on-line algorithms*, Proceedings of the 7^{th} SIAM Symposium on Discrete Algorithms, January 1996, pp. 142–151. (Cited on page 80.)

[29] 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 **22** (1997), 513–544. (Cited on pages 79, 80 and 81.)

[30] L.A. Hall and D.B. Shmoys, *Approximation schemes for constrained scheduling problems*, Processing of the 30^{th} Ann. IEEE Symp. on Foundations of Computer Sci., 1989, pp. 134–139. (Cited on page 28.)

[31] M. Held and R. M. Karp, *A dynamic programming approach to sequencing problems*, J. SIAM **10** (1962), 196–210. (Cited on page 27.)

[32] D. S. Hochbaum, *Approximation algorithms for np–hard problems*, PWS Publishing Company, 1995. (Cited on page 5.)

[33] K.S. Hong and J.Y.-T. Leung, *On-line scheduling of real-time tasks*, IEEE Transations on Computing **41** (1992), 1326–1331. (Cited on page 28.)

[34] J. R. Jackson, *Scheduling a production line to minimize maximum tardiness*, Mgmt Sci. Res. Project, UCLA (1955). (Cited on page 26.)

[35] D. S. Johnson, *Approximation algorithms for combinatorial problems*, Journal of Computer and System Sciences **9** (1974), 256–278. (Cited on page 5.)

[36] S. M. Johnson, *Optimal two- and three-stage production schedules with setup times included*, Naval Research Logistics Quarterly **1** (1954), 61–67. (Cited on page 26.)

[37] B. Kalyanasundaram and K. Pruhs, *Speed is as powerful as clairvoyance*, Journal of the ACM **47** (2000), no. 4, 217–243. (Cited on page 25.)

[38] A. H. G. Rinnooy Kan, *Machine scheduling problems*, Martinus Nijhoff, The Hague, 1976. (Cited on page 34.)

[39] H. D. Karatza, *A simulation based performance analysis of scheduling in a parallel systems*, Proceedings of the 12^{th} European Simulation Symposium and Exhibition, 2000, pp. 582–586. (Cited on page 55.)

[40] D. Karger, S. J. Phillips, and E. Torng, *A better algorithm for an ancient scheduling problem*, J. Algorithms **20** (1996), no. 2, 400–430. (Cited on page 19.)

[41] A. Karlin, M. Manasse, L. Rudolph, , and D.D. Sleator, *Competitive snoopy caching*, Algorithmica **3** (1988), 79–119. (Cited on page 22.)

[42] R. M. Karp, *Reducibility among combinatorial problem, in: Complexity of computer computations*, R. E. Miller and J. W. Thatcher (eds.), Plenum Press, New York, 1972. (Cited on pages 4 and 27.)

[43] ———, *On the computational complexity of combinatorial problems*, Networks **5** (1975), 45–68. (Cited on page 27.)

[44] T. Kawaguchi and S. Kyan, *Worst case bound of an LRF schedule for the mean weighted flow-time problem*, SIAM Journal on Computing **15** (1986), no. 4, 1119–1129. (Cited on pages 10, 38, 39, 79, 80, 86, 87, 94 and 95.)

[45] K. Kempf, R. Uzsoy, and S. Smith ans K. Gary, *Evalution and comparison of production schedules*, Computer in Industry **42** (2000), 203–220. (Cited on pages 7, 38 and 60.)

[46] J. Labetoulle, E.L. Lawler, J.K. Lenstra, and A.H.G Rinnooy Kan, *Preemptive scheduling of uniform machines subject to release dates, in: W.R. Pullybank (ed.)*, Progress in Combinatorial Optimization, Academic Press, Toronto, 1984, pp. 245–261. (Cited on page 21.)

[47] E. L. Lawer and J. M. Moore, *On functional equation and its application to resource allocation and sequencing problem*, Mgmt. Sci. **16** (1969), 77–84. (Cited on page 27.)

[48] E. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D. Shmoys, *Sequencing and scheduling: Algorithms and complexity*, Handbook of Operations Research and Management Science, vol. 4, Elsevier Science Publishers, 1993, pp. 445–522. (Cited on page 78.)

[49] E.L. Lawler, *Recent results in the theory of machine scheduling*, A. Bachem, M. Grötschel, and B. Korte (eds.), Mathematical Programming: Bonn 1982. The State of the Art, Berlin: Springer, 1982, pp. 202–234. (Cited on page 5.)

[50] C. B. Lee, Y. Schwartzman, J. Hardy, and A. Snavely, *Are user runtime estimates inherently inaccurate?*, Proceedings of the 10^{th} Workshop on Job Scheduling Strategies for Parallel Processing, June 2004, pp. 153–161. (Cited on pages 6 and 58.)

[51] J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker, *Complexity of machine scheduling problems*, Annals of Discrete Mathematics **1** (1977), 343–362. (Cited on pages 5 and 79.)

[52] X. Lu, R. A. Sitters, and L. Stougie, *A class of on-line scheduling algorithms to minimize total completion time*, Operations Research Letters **31** (2003), 232–236. (Cited on page 82.)

[53] M. Mastrolilli, *Scheduling to minimize max flow time: Offline and online algorithms*, International Journal of Foundations and Computer Science **15** (2004), no. 2, 385–401. (Cited on pages 17 and 34.)

[54] N. Megow and A. S. Schulz, *Scheduling to minimize average completion time revisited*, Operations Research Letters **32** (2004), 485–490. (Cited on page 82.)

