Design and Evaluation of Algorithms for Online Machine Scheduling Problems

Ming Liu

ÉCOLE CENTRALE DES ARTS ET MANUFACTURES « ÉCOLE CENTRALE PARIS »

THÈSE

présentée par

Ming LIU

pour l'obtention du

GRADE DE DOCTEUR

Spécialité : Génie Industriel

Laboratoire d'accueil : Laboratoire Génie Industriel

SUJET :

Design and Evaluation of Algorithms for Online Machine Scheduling Problems

soutenue le : 24 Septembre 2009

devant un jury composé de :

CHU, Chengbin	**Directeur de thèse**
BAPTISTE, Philippe	**Rapporteur**
MARTINEAU, Patrick	**Rapporteur**
SOURD, Francis	**Examinateur**
XU, Yinfeng	**Examinateur**
DU, Dingzhu	**Examinateur**

Contents

List of Figures

Acknowledgements

Thanks to the jury members who would like to spend time on evaluation of my work and participation in my defense.

Nearly three years ago, I started my master under the supervision of Prof. Yinfeng XU. He guided me to a new area: online algorithms. At that time, this area was brand new to me. After a while, an opportunity appeared in front of me. With good luck, I also became a Ph.D. student of Prof. Chengbin CHU who is an expert in the area of scheduling. Due to the cooperation of my two excellent supervisors, my research domain has been determined coincidentally to be online scheduling.

Although only my name appears on the cover of this thesis, I am indebted to many people who contributed to my work. Therefore, I would like to express my gratitude towards them.

First of all, I thank Chengbin CHU and Yinfeng XU for giving me all the opportunities and help to successfully complete this thesis. Prof. CHU always gave me some insightful ideas to break through the difficulties. This resulted in my publications. Other than that, he carefully reads drafts of my work and corrected my faults succinctly. I will never forget his efforts to improve my work to be conspicuous.

Secondly, I would like to thank Feifeng ZHENG. The majority of my publications benefits from the cooperation with him. I also thank him for his many suggestions to improve both the style of my writing and the layout of my publication.

Next, I express my gratitude to Zhiyi TAN. He cleared my doubt about some algorithms. I am also grateful to Yiwei JIANG. With his help, I could not waste my time on some obtained results. Also, I could entirely understand some algorithms with his detailed explanation.

Furthermore, I thank my family and friends for their continuing support. I would like to thank my girlfriend, Lu WANG. She constantly encouraged me to pursuit my research when I face some difficulties.

At last, thanks to all laboratory colleagues who accompany me during my stay in LGI. Many thanks!

Abstract

This thesis proposes and evaluates some online algorithms for machine scheduling problems. Deterministic scheduling models have been extensively studied in the literature. One of the basic assumptions of these models is that all the information is known in advance. However, this assumption is usually not realistic. This observation promotes the emergence of online scheduling. In online scheduling problems, an online algorithm has to make decisions without future information. Competitive analysis is a method invented for analyzing online algorithms, in which the performance of an online algorithm (which must satisfy an unpredictable sequence of requests, completing each request without being able to see the future) is compared with the performance of an a posteriori optimal solution where the sequence of requests is known. In the framework of competitive analysis, the performance of an online algorithm is measured by its competitive ratio.

We mainly deal with two online paradigms: the one where jobs arrive over list and the one where jobs arrive over time. Based on these two paradigms, we consider different models: single machine, two identical parallel machines, two uniform parallel machines, batch processing machine and open shop. For each of the problems, we prove a lower bound of competitive ratios and propose online algorithms. Then we further measure the worst case performance of these algorithms. For some problems, we can show that the algorithms we proposed are optimal in the sense that their competitive ratios match the lower bounds.

Chapter 1

Introduction

This thesis is concerned with design and evaluation of algorithms for online machine scheduling problems. This chapter presents scheduling problems, shows the motivations and the relevance of this research, and summarizes the contributions of the thesis.

1.1 Scheduling

Scheduling deals with the allocation of scarce resources to activities (or tasks) over time. It is a decision-making process to optimize one or more objective functions while satisfying some constraints.

The resources and activities in an organization can take many forms. The resources may be database servers in a network, machines in a workshop, crews at a construction site, runways at an airport, CPU in a computer, and so on. The activities may be data in a network, operations in a production process, stages in a construction project, take-offs and landings at an airport, executions of computer programs, and so on. Each activities may have some parameters, such as a certain priority level, a weight, a grade of service request (GoS), a release time, a due date, and so on. The objectives can also take many forms, such as the minimization of makespan, the minimization of total completion time, the minimization of total tardiness, the maximization of early jobs, and so on.

Scheduling plays an important role in many manufacturing and production systems as well as in many information-processing environments. In transportation and distribution environments, and other types of service industries, the role of scheduling is also indispensable.

1.2 Motivations

Traditional scheduling theory assumes 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 lack of information (or knowledge). For example, no one knows the exact number of phone calls that are going to reach a switchboard 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. Based on these scenarios in reality, it is known that uncertainties frequently encountered in scheduling environments include the appearance of new jobs and unknown processing times. These reasons promote the emergence of online scheduling. Similar to classical scheduling problems, online scheduling problems also originate from allocating certain scarce resources to activities, but over time (or over list), without having full knowledge of the future.

The notion of online algorithm is intended to formalize the realistic scenario where the algorithm does not have access to the whole input instance, unlike the (offline) algorithms for classical scheduling problems. Instead, it learns the input piece by piece, and has to react to the new requests with only a partial knowledge of the input. Its applications can be found in many areas as scheduling. This thesis is devoted to such online scheduling problems with the aim of attaining the highest resource utilization.

For instance, in real world, the following online scenario may happen in electronic commerce if no a couple of jobs are allowed to share a 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 customers job always uses its assigned machine exclusively for its whole processing time (i.e., 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 process their jobs. Forcing the customers to provide the processing times of their jobs in advance may be a hassle to those customers and is very unreliable.

It is reasonable to assume that the system owner primarily tries to maximize system utilization. The system owner has no information about any future request. Therefore, without additional knowledge on the jobs, no job selection strategy can guarantee a better schedule than any other. The system owner may use a strategy by immediately assigning a free machine to an open request, that is, he generates a greedy schedule. On the other hand, he postpones the assignment of a job to a machine until a machine becomes available (for feasibility). If several requests are open at the same time he may use any arbitrary policy to pick any one of them. In such a situation, he is using an online algorithm and more specifically he uses a list scheduling algorithm, which is a classical algorithm in online scheduling.

1.3 Our contributions

In this thesis, we address some online scheduling problems by designing and analyzing some algorithms. In online scheduling, any algorithm is a heuristic. The performance of such heuristics can be evaluated either by average performance or by worst-cast performance (the evaluation is done by comparing the obtained solution with offline optimal solution. The average performance evaluation needs assumptions on probability distribution of various parameters, such as arrival interval or processing times. Furthermore, the structure of the distribution function must be special to obtain analytical expressions of expected performance of both obtained solutions and optimal offline solutions.

Because of these restrictions, we focus on worst-case analysis, i.e., we provide

a performance guarantee of algorithms. This thesis considers a wide range of problems. for each problem, we try to establish a lower bound of the worst-case performance and attempt to design an algorithm such that the worst-case performance is as good as possible (as close as possible to the lower bound).

The next paragraphs summarize the main features of the problems considered in the thesis and illustrate the relevance of these problems. In this thesis, we consider several scheduling models: single machine model, parallel machine model, batch processing machine model, and open shop model. Different constraints are investigated, such as bounded delivery times, grade of service (GoS), reassignment, bounded processing times and so on.

Delivery time has a great significance in modern logistics. In order to attain a high integration, decision-makers usually consider jobs' delivery times into scheduling problems. Thus, we first study an online scheduling problem with jobs' delivery times. As we all know, each job's processing time can not be infinite. We consider the case where each job's delivery time in not too larger comparing to its processing time. In this scenario, we design an optimal online algorithm.

Grade of Service (GoS) provision is an attracting research field, since it can equip manufactures to guarantee high quality products for certain customers. As the reason mentioned above, we deal with the variant with jobs' bounded processing times. Under certain conditions, we respectively give optimal algorithms for different ranges (or intervals) of jobs' processing times.

With the technological development, machine's renovation happens more and more frequently in factories. An effect of such development is reduction of time (or cost) caused by processing the task, in other words, machine's speed has been increased. Machines with different speeds may run at the same time, i.e., the scenario of uniform parallel machines. So we draw some results for two uniform machine environment. After that, we further consider the problem with reassignment, which allows us to reassign jobs in some conditions. Such research is motivated by realistic phenomenon. For example, on the assembly line, a time delay is existing between assignment and process. In the process of dealing with all the jobs assigned, some newly arrived jobs may not have been processed yet. In order to minimize the completion time, we need to reassign these jobs to gain a better effect. In other cases, such as hotel or restaurant reservation and reception, we can do some adjustment in order to gain more profits. For the two uniform machine environment with reassignment, we design and analyze some algorithms.

Everyone has a common experience: in school homework must be handed in by a given due date. Similarly, many other assignments in the business and public worlds have dates by which the task must be completed and returned to the person who assigned the task, their due dates. Therefore, in this thesis, we also consider due date constraint in online scheduling to maximize the number of early jobs. We show the performance of an classical algorithm, called SRPT or ECT, for preemption scenario.

Another kind of machine environment is batch processing machine. Batch processing machine scheduling has been motivated by burn-in operations in the final testing stage of semiconductor manufacturing. We consider two variants on a single batch processing machine with jobs' nondecreasing processing times and jobs' nonincreasing processing times, respectively. For these two special cases, we respectively develop optimal algorithms.

Open shop scheduling problems arise in several industrial situations. For example, consider a large aircraft garage with specialized work-centers. An airplane may require repairs on its engine and electrical circuit system. These two tasks may be carried out in any order but it is not possible to do these tasks on the same plane simultaneously. There are other applications of open shop scheduling problems in automobile repair, quality control centers, semiconductor manufacturing, teacher-class assignments, examination scheduling, and satellite communications. For two open machine environment with preemption, considering jobs' processing times, we obtain an optimal algorithm.

1.4 Outline of the thesis

This thesis focuses on the theory of scheduling, and deals with the detailed sequencing and scheduling of jobs. It is divided into several parts.

We first discuss some models and results on online scheduling in Chapter 2.

In Chapter 3, we consider online scheduling on a single machine with bounded delivery times. This chapter is mainly based on [70]. Jobs arrive over time. The objective of this problem is to minimize the makespan considering jobs' delivery times. Note that makespan means the last moment by which all jobs have been delivered to the customers. Based on the results of literature, we further consider the situation where jobs' delivery times are bounded. For this problem, we give an optimal online algorithm, which is a generalization of the results in the literature.

In Chapter 4, we deal with online scheduling on two identical parallel machines with grade of service provision (GoS). This chapter is mainly based on [69]. The problem under consideration is to minimize the makespan. Jobs arrive over list. By bounding jobs' processing times, we divide the problem into different subproblems. We analyze different subproblems and present some optimal online algorithms under certain conditions. Further more, we suppose that the total processing time is known. Under this assumption, we give an optimal algorithm under some conditions.

Chapter 5 addresses the problem: online scheduling on two uniform (parallel) machines to minimize the makespan. This chapter is mainly based on [74]. The problems considered are: online scheduling under GoS and online scheduling with reassignment. Jobs arrive over list. For the first problem, we obtain an optimal algorithm under certain conditions. For the second problem, we give some lower bounds and upper bounds.

In Chapter 6, we study the problem of online scheduling on m identical parallel machines. The objective is to maximize the number of early jobs. This chapter is mainly on the basis of [73]. The problem is online in the sense that all jobs arrive over time. Each job's characteristics, such as processing time and due date, become known at its arrival time. We consider the *preemption-restart model*, which means that preemption is allowed and once a job is restarted it loses all the progress that has been made on this job so far. If in some schedule a job is completed before or at its due date, then it is called *early* (or *on time*). We show that an upper bound of competitive ratio is $1 - \frac{1}{2m}$ and prove that ECT (earliest completion time) algorithm is $\frac{1}{2}$-competitive.

In Chapter 7, we consider two semi-online scheduling problems on a single batch (processing) machine with jobs' nondecreasing processing times and jobs'

nonincreasing processing times, respectively. This chapter is mainly based on [71]. The objective is to minimize the makespan. A batch processing machine can handle up to B jobs simultaneously. We study an unbounded model where $B = \infty$. The jobs that are processed together construct a batch, and all jobs in a batch start and complete at the same time. The processing time of a batch is given by the longest processing time of any job in the batch. Jobs arrive over time. Let p_j denote the processing time of job J_j. Given job J_j and its following job J_{j+1}, we assume that $p_{j+1} \geq \alpha p_j$, where $\alpha \geq 1$ is a constant number, for the first problem with jobs' nondecreasing processing times. For the second problem, we assume that $p_{j+1} \leq \alpha p_j$, where $0 < \alpha < 1$ is a constant number. We propose an optimal algorithm for both problems.

In Chapter 8, we deal with a two-machine open shop problem. This chapter is mainly based on [68]. The objective is to minimize the makespan. Jobs arrive over time. We study preemption-resume model, i.e., the currently processed job may be preempted at any moment in time, and it may be resumed at any later moment. Let $p_{1,j}$ and $p_{2,j}$ denote the processing time of job J_j on machines M_1 and M_2, respectively. Bounded processing times mean that $1 \leq p_{i,j} \leq \alpha$ $(i = 1, 2)$ for each job J_j, where $\alpha \geq 1$ is a constant number. We obtain an optimal online algorithm.

Finally, in Chapter 9, we give some conclusions from this research and dicuss some further research directions.

Chapter 2

State of the art

In this chapter, we review the literature on machine scheduling. Scheduling problems can be distinguished into two categories: offline scheduling and online scheduling. In offline scheduling models, it is assumed that the characteristics of the activities and/or the resources are known in advance when establishing the planning. As a consequence, offline models are deterministic models.

In practice, it is not always possible to have all the data concerning the jobs and the resources. To deal with this situation, various models are developed, especially stochastic scheduling models and online models.

In stochastic models, the unknown data are represented by random variables with known probability distribution. Among stochastic scheduling models, queuing theory is widely used.

In contrast to offline deterministic models and stochastic models, online scheduling make very few assumptions about the characteristics of jobs. This thesis is focused on online scheduling models, that is why the remainder of the chapter is mainly devoted to the literature review of online scheduling models.

2.1 Scheduling models

The models in machine scheduling problems are highly standardized. In all the scheduling problems, the number of jobs and machines are assumed to be finite. The number of jobs (or tasks/activities) is denoted by n and the number of machines (or resources) by m. It is generally assumed that each machine can process no more than one job at a time, except when the machines are batch precessing machines. The aim is to optimize some objectives. Usually, the subscript j refers to a job, whereas the subscript i refers to a machine. J_j denotes a job and M_i denotes a machine. If a job requires a number of processing operations, then $J_{i,j}$ denotes the processing operation of job J_j on machine M_i. Some parameters (or pieces of data) are associated with job J_j.

Release time (r_j): the time when the job J_j arrives at the system, i.e., the earliest time at which job J_j can be processed. Release time is also called *release date*, *arrival date* or *arrival time*.

Processing time ($p_{i,j}$): the processing time of job J_j on machine M_i. If the J_j's processing times on different machines are identical, we use p_j instead of $p_{i,j}$ to denote the processing time of J_j.

Due date (d_j): the committed completion date, i.e., the date the job is promised to the customer.

Delivery time (q_j): job J_j needs to be delivered after its processing on the machine, which takes a certain time. This time is called delivery time.

Weight (w_j): the importance of job J_j relative to the other jobs in the system.

2.1.1 Variety of scheduling problems

A scheduling problem is described by a triplet $\alpha|\beta|\gamma$. The α field describes the machine environment and usually contains a single entry. The β field provides details of processing characteristics and constraints and may contain no entry at all or multiple entries. The γ field describes the objective to be minimized (or maximized) and usually contains a single entry. The specification of a machine scheduling model requires the description of a machine environment (α field), job characteristics (β field), and an optimality criterion (γ field).

Some possible machine environments specified in the α field are:

Single machine (1): There is only one machine. A single machine is the simplest of all possible machine environments.

Identical parallel machines (Pm): There are m identical machines in parallel. Job J_j requires a single operation and can be processed on any one of the m parallel machines.

Uniform parallel machines (Qm): There are m machines in parallel with different speeds. The speed of machine M_i is denoted by s_i. The processing time $p_{i,j}$ that job J_j spends on machine M_i is equal to p_j/s_i (assuming job J_j is not preempted). If all the machines have the same speed, then this environment becomes identical parallel machines.

Unrelated parallel machines (Rm): This environment is a generalization of the previous one. There are m parallel machines. Machine M_i can process job J_j at speed $s_{i,j}$. The time $p_{i,j}$ that job J_j spends on machine M_i is equal to $p_j/s_{i,j}$ (if this job is fully processed on machine M_i).

Open shop (O_m): There are m machines. Each job has to be processed again on each one of the m machines. There are no restrictions with regard to the routing of each job through the machine environment. The decision maker is allowed to determine a route for each job.

Flow shop (Fm): There are m machines in series. Each job has to be processed on each one of the m machines. All jobs have to follow the same route. After completion on one machine, a job joins the queue at the next machine. If all queues operate under the *First In First Out (FIFO)* discipline, the flow shop is referred to as a *permutation* flow shop.

Flexible flow shop (FFc): A flexible flow shop is a generalization of the flow shop and the parallel machine environments. Instead of m machines in series, there are c stage in series with a number of identical machines in parallel at each stage. Each job has to be processed first at Stage 1, then Stage 2, and so on. A stage functions as a bank of parallel machines; at each stage, job J_j requires processing on only one machine and any machine can satisfy. The queue at a stage may or may not operate according to the *First Come first Served (FCFS)* discipline. This machine environment is also referred to as a *hybrid flow shop*, *compound flow shop*, or *multiprocessor flow shop*.

Job shop (Jm): In a job shop with m machines, each job has its own predetermined route to follow.

Flexible job shop (FJc): A flexible job shop is a generalization of the job shop and the parallel machine environments. Instead of m machines, there are c work centers with a number of identical machines in parallel at each work center. Each job has its own route to go through the shop. Job J_j requires processing at each work center on only one machine and any machine can satisfy. If a job on its route through the shop may visit a work center more than once, then the β-field contains the entry *recrc* for the reason of recirculation.

The processing constraints are specified in the β field. These restrictions may be:

Release times (r_j): job J_j can not be processed before this time.

Preemptions (*pmpt*): The scheduler is allowed to interrupt the processing of a job at any point in time and put a different job on the machine instead. The preempted job may be continued at some time later. When a preempted job is afterward put back on the machine (or another one), it may need be processed for its remaining processing time or for its (original) processing time. The former case is called *resume*, and the latter case is called *restart*.

Precedence constraints (*prec*): Precedence constraints may appear in a single machine or in a parallel machine environment, requiring that one or more jobs may have to be completed before another job is allowed to start its processing. There are several special forms of precedence constraints: If each job has at most one predecessor and at most one successor, the constraints are referred to as *chains*. If each job has at most one successor, the constraints are referred to as an *intree* or "join" type. If each job has at most one predecessor, the constraints are referred to as an *outtree* or "fork" type.

Grade of Service (*GoS*): Grade of service (GoS) is a qualitative concept, and it's often translated into the level of access privilege of different service provision. For example, suppose we have 2 machines (or processors). One of them can provide high quality service (or high GoS) while the other one provides normal service (or low GoS). Some jobs which request high quality must be processed by high GoS machine, while other jobs with low quality requests can be processed by both machines. GoS is also called *hierarchy*.

Availability constraint (h): There is non-availability interval $[s, t]$ on each of the machines. During this period, the machine cannot process any job. There are two models divided by how to handle the *crossover* job, which is the job affected by the non-availability interval i.e., the jobs starting beforing the unavailability period but having to be completed afterward. The first is non-resumable model where the crossover job that cannot be completed by time s is restarted from scratch at time t. Under the resumable model, the crossover job is interrupted at time s and resumed from the point of interruption at time t.

Machine eligibility restrictions (M_j): The M_j symbol may appear in the β field when the machine environment is m machines in parallel. When the M_j present, not all m machines are capable of processing job J_j. The set M_j denotes the set of machines that can process job J_j.

Permutation (*prmu*): A constraint that may appear in the flow shop environment is that the queues in front of each machine operate according to the *First In First Out (FIFO)* discipline. This implies that the order in which the jobs go through the first machine is maintained throughout the system.

Blocking (*block*): Blocking may occur in flow shop. If a flow shop has a

limited buffer between two successive machines, then it may happen that when the buffer is full the upstream machine is not allowed to release a completed job.

No-wait (*nwt*): The *nwt* symbol may appear in flow shop. Jobs are not allowed to wait between two successive machines. This implies that the starting time of a job at the first machine has to be delayed to ensure that the job can go through the flow shop without having to wait for any machine. It is clear that under *nwt* constraint, the machines also operate under the *FIFO* discipline.

Recirculation (*recrc*): *recrc* may occur in a job shop or flexible job shop when a job may visit a machine or work center more than once.

Online environment (*online*): This constraint means that decision must be made without full knowledge of jobs' information (or with partial jobs' information). Jobs arrive one by one.

Nonincreasing processing times (*nonincreasing*): The *nonincreasing* symbol may appear in the β field when the machine environment is *online*. When this item is presented, jobs are sorted in the order of nonincreasing processing times.

Nondecreasing processing times (*nondecreasing*): This symbol is defined in the opposite way of the previous one.

Any other entry that appear in the β field is self-explanatory. For example, $p_j = p$ implies that all processing times are equal, and similarly $d_j = d$ means that all due dates are equal (also called *common due date*). If $p_j \leq p_{j+1}$ appears in β field, the processing time of job J_j is not greater than that of its following job.

In order to present some possible entries in the γ field to be minimized, we first give some basic functions.

The objective function is usually a function the completion times of the jobs. The time job J_j exits the machine system is denoted by C_j, i.e., its completion time on the last machine on which it requires processing. Given a due date d_j of job J_j, the ***lateness*** of J_j is defined as

$$L_j = C_j - d_j,$$

which is positive when J_j is completed late and negative when it completed early. The ***tardiness*** of J_j is defined as

$$T_j = \max\{L_j, 0\}.$$

The latter notation is more reasonable since the tardiness never is negative. The ***unit penalty*** of J_j is defined as

$$U_j = \begin{cases} 1 & \text{if } C_j > d_j \\ 0 & \text{otherwise} \end{cases}.$$

Some commonly discussed objective functions are:

Makespan (C_{max}): The completion time of the last job completed in the system. A minimum makespan usually signifies a high utilization of the machines, or a high throughput.

Total completion time ($\sum C_j$): The sum of the completion times of all jobs. This criterion indicates the total holding or inventory cost incurred by the schedule.

Total weighted completion time ($\sum w_j C_j$): The sum of the weighted completion times of all jobs.

Total weighted tardiness ($\sum w_j T_j$): This is a more general cost function than the total weighted completion time.

Weighted number of tardy jobs($\sum w_j U_j$): This objective has both academic and practical values.

All the objective functions listed above are called **regular** performance measures. Regular performance measures are functions that are nondecreasing in $C_1, ..., C_n$.

2.2 Online paradigms

In reality, it is unlikely to have all the information necessary to define a problem instance beforehand. This reason prompted the emergence of research on online models. These models differ from each other with respect to the way in which the algorithm has the information, step by step. Sgall distinguished several paradigms in an excellent survey [93]. Note that the first two models are commonly studied in the literature [87] and will be the main focus of this thesis.

Jobs arrive over list (or in a list): In this model, the jobs are ordered in some list and are presented one by one to the algorithm. When a job appears, all its characteristics, such as its processing time, become known. The online algorithm must assign the job immediately to a time slot and a machine before the next job is seen, consistently with some restrictions given by the problem, such as GoS. We are allowed to assign the jobs to arbitrary time intervals, i.e., they can be delayed. This assignment is irrevocable, i.e., the algorithm cannot change it after it sees the next job in the list.

This model corresponds to the standard online model of request sequences. It is used for problems such as bin packing, bin stretching, load balancing, graph coloring, and paging. There is no release time in this model, the only online feature is the limited information of future requests.

Jobs arrive over time: Jobs become available to the algorithm over time at their release time. When a job arrives, all its characteristics, such as processing time, due date, delivery time, are known by the algorithm. At each time when the machine is idle, the online algorithm decides which one of the available jobs is scheduled, if any. The online features of this model are the lack of knowledge of future requests and of their arrival time. If preemptions or restarts are allowed, the online algorithm can decide to preempt or stop any job that is currently being processed.

Unknown processing times: The main online feature in this model is that the processing time of a job is unknown until the job finishes. The online algorithm only knows whether a job is still being processed or not. The jobs may become available over time according to their release times or precedence constraints, but the case where all jobs are available at the beginning plays an important role in this model. If there are other characteristics of a job other than its processing time, they are known when the job becomes available.

Interval scheduling: This model assumes that each job has to be executed in a precisely given time interval. If this is impossible it may be rejected. This scenario is very different from the previous three. For example, it is meaningless

to measure the makespan or total completion time of the schedule; instead we measure the number of accepted jobs. It is also related to load balancing.

If an online algorithm is ***clairvoyant***, the processing time of a job becomes known at its arrival. In contrast, if an online algorithm is ***non-clairvoyant***, the processing time of each job is still unknown when the job is available; as long as the job is not finished, a non-clairvoyant online algorithm only knows whether it is still being processed or not.

Other than in offline scheduling, the option to preempt a job can be very important in the online paradigms where jobs arrive over time. It enables the scheduler to correct mistakes in a limited range. Restarts have the same effect.

In this thesis, we only consider two online models: the one where jobs arrive over list and the one where jobs arrive over time. Most of the time, we focus on clairvoyant algorithms.

2.3 Performance measures and computation

For a scheduling problem, we need an approach to measure the performance of each algorithm for its solution. In this thesis, we use the *competitive analysis* [12] to measure the performance of online algorithms. Precisely, we measure the quality of an online algorithm by its ***competitive ratio***, which is an upper bound of the ratio of the objective function value given by the considered algorithm over the optimal one obtained by offline optimal algorithm (which has access to all information in advance).

In order to evaluate a competitive ratio, the online algorithm is compared to an *adversary*. The adversary makes a sequence of requests, which have to be processed by the online algorithm. The adversary itself also has to serve the requests, but not necessarily at the same time as it creates them. A good adversary tries to develop a sequence of request such that the cost for processing the sequence by the online algorithm is as high as possible compared to its own cost for serving the same sequence of requests.

There are three different kinds of adversaries introduced in [12]. The difference concerns the information about the online algorithm that is available to the adversary and how the adversary processes the requests.

Oblivious adversary: He constructs the request sequence in advance, based only on the description of the online algorithm, and processes the requests himself optimally. This adversary knows the algorithm's code, but does not get to know the randomized results of the algorithm.

Adaptive online adversary: He makes the next request based on the algorithm's answers to previous ones, but processes it immediately. This adversary must make its own decision before it is allowed to know the decision of the algorithm. This adversary is stronger than the previous one, since he has opportunity to change the request sequence.

Adaptive offline adversary: He makes the next request based on the algorithm's answers to previous ones and processes them optimally in the end. This adversary knows everything, even the random number generator. This adversary is so strong that randomization does not help against him.

The competitive ratio of an online algorithm is defined as follows. An online algorithm is called ρ-***competitive*** if for every possible request sequence generated by the adversary, the objective value of the schedule obtained by the online

algorithm is at most ρ times the value of the schedule produced by the adversary plus a constant. If we allow *randomization*, then we compare the *expected* objective values, where the expectation is taken over the random choices of the algorithm.

Due to lack of information in online scheduling, the quality of any online schedule is normally not as good as an (offline) optimal schedule. Specifically, for any input job sequence I, let $C_{ON}(I)$ denote the objective value of the schedule produced by the online algorithm $\mathcal{A}_{ON}$ and $C_{OPT}(I)$ denote the objective value of an (offline) optimal schedule. We say that $\mathcal{A}_{ON}$ is ρ-competitive if

$$C_{ON}(I) \leq \rho C_{OPT}(I) + k$$

where k is a constant number. This inequality is a general form. Usually, we omit the parameter k, i.e., $k = 0$. We also say that ρ is the competitive ratio of $\mathcal{A}_{ON}$. In other words, we measure the quality of an online algorithm $\mathcal{A}_{ON}$, one for the online scheduling problem, by its competitive ratio ρ, which is defined to be (at least) the supremum of ratio $C_{ON}(I)/C_{OPT}(I)$ over each of the problem instances I.

In other words, if for each input instance the (expected) objective value of the schedule produced by this algorithm is at most ρ times the value of the optimal schedule plus a constant, then the deterministic (randomized) algorithm has a competitive ratio ρ (also called upper bound). This corresponds to the competitive ratio against an oblivious adversary. For a specific algorithm, we can give a special instance to make the algorithm perform the worst possible (i.e., the worst case). In this way, we obtain a lower bound (of competitive ratio) of this algorithm. For this algorithm, if its competitive ratio matches it lower bound, we say that this bound is tight.

If a part of information is known in advance, such as the total processing time or the biggest processing time, the problem is called ***semi-online***. When all information is available at the beginning (before scheduling), the problem is called ***offline***.

Note that the above definition of competitive ratio is specified for minimization problems and $\rho \geq 1$. In maximization problems, for any input job sequence I, we say that $\mathcal{A}_{ON}$ is ρ-competitive if

$$C_{ON}(I) \geq \rho C_{OPT}(I).$$

We also say that ρ is the competitive ratio of $\mathcal{A}_{ON}$. Clearly, $0 \leq \rho \leq 1$. The closer the ratio ρ comes to 1, the better the performance of the online algorithm $\mathcal{A}_{ON}$ is.

Now the question is: "without future information, how well an online algorithm can reach?". This promotes the research of the (general) lower bound of competitive ratios for all online algorithms (for minimization problems). For a certain minimization problem, a ***lower bound*** of competitive ratios (also called a lower bound of the problem) implies that there exists no online algorithm with a competitive ratio less than this lower bound. Correspondingly, for a maximization problem, an upper bound of competitive ratios (or the problem) means that no online algorithm has a competitive ratio higher than it. This kind of upper bound is also referred to as a lower bound in the literature for consistency. We use a minimization problem to explain how to draw a lower bound. Usually, a lower bound of the problem is obtained by giving a specific

job instance (sometimes, a set of job instances) on which no online algorithm can performance very well. In this special instance, we first give several jobs in order to detect online algorithms' decisions. Depending on their decisions, we further give jobs to make their decisions the worst possible. If after giving these two sets of jobs, some online algorithms do not achieve their worst cases, the third set of jobs arrive in the system. With the purpose to make any online algorithm the worst possible, we carefully design the special job instance step by step. At last, we will obtain a common value which can not be beaten by any online algorithm. A fundamental principle to beat greedy algorithms is to release jobs immediately after the algorithms schedule the previous jobs.

An online algorithm is considered to be ***best possible*** or *optimal* if its competitive ratio matches the lower bound of competitive ratios (or the problem). In this situation, we can also say that the bound is tight.

Readers should note that the difference between lower bound of the algorithm and the lower bound of competitive ratios. The lower bound of an algorithm means that for an online problem, this specific algorithm cannot have a competitive ratio less than this lower bound. It is associated with a specific algorithm. However, the lower bound of the problem is not specific for any algorithm, but related to the problem. For an online problem, it is the lower bound of all algorithms' competitive ratios. The lower bound of the algorithm is worth to research only in the case where the lower bound of the problem and the competitive ratio of this algorithm do not match. The lower bound of the algorithm can be obtained by designing a special instance for this specific algorithm such that the algorithm performs as bas as possible.

There is also a relative concept, called upper bound of the problem. For an online problem, an upper bound is determined by (or equal to) the competitive ratio of an online algorithm which is currently known to be the best. When the upper bound is equal to the lower bound of the problem, the problem is solved. Usually, there is a gap between two bounds for an online problem. Our task is to improve two bounds to decrease gap. For an online problem, if the upper bound obtained by an algorithm is tight and there is also a gap between this upper bound and the lower bound of the problem, the task is to seek new algorithms with lower competitive ratio (or upper bound), or to improve the lower bound of competitive ratio for the problem.

In order to prove the competitive ratio of an online algorithm, we usually use a technique, called ***minimum counter example*** or *smallest counter example*. Suppose that we want to prove that algorithm $\mathcal{A}_{on}$ is ϕ-competitive. We first assume that there exists a smallest counter example $\mathcal{I}$ with the least jobs such that $A_{on}(\mathcal{I}) > \phi A_{opt}(\mathcal{I})$. Then based on this assumption, using the features of the algorithm, we derive a contradiction. So we claim that such smallest counter example does not exist. Therefore, we have the desired result.

Since this thesis is mainly concerned with the deterministic online algorithms, two adaptive adversaries are equivalent. In this thesis, we consider the competitive ratio against an adaptive (online) adversary. Competitive analysis is a *worst-case analysis* which differs from an *average-case analysis*. This measurement gives a kind of pessimistic impression.

2.4 Modeling online scheduling problems

The basic situation of online scheduling problems is the following. We have some sequence of jobs that have to be processed on the machines. In the most basic problem, each job is characterized by its processing time and has to be assigned on one of the machines. Jobs arrive with some online features. In some complex variants, there may be additional restrictions or relaxations. We usually want to schedule the jobs as efficiently as possible.

The concept of ***online algorithm*** is used to formalize the realistic scenario, where the algorithm does not have the information of the whole input instance. In contrast, the offline algorithms are assumed to have access to the whole input instance. Online algorithms learn the input piece by piece, and has to react to the new jobs with only a partial knowledge of the input.

When the theory of NP-completeness was developed, many scheduling problems have been shown to be NP-complete [28]. After the completeness of 18 basic scheduling problems had been shown [37], the focus shifted to designing approximation algorithms. Online algorithms are basically kinds of approximation algorithms and many natural heuristics for scheduling are in fact online algorithms.

In order to paraphrased the online models in a realistic way, we adopt some examples illustrated in [108].

Example 1: A queue of cars is standing in front of a ferry-boat waiting to be transported to the other shore. The ferry man that is standing in front of this queue is given the task to assign each of the cars to a location on the ferry-boat so as to minimize the free area of the ship. Because the view of the ferry man is partially blocked, he can only see the first car in the queue at a time. Once this car has been assigned to a position on the ship, the cars move up and he sees the next car in the queue. Due to his years of experience, the ferry man knows exactly the space needed by a car once he sees it. For obvious reasons, the assignment of a car cannot be changed once the decision has been made. To avoid chaos, the assignment of the cars to the deck of the ferry-boat is highly structured. The deck of the ferry-boat is divided into parallel lanes of equal width and equal length. Each lane is wide enough to be able to contain any of the cars. Cars can only be assigned to the prescribed lanes and must be located one after another in each of the lanes.

We can formulate this problem in terms of the online models mentioned in the above section. Look for the machine environment (α field), job characteristics (β field) and an optimality criterion (γ field). The jobs in this example are the cars, and the processing time of a job is the length which the car requires. The machines are the lanes into which the deck is partitioned. Every job can be assigned to any lane and uses an amount of space that is independent of the lane. This means that the environment corresponds to a set of identical parallel machines. The objective is to minimize the free area, which is equivalent to minimizing the total length of the cars left ashore. Define a due date and a weight for each job to be equal to the length of the lanes and the length of the car it represents, respectively. The objective hence is to minimize the weighted number of tardy jobs. The online model is the one where jobs arrive over list.

Example 2: A factory faces a continuous demand for cardboard boxes of various types. The boxes are produced by a group of machines. Every machine can make a complete box from a single cardboard sheet. This includes printing

in multi-colors, cutting, folding, and gluing. Because the machines were bought over the years, they all have different processing speeds. Every order consists of a specified number of boxes of a certain type. All orders have a due date, which is the promised date of delivery. The decision makers have to decide how to assign the orders to the machines so as to minimize the number of late orders.

Similar to the analysis of previous example, we should look for the machine environment, job characteristics and an optimality criterion. The jobs in this example are the orders that have to be produced, and the machines are simply the machines in the factory. The objective is to minimize the number of tardy jobs. Every job can be processed by processed by each of the machines. Considering the fact that all machines have different speeds, this implies that the machine environment corresponds to a set of uniform parallel machines. The online model is the one where jobs arrive over time.

Example 3: A computer network of several computers is used for transmitting messages between the users of the network. Each computer has many users who want to transmit messages via this network. Before transmitting a message, a route is chosen along which the transmission should take place. Each of the links in the route has a fixed bandwidth, i.e., the number of bits per second that they can transmit is fixed. The requests for sending a message arrive online over time at the computers, and each message has a size in bits, which is unknown at the time of the request. No computer in the route can forward a message until it has been received completely. Given the transmission route, the aim is to find a protocol that regulates the communication traffic in such a way that the average waiting time is minimized.

The jobs in this example are the messages. The links between the computers can be viewed as machines. All messages are traveling via a route that is known in advance, and the order in which the machines are visited is known beforehand. The objective is to minimize the average waiting time. This is the same as minimizing the total waiting time. By adding all jobs' processing times (transmission times in the network without waiting), the objective function becomes the minimization of total completion time. The problem therefore can be formulated as a job shop to minimize the total completion time. The online feature that is used is obviously the model where jobs arrive over time and only non-clairvoyant algorithms have to be considered.

2.5 Literature review of online scheduling

There are hundreds of online scheduling results associated with different online paradigms in the literature, but our work is with regard to two kinds of online paradigms: jobs arrive over list, and jobs arrive over time. Since our research does not involve randomization, we mainly present the results of deterministic algorithms. In order to organize these results, we respectively state them in different online paradigms.

Before categorizing, we show some general results. The first one is LIST algorithm, which was given by Graham [40]. He studied a simple greedy algorithm.

He investigated a basic model, where we have m identical machines and a sequence of jobs characterized by their processing times. The objective is to minimize the makespan. We assume that all jobs are arranged in a list according

to their release times, even if some of them do not arrive. Whenever a machine is available, LIST algorithm schedules the first job in the list on this machine. The list is formed online (no matter jobs arrive over time or over list). Since LIST algorithm does not use the information of processing times, so it also hold the online feature of unknown running times. Therefore, LIST algorithm works in the first three online paradigms.

Graham showed that the competitive ratio of LIST is $2-\frac{1}{m}$. In the following paper [41], Graham showed that the factor of 2 decreases if we modify the algorithm so that some number of long jobs is scheduled first using an optimal schedule, and the rest is scheduled by LIST algorithm. Clearly, this algorithm is no longer online in any of the paradigms. However, this gives an intuition to design approximation algorithms [62] and online algorithma [23].

2.5.1 Jobs arrive over list

This online model corresponds most closely to the standard model of request sequence in competitive analysis. In this paradigm, there is no release times and precedence constraints, since these restrictions appear to be unnatural with scheduling jobs in a list. In most variants, it is not necessary to introduce idle time on any machine.

Minimizing makespan

The basic model is that: we have m machines in parallel and a sequence of jobs characterized by their processing times. The jobs arrive one by one, and we have to schedule each job before we see the next one. The objective is to minimize the makespan. Each job is assigned to a single machine. Each machine can process at most one job at a time. There is no additional constraints. Preemption is not allowed, and all the machines are identical.

LIST algorithm [40] is $(2-\frac{1}{m})$-competitive. This algorithm is optimal for $m = 2, 3$, but for larger m we should develop optimal algorithms [33]. We know that there is a weakness of LIST algorithm. If currently all machines have equal loads and a job with long processing time arrives, LIST algorithm obtains a schedule which is almost twice as long as the optimal one. So here is the problem: if the scheduled jobs are sufficiently small, and the optimal schedule consists of scheduling them evenly on $m-1$ machines in parallel with the last long job on the remaining machine. Observing this fact, to obtain a better scheme, we have to create some imbalance and keep some machines lightly loaded (for the use by long jobs).

To attain a competitive ratio σ, a natural idea is to assign the current job to any machine such that immediately after this assignment the competitive ratio is at most σ. In order to avoid the weakness of LIST algorithm, suppose we always choose the most loaded of the machines to create as large imbalance as possible. Nevertheless, it does not work. If this algorithm faces a sequence of jobs with identical processing times, it assigns them evenly on a constant fraction of the machines, with only one job scheduled on each of the remaining machines. Now a sequence of long jobs presents. The algorithm first makes the load distributed evenly on all machines, then has to force the schedule to be too long. Thus, this natural method fails again.

To design a good algorithm, two different approaches are commonly used. One is to schedule each job on one of the two currently least loaded machines [36, 19]. This method yields better results than LIST for any $m \geq 4$ and achieves the currently best upper bounds for small m. However, the competitive ratio approaches 2 when m is large. This approach leaves at most one lightly loaded machine. In order to obtain a better upper bound for large m, it is necessary to keep some constant fraction of machines lightly loaded. Based on this idea, several algorithms were developed [9, 59, 1].

We know that LIST algorithm always assigns the incoming job to the least loaded machine. Based on this algorithm, Cho and Shani [24] further studied the two-uniform-machine environment (where the speeds of machines are 1 and $s \geq 1$ respectively) and proved that LIST algorithm is $\frac{1+\sqrt{5}}{2}$-competitive for all s and the bound is tight when $s = \frac{1+\sqrt{5}}{2}$. Epstein et al. [30] provided randomized algorithms with better performance than LIST algorithm in the case where no preemption is allowed. In the same paper, for the problem with preemption, they proposed an optimal $(1 + \frac{s}{s^2+s+1})$-competitive algorithm for all $s \geq 1$, which cannot be beaten by any randomized algorithm [109]. Angelelli et al. [3] considered the semi-online scheduling on two uniform processors in the case where the total processing time of jobs is known in advance and presented algorithms which are optimal for $s \geq \sqrt{3}, s = 1$ and $\frac{1+\sqrt{17}}{4} \leq s \leq \frac{1+\sqrt{3}}{2}$. In that paper, they first showed that the LIST algorithm does not improve its performance when the total sum of the tasks is fixed with respect to (pure) online problem, and then they presented three different algorithms that are optimal in some intervals of the values of the speed s. The idea of these three algorithms are simple. First assign the current job to a machine and when the load of that machine exceeds a value (or function of s), then schedule this job on a specified machine. These three algorithms differ one from another each other with regard to the function of s and the specified machine.

Minimizing makespan with GoS

GoS is a qualitative concept, and it is often translated into the level of access privilege of different service provision. One simple scheme for providing differentiated service is to label machines and jobs with pre-specified grade of service (or GoS) levels and allow each job to be processed by a particular machine only when the GoS level of the job is no less than the GoS level of the machine. In effect, the processing capability of the machines labeled with high GoS levels tends to be reserved for the jobs with high GoS levels. Hence, if we label relatively higher GoS levels on the jobs from valued customers, we can ensure better service to more valued customers. In such a situation, assigning jobs to the machines becomes a parallel machine scheduling problem with a special eligibility constraint. For example, suppose that we have two machines (or processors). One of them can provide higher service quality (higher GoS) while the other one is normal (lower GoS). Some jobs which are requested to have higher quality must be processed by the higher GoS machine. While the normal jobs can be processed by both machines whenever they are available. Within an offline context, the concept of GoS was introduced and analyzed in [55]. They proposed a simple algorithm, called lowest grade-longest processing times first (LG-LPT) whose computational complexity is strongly polynomial

and the worst-case makespan of the generated schedule is no more than twice the optimum. In particular, the makespan of the schedule generated by the proposed algorithm is proven to be no more than 5/4 for $m = 2$ and $2 - 1/(m-1)$ for $m \geq 3$ times the optimum. The idea of algorithm LG-LPT comes from a classical algorithm LPT for problem $Pm||C_{max}$. Considering the GoS contraint, they modified the LPT algorithm by sorting the job sequence in a order of increasing GoS where ties are broken by LPT rule. In other words, LG-LPT algorithm assigns the jobs in the order of increasing GoS and when several jobs have the same GoS, they are scheduled by LPT rule.

Park et al. [82] studied the online scheduling of two machines under a grade of service (GoS) provision and its semi-online variant where the total processing time is known. They gave an optimal online algorithm whose competitive ratio is $\frac{5}{3}$ and an optimal semi-online algorithm with a competitive ratio $\frac{3}{2}$. The idea of online algorithm they proposed is try to assign jobs with lower GoS as many as possible on lower-GoS machine. To be more specific, when a higher-GoS job arrives, we have to assign it to the higher-GoS machine. When a lower-GoS job is presented, we need to determine which machine should process this job. If the load of lower-GoS machine (after assigning the current lower-GoS job on that machine) is not bigger than a certain value (or a function of previous jobs' processing times), then we do schedule lower-GoS job on lower-GoS machine. Otherwise, schedule it on the higher-GoS machine. But the criterion (or function of previous jobs' processing times) is a little difficult to design. In order to prove the online algorithm is σ-competitive, they first showed a lower bound (also a function of previous jobs' processing times) of optimal value. The first principle is to assign lower-GoS jobs as many as possible on lower-GoS machines. The second principle is to prevent the makespan from exceeding σ times the optimal value (after scheduling the currently by the first principle). Thus, if this the makespan is bigger than the criterion, the algorithm has to assign this job on a higher-GoS machine. Based on the idea of this online algorithm, by slightly changing the lower bound of optimal value with the information of total processing times, Park et al. [82] proposed an optimal semi-online algorithm in the same paper.

Jiang [57] extensively investigated the problem of online scheduling on parallel machines with two GoS levels. He assumed that the number of machines providing high GoS is not known before scheduling and decisions must be made without knowledge of the exact number of machines providing higher GoS. In other words, we only know that in all 10 parallel machines there are k ($1 \leq k \leq 9$) machines which can provide higher GoS. Under this consideration, he proved that 2 is a lower bound of online algorithms and proposed an online algorithm with a competitive ratio of $\frac{12+4\sqrt{2}}{7}$. The online algorithm they designed is with many details. The general idea is also to use some criterion to distinguish different situations. Some criteria are chosen in a similar way as that in [82].

To the best of our knowledge, the most recent results with GoS constraint are the research of online scheduling on two uniform machines [74, 67].

Minimizing makespan plus penalties

In this variant, jobs may be rejected at a certain penalty. Each job is characterized by the processing time and the penalty. A job can either be rejected

by paying its penalty or scheduled on one of the machines. The objective is to minimize the makespan of the schedule for accepted jobs plus the sum of the penalties of all rejected jobs. The purpose of an online algorithm is to balance the penalties and the increase in the makespan. (It is a little puzzling to add two different quantities together as one objective function. Given two weights on two kinds of quantities, this objective can be viewed as a multiple-objective decision.)

This problem of Multiprocessor Scheduling with Rejection (MSR) was introduced by Bartal et al. [10]. The main goal of an algorithm designed for online MSR problem is to choose the proper balance between the penalties yielded by rejecting jobs and the increase in the makespan for the accepted jobs. At the beginning it might have to reject some jobs if the penalty for their rejections is small compared to their processing times. However, at a certain point, it would be better to schedule some of the previously rejected jobs since the increase in the makespan due to scheduling those jobs in parallel is less than the total penalty incurred. In this scenario, the online MSR problem can be seen as a nontrivial generalization of the well known *Rudolph's ski rental* problem. (In that problem, a skier has to choose whether to rent skis for the cost of 1 per trip, or to buy them for the cost of c without knowing the future number of trips. The best possible deterministic strategy is to rent for the first c trips and buy afterwards. In online MSR problem, rejecting jobs is analogous to renting while scheduling one job is analogous to buying.) They showed an $1 + \phi \approx 2.618$-competitive algorithm, where ϕ is the golden ratio. Their algorithm used two simple rules. First, all jobs in the set B are rejected, where B denote the set of jobs such that each jobs' penalty is smaller than its load. The second rule that a job is rejected unless its penalty added to the total penalty of the hitherto rejected jobs would be higher than some prescribed fraction of its processing time. This rule is inspired by the relation of MSR to the ski-rental problem. The algorithms of Bartal et al. [10] consist of two parts, a rejection scheme, which decides which jobs are rejected, and a scheduling algorithm, which assigns accepted jobs to a machine. The rejection schemes are combined with Graham's LIST algorithm.

If preemption is allowed, Seiden [92] gave an $\frac{4+\sqrt{10}}{3} \approx 2.38743$-competitive algorithm and showed a lower bound of 2.12457. The algorithms they proposed also consist of two parts, a rejection scheme and a scheduling algorithm. In fact, the rejection schemes given in [10] can be combined with any scheduling algorithm. The general idea in [92] is that by combining the rejection schemes with algorithms for different scheduling models, Seiden get new algorithms for scheduling with rejection.

Gyorgy an Imreh [42] investigated a new scheduling model where the number of machines is not fixed, the algorithm has to purchase the used machines, and the jobs can be rejected. They presented an algorithm OPTCOPY with a competitive ratio 2.618. The basic idea behind this algorithm is to consider a relaxed version where we replace part of the cost of the schedule (purchasing cost of machines plus the makespan) with a lower bound of it. Imreh [56] further considered the problem with general machine cost functions.

Online scheduling of open shop scheduling problem was studied in [21, 18]. There are not many results in this research field, and all this kind of research concentrates on looking for permutation algorithms, where the schedule has the

same job sequence on each of the machines.

Machine covering problems

The machine covering problem deals with partitioning a sequence of jobs among a set of machines, so as to maximize the completion time of the least loaded machine, where the load of a machine is the total time required to complete all jobs assigned to it. Machine covering problem is also called the *Santa Claus* problem, where n indivisible goods are to be partitioned among m clients. The goal is to distribute the goods in a way that the least satisfied client is still as pleased as possible. The offline problem is strongly NP-hard. The online machine covering is associated with the online paradigm where jobs arrive one by one, since there is no need to refer to release times. The objective of online machine covering problems is to maximize the minimum load of any machine (denoted by C_{min}).

Maximizing the minimum machine completion time is NP-complete in the strong sense. Deuermeyer et al. [11] examined the behavior of the Longest Processing Time (LPT) rule to approximate the optimum solution. The LPT rule sorts all jobs into a sequence of nonincreasing processing times and then sequentially assigns each job to the next machine available. They showed that the minimum completion time in the schedule produced by LPT is at least 3/4 times the minimum completion time in the optimum schedule. Csirik et al. [25] tightened the analysis of LPT by showing that the LPT remains within a factor of $(3m-1)/(4m-2)$ of the optimum solution and that this bound is tight. Observe that in the online paradigm where jobs arrive list, LIST algorithm behaves like LPT, but starts with an unsorted list (partial unknown). Thus, in the research of online machine covering, LIST and LPT are the same.

For m identical machine environment, Woeginger [110] analyzed an simple online algorithm LPT which assigns a new job to the least loaded machine and showed that LPT is best possible with a competitive ratio m. Obviously, LPT is a kind of greedy algorithm.

After that, some scholars tried to overcome high competitive ratio by using additional information, i.e., they studied semi-online variants. Azar and Regev [6] considered those problems whose optimal objective function value is known in advance (denoted by *opt*). Kellerer et al. [60] considered the problem that the total processing time of all the jobs is known in advance (denoted by *sum*). He and Zhang [46] considered the problem that the largest processing time of all jobs is known in advance (denoted by *max*). In the same paper, they also considered the problem that the processing time of jobs are tightly-grouped or bounded in a interval (denoted by *tightly-grouped*). Seiden et al. [91] considered the information that jobs are arriving in nonincreasing processing time order (denoted by *decr*). For problem $Pm|online, tightly-grouped|C_{min}$, He Yong [44] proved that LIST algorithm is still optimal.

Azar and Epstein [4] proposed an algorithm FILL for $Pm|opt|C_{min}$ with a competitive ratio $2-1/m$, and it is optimal for $m = 2, 3, 4$. In contrast to LIST algorithm which tries to assign jobs to all the machines evenly, FILL works as follows (The optimal solution is 1.):
"*If there are no empty machines, assign a new job to the least loaded machine. Otherwise, if the processing time of the new job is at least* $m/(2m-1)$*, assign it to an empty machine. If the processing time of the job is less than* $m/(2m-1)$

assign it to the active machine if exists, and otherwise to an empty machine."

At first, the FILL algorithm tries to keep a certain load difference between all empty machines and non empty machines. If there exist no empty machine, FILL algorithms works as LIST.

Tan and Wu [102] investigated the semi-online machine covering problems on $m \geq 3$ parallel identical machines. Three different semi-online versions are studied and optimal algorithms are proposed in that paper. They proved that if the total processing time of all jobs or the largest processing time of all jobs is known in advance, the competitive ratios of the optimal algorithms are both $m-1$. If the total processing time and the largest processing time of all jobs are both known in advance, the competitive ratios of the optimal algorithms are $3/2$ when $m = 3$, and $m-2$ when $m \geq 4$. Generally speaking, all these three semi-online algorithms are designed based on LIST algorithm.

Another part of semi-online research concentrates on two uniform machines. Epstein [29] analyzed LPT algorithm with the knowledge of optimal value on two uniform machines, where the speeds of two machines are s and 1 respectively. He showed that LPT is $s+1$-competitive. In the same machine environment, the case where total processing time is known in advance was studied in [99]. Cao and Tan [15] considered the case where the size of the largest job is declared in advance.

Minimizing makespan with parallel jobs

First note that parallel jobs are also called multi-processor jobs. In practice (i.e., in parallel supercomputers) some jobs can only be processed on several processors in parallel. The online problem of parallel job scheduling can be described as follows. A job $J_j = (s, p)$ is associated with two parameters p and s, where p is the processing time of the job and s is the width (or number of machines) required for simultaneous processing. The jobs arrive over list. Preemption and reassignment are not allowed. The goal is to minimize the makespan.

It has been shown that LIST algorithm is best possible in the online models: the one where jobs arrive over time and the one with unknown processing times [94].

In general, if a job can request any number of available machines, Johannes [58] presented an online algorithm with competitive ratio 12. She also proved that no online algorithm is better than 2.25-competitive. Johannes' algorithm consists of two parts: partition part and schedule part. In partition part, the time axis is divided into different length intervals with such that the length is doubled at each time (i.e., intervals $I_i := [2^i, 2^{i+1}], i = 0, 1...$). The purpose of this step is to contain waiting strategy in the algorithm. In the second part, the algorithm schedules the job with bigger width as late as possible and the one with smaller width as early as possible. We call it *small early big late* principle.

Ye and Zhang [111] improved the upper bound by giving an 8-competitive on-line algorithm *DW*. The nature of DW algorithm is also a combination of waiting strategy and a principle of *small early big late*. But DW algorithm differs from Johannes' algorithm in the aspect of waiting time. There were also some results for the cases where some extra information on the jobs is known in advance, i.e., semi-online algorithms. If the jobs arrive in a non-increasing order of processing times, Ye and Zhang proved that greedy strategy works well

by showing that greedy algorithm is 2-competitive [111]. In this situation, there is no need to introduce waiting strategy.

Recent results of online scheduling on parallel jobs focus mostly on a special case with two machines. We know that greedy algorithm simply schedules all jobs one after another leaving no idle time between jobs. Chan et al. [16] mentioned that greedy algorithm is 2-competitive and proved a lower bound $1 + \sqrt{2/3}$. It seems that greedy algorithm works well in this machine setting. Hurink and Paulus [54] further revealed that the lower bound is 2, which means that greedy algorithm is the best possible in two machine environment. For two machine scheduling problem, to improve the competitive ratio, Chan et al. [16] proved that greedy algorithm is optimal with a competitive ratio 3/2 in the case where jobs arrive in a non-decreasing order of processing times. In contrast, in the case where jobs arrive in a non-increasing order of processing times, greedy was shown to be 4/3-competitive [16].

Minimizing total completion time plus penalties

This objective is rarely studied in the online paradigm where jobs arrive over list. Epstein et al. [31] investigated a single machine environment to minimize the total completion time plus penalties. Even if the machine environment is relatively simple compared to those of the previous section, but the objective is more difficult. To solve such a problem, they had to appeal to a strong assumption that jobs' processing times are identical. Otherwise, they showed that there exist no online algorithm with finite competitive ratio. The main idea of their algorithm is not difficult to understand. The nature of their algorithm is to select a criterion to determine accepting or rejecting a current job, as those algorithms mentioned in the previous section. Since all jobs have identical processing times, the number of accepted jobs can denote the total completion time.

Algorithm GREEDY they proposed is a quite simple greedy-type algorithm: when deciding about job J_j, it has the choice between processing the job at a cost (a function of the number of currently accepted jobs), and rejecting the job at a cost. In the function of currently accepted jobs' number, GREEDY multiplies a factor for the reason that an accepted job is more dangerous since it may increase the cost of later jobs, whereas a rejected job cannot. A benefit of this factor is easy to adjust the algorithm in order to achieve a small competitive ratio.

Semi-online scheduling problems

Some scholars tried to improve the algorithm by considering additional jobs' information. Some constraints are extensively studied in the literature. He and Dosa [45], as well as Du [27], considered the constraint that jobs' processing times are bounded in a certain interval. Cheng et al. [22] investigated the semi-online problems with knowledge of total processing times.

Tan and He [100] investigated the semi-on-line versions of scheduling problem with combination of two types of information which belong to five basic types. These types of information are the total processing times (sum), the largest processing time (max), non-increasing processing times ($decr$), the knowledge

of last job arriving ($sugg$), the knowledge of last job with the largest processing time (LL).

Tan and He [101] further studied semi-online scheduling problems with the assumption that some partial additional information is not exactly known in advance. Three versions were considered [101], where we know in advance that the total size of all jobs, the optimal value, and the largest job size are in given intervals, respectively, while their exact values are unknown.

2.5.2 Jobs arrive over time

In this online paradigm, online features are the existence of the jobs whose release time does not pass yet and the specific release time.

Minimizing makespan

If preemption is allowed, there exists an optimal online algorithm which is 1-competitive for identical machine scheduling [38, 49]. The idea behind the algorithm is that whenever a new job arrives, we reschedule the unfinished parts of previous jobs and all unscheduled jobs so that they are finished as early as possible. For uniformly related machines, an optimal algorithm exists if and only if the speeds of machines satisfy $\frac{s_{i-1}}{s_i} \leq \frac{s_i}{s_{i+1}}$, where s_i is the speed of ith fastest machine [107].

Without preemption, the best upper bound for parallel machine scheduling was obtained for the simple algorithm which always schedules the longest job within available jobs. This algorithm is $\frac{3}{2}$-competitive and a lower bound of 1.3473 is shown [20].

If we consider jobs' delivery times, the objective function makespan shifts to the maximum time by which all jobs have been delivered. For a single machine scheduling problem, Hoogeveen and Vestjens [51] proposed an optimal online algorithm *D-LDT* which is $\frac{\sqrt{5}+1}{2}$-competitive. The idea of algorithm D-LDT is that if no jobs with a large processing requirement are available, then we should schedule the job with the largest delivery time; otherwise, we should decide whether to schedule the large job, the job with the largest delivery time, or no job at all. Based on this idea, the situation with bounded delivery times was further considered in [105, 70]. For parallel machine environment, Vestjens [108] gave a lower bound 1.5.

For open shop scheduling on two machines, the greedy algorithm is optimal with a competitive ratio $\frac{3}{2}$ [108]. If preemption is considered, he developed an optimal $\frac{5}{4}$-competitive algorithm [108].

Minimizing makespan on batch machine environment

When the machine environment shifts to batch processing machine, the results get scarce, especially for open shop and flow shop problems. A batch processing machine can handle up to B (the capacity of a machine) jobs simultaneously. The jobs that are processed together form a batch, and all jobs in a batch start and complete at the same time. The processing time of a batch is given by the longest processing time of any job in the batch. Jobs arrive over time. The objective is to minimize the makespan. For the unbounded case where the capacity B is sufficiently large (i.e., all jobs can be processed simultaneously

in a single batch), Zhang et al. [113] designed an optimal $\frac{\sqrt{5}+1}{2}$-competitive algorithm H^{∞}. The intuition behind this algorithm is that jobs should rather wait for a while than being processed immediately after they arrive, i.e., waiting strategy. In order to show the algorithm is σ-competitive, the waiting time is determined for the purpose that the completion time of one job can not exceed σ times the optimal value, i.e., the completion time of that job in an optimal schedule.

For parallel batch machine environment, the difficulty of the problems increase dramatically. Zhang et al. [114] further considered the problem on m identical parallel batch machines. When the batch capacity is infinite, i.e., unbounded case, we need not only to group the jobs into batches as for the single machine problem, but also to choose a machine for an available batch. Under the assumption that jobs' processing times are identical, Zhang et al. [114] proposed a best possible online algorithm with competitive ratio $1+\beta_m$, where β_m is the positive solution of equation $(1+\beta_m)^{m+1} = \beta_m + 2$. The intuition behind this algorithm is similar to the algorithm H^{∞} they proposed for a single batch machine environment. The main idea is also to postpone the processing of jobs. The algorithm for m batch machines contains a waiting time, which is determined in order to control the completion time of one job not greater than σ times the optimal value (i.e., the completion time of that job in an optimal schedule). When the batch capacity is finite, they provided a best possible online algorithm $A^b(\alpha)$ with a competitive ratio $\alpha = \frac{\sqrt{5}+1}{2}$. For this algorithm, the basic idea is also to postpone the starting times of jobs such that their completion times can not exceed α times the optimal value. In the algorithm, we also have to make sure that the capacity of a batch does not exceed its capacity. This character is due to the machine environment, i.e., the batch capacity is finite.

Here is a question: "Does there exist an optimal algorithm for parallel batch machines without any restriction?" Nong et al. [81] investigated the problem on two parallel batch machines with infinite capacity. They provided an algorithm with a competitive ratio $\sqrt{2}$. The idea of their algorithm is similar to Zhang's algorithm. Use waiting strategy to design the algorithm and control the completion time of a job not bigger than $\sqrt{2}$ times the optimal one. Tian et al. [106] showed a lower bound $\sqrt{2}$ of competitive ratio, which means that Nong's algorithm is optimal. They also proposed a new algorithm also using waiting strategy, which is more efficient than Nong's algorithm.

Considering delivery time in batch machine scheduling, Tian et al. [104] provided an online algorithm H_1^{∞} with a competitive ratio 2 for infinite batch machine situation and an algorithm H^B with a competitive ratio 3 for finite batch situation. The idea of algorithm H_1^{∞} is to use a forwards dynamic programming proposed in [14] as a component to schedule current arriving jobs. For algorithm H^B, the idea is similar to regular one which uses a waiting strategy and controls the completion time of one job not bigger than 3 times the optimal value. The speciality of this problem is that jobs' delivery times contribute to their completion times. It is also necessary to make sure that the capacity of a batch does not exceed its capacity.

Since no optimal algorithm for the problems with delivery times had been obtained, Yuan et al. [112] considered two restricted models: (1) the jobs have small delivery times, i.e., for each job J_j its delivery time q_j is not bigger than

its processing time, (2) the jobs have agreeable processing and delivery times, i.e., for any two jobs J_i and J_j , $p_i > p_j$ implies $q_i \geq q_j$. They proposed an optimal algorithm for these two restricted situation. The intuition behind this algorithm is also to postpone the starting time of batch.

There are other papers which studied batch machine scheduling models with additional constraints. Ridouard et al. [88] also considered agreeable processing times constraint. Fu et al. [35, 34] investigated the problems with restarts. Nong et al. [80] addressed the problem with family jobs.

Minimizing total completion time

For preemptive scheduling on a single machine, it is easy to construct an optimal schedule by always scheduling the job with shortest remaining processing time. The same rule yields a 2-competitive algorithm for identical machine problem [83].

For single machine problem, if preemption is not allowed, 2-competitive algorithms were given and they are optimal [108, 77]. The idea behind these algorithms (such as *Delayed SPT* [108]) is to shift release times, then schedule them by SPT (shortest processing time) rule.

Stee and Poutre [98] investigated single machine variant with restarts. Allowing restarts means that the processing of a job may be interrupted, losing all the work done on it. In this case, the job must be started again later (restarted), until it is completed without interruptions. By using restarts, Stee and Poutre [98] designed an algorithm RSPT (restarting SPT) with a competitive ratio $\frac{3}{2}$. RSPT bases the decision about whether or not it will interrupt a current processing job J_j for an arriving job $J_{j'}$ solely on J_j and $J_{j'}$. It ignores, for example, all other jobs that are waiting to be processed. RSPT only interrupts a job J_j for jobs that are smaller and that can finish earlier than J_j. In fact, RSPT algorithm is identical to SRPT (Shortest Remaining Processing Time) rule and ECT (Earliest Completion Time) rule. Their results showed that restarts help.

For minimizing total weighted completion time, a slight modification of Delayed-SPT also yields an optimal online algorithm with a competitive ratio 2 [2]. The intuition behind this modification *Delayed SWPT* is to shift the release times, then schedule them by SWPT (shortest weighted processing time) rule.

For parallel machine variants, Vestjens [108] showed a lower bound 1.309. Liu and Lu [75] developed a slightly modified algorithm *DSPT* with a competitive ratio 2. Their algorithm DSPT differs from Delayed SPT in choosing one of the available machines. In the preemption-resume environment, the currently known lower bound is $\frac{22}{21}$ given by Vestjens [108].

Chekuri et al. [17] presented a relaxation technique that converts a scheduling problem of parallel machines to a preemptive scheduling problem on a single machine. Based on this technique, they gave a $(3-\frac{1}{m})$-competitive algorithm for $P|r_j, online|\sum C_j$. Recently, Liu and Lu [76] studied two uniform machine variant and presented a 2.618-competitive algorithm FCFS for $Q2|r_j, online|\sum C_j$. They also used this relaxation technique to convert the original problem to a preemptive single machine problem, and then applied SRPT rule for this relaxation problem. FCFS algorithm uses the new schedule obtained by solving the relaxation problem to determine how to assign the current job. Since SRPT rule works online, the FCFS is an online algorithm. For two uniform machine en-

vironment, they further considered the weights of jobs and preemption-resume setting, i.e., $Q2|r_j, online, pmtn|\sum w_jC_j$. Suppose two machines are M_1 with speed 1 and M_2 with speed s, respectively. They showed that WSPT (Weighted SPT) algorithm is 2-competitive. We say that one job J_j with a lower value of the ratio p_j/w_j has a higher priority, where p_j is the processing time, and w_j is the weight. WSPT rule always schedules the job with the highest priority among available processing jobs on M2, then schedule the job with the highest priority among remaining available processing jobs on M_1, if any. This algorithm is a extension of WSPT rule for parallel machine situations. Liu et al. [72] further studied $Q|r_j, online|\sum C_j$ and $Q|r_j, online, pmtn|\sum w_jC_j$.

Maximization the number of early jobs

Maximization problems were rarely studied. In offline scheduling, a similar objective function, minimizing the number of late jobs (or weighted number of late jobs), was widely studied [7, 8, 63, 79, 86]. Other objective functions related to earliness and tardiness were also investigated in offline framework [97, 96, 47, 48, 89].

In online scheduling, for maximizing the number of early jobs on a single machine in the preemption-restart model, Hoogeveen et al. [50] provided an optimal algorithm SRPT with a competitive ratio 1/2. This algorithm is based on the SRPT (shortest remaining processing time) rule or ECT (earliest completion time) rule. Further study for parallel machines were investigated by Liu et al. [73].

Other constraints and objectives were also studied in the literature. Epstein and Stee [32] investigated the model with restarts. Azar and Epstein [5], as well as Huo et al. [53], studied online scheduling with precedence constraints. Kellerer et al. dealt with the objective of minimizing total flow time. Megow and Schulz [78] addressed the online problem to minimize average completion time revisited. Zheng et al. [115] considered online scheduling with cancelation. Havill and Mao [43] considered online scheduling of perfectly malleable jobs with setup times. Zheng et al. [116] investigated semi-online variant with *lookahead* information.

2.5.3 Summary

Greedy and waiting strategies are often used in designing the algorithms. In parallel machine scheduling, LIST algorithm plays an important role. The golden ratio occurs in the computation of competitive ratio. (Is this a coincidence or the beauty of mathematics?)

Even if hundreds of online scheduling problems and models have been studied and analyzed in the past, there are still a lot of problems which have not been employed or perfectly solved.

Chapter 3

Single machine model

We study an online scheduling problem on a single machine with delivery times. The problem is online in the sense that all jobs arrive over time. Each job's characteristics, such as processing time and delivery time, become known at its arrival time. Preemption is not allowed and once the processing of a job is completed we delivery it to the destination by a vehicle. The objective is to minimize the time by which all jobs have been delivered. In this chapter, we assume that all jobs have bounded delivery times, which means that given a certain positive number $\beta \geq \frac{1}{2}$, for each job J_j, we have a release time $r_j \geq 0$ and $\beta q_j \leq p_j$, where