[55] N. Megow, M. Uetz, and T. Vredeveld, *Stochastic Online Scheduling on Parallel Machines*, To appear in Proceedings of 2^{nd} Workshop on Approximation and Online Algorithms (WAOA), September 2004. (Cited on page 83.)

[56] R. H. Möhring and F. J. Radermacher, *An introduction to stochastic scheduling problems*, In Contributions to Operations Research (Berlin) (K. Neumann and D. Pallachke (eds.), eds.), Springer–Verlag, Lecture Notes in Economics and Mathematical Systems 240, 1985, pp. 72–130. (Cited on page 27.)

[57] R. H. Möhring, A. S. Schulz, and M. Uetz, *Approximation in stochastic scheduling: The power of LP-based priority policies*, Journal of the ACM **46** (1999), no. 6, 924–942. (Cited on pages 82 and 83.)

[58] R. Motwani, S. Phillips, and E. Torng, *Non-clairvoyant scheduling*, Theoretical Computer Science **130** (1994), no. 1, 17–47. (Cited on pages 20 and 57.)

[59] A. W. Mu'alesm and D. G. Feitelson, *Utilization, predictability, workloads, and user runtime estimates in scheduling ibm sp2 with backfilling*, IEEE Trans. Parallel & Distributed Syst. **12** (2001), no. 6, 529–543. (Cited on page 58.)

[60] A. Munier, M. Queyranne, and A. Schulz, *Approximation bounds for a general class of precedence constrained parallel machines scheduling problems*, In Proceedings of IPCO VI, Springer–Verlag Lecture Notes in Computer Science LNCS 1412, 1998, pp. 367–382. (Cited on pages 7 and 28.)

[61] H. V. D. Parunak, *Characterizing the manufacturing scheduling problem*, Journal of manufacturing systems **10** (1991), no. 3, 241–258. (Cited on pages 4 and 5.)

[62] C. Philips, C. Stein, and J. Wein, *Scheduling jobs that arrive over time*, Proceedings of the 4^{th} International Workshop on Algorithms and Data Structures, Lecture Notes in Computer Science, LNCS 955, 1995, pp. 290–301. (Cited on page 80.)

[63] C. Phillips, C. Stein, E. Torng, and J. Wein, *Optimal time-critical scheduling via resource augmentation*, Algorithmica (2002), 163–200. (Cited on page 25.)

[64] C. Phillips, C. Stein, and J. Wein, *Minimizing average completion time in the presence of release dates*, Mathematical Programming **82** (1998), 199–223. (Cited on page 82.)

[65] M. Pinedo, *Scheduling: Theory, algorithms, and systems*, first ed., Prentice-Hall, Englewood Cliffs, NJ, 1995. (Cited on pages 12 and 57.)

[66] ———, *Scheduling: Theory, algorithms, and systems*, second ed., Prentice-Hall, New Jersey, 2002. (Cited on pages 3, 7, 12, 17, 30 and 34.)

[67] R. Righter, *Stochastic scheduling*, In Stochastic Orders (M. Shahed and G. Shanthikumar (eds.), eds.), Academic Press, San Diego, 1994. (Cited on page 27.)

[68] S. Sahni and Y. Cho, *Nearly on line scheduling of a uniform processor system with release times*, SIAM Journal on Computing **8** (1979), 275–285. (Cited on page 28.)

[69] A. S. Schulz and M. Skutella, *Scheduling unrelated machines by randomized rounding*, SIAM Journal on Discrete Mathematics **15** (2002), no. 4, 450–469. (Cited on pages 81 and 83.)

[70] U. Schwiegelshohn and R. Yahyapour, *Fairness in parallel job scheduling*, Journal of Scheduling **3** (2000), no. 5, 297–320. (Cited on page 38.)

[71] S. Seiden, *Barely random algorithms for multiprocessor scheduling*, Journal of Scheduling **6** (2003), 309–334. (Cited on page 19.)

[72] J. Sgall, *On-Line Scheduling–A Survey*, In Online Algorithms: The State the Art (A. Fiat and G.J. Woeginger(eds), eds.), Springer–Verlag, Lecture Notes in Computer Science vol. 1442, 1998, pp. 196–231. (Cited on page 18.)

[73] D. Shmoys, J. Wein, and D. Williamson, *Scheduling parallel machines on-line*, SIAM Journal on Computing **24** (1995), no. 6, 1313–1331. (Cited on pages 29 and 36.)

[74] M. Skutella and G. J. Woeginger, *A ptas for minimizing the total weighted completion time on identical parallel machines*, Mathematics of Operations Research **25** (2000), no. 1, 63–75. (Cited on page 80.)

[75] D.D. Sleator and R.E. Tarjan, *Amortized efficiency of list update and paging rules*, Communications of the ACM **28** (1985), no. 2, 202–308. (Cited on page 22.)

[76] W. Smith, *Various optimizers for single-stage production*, Naval Research Logistics Quarterly **3** (1956), 59–66. (Cited on pages 26, 38, 77 and 87.)

[77] A. Souza and A. Steger, *The expected competitive ratio for weighted completion time scheduling*, 21^{st} Annual Symposium on Theoretical Aspects of Computer Science

(V. Diekert and M. Habib, eds.), Springer–Verlag, Lecture Notes in Computer Science LNCS 2996, 2004, pp. 620–631. (Cited on page 83.)

[78] D. J. White, *Dynamic programming*, Oliver and Boyd, Edinburgh, 1969. (Cited on page 27.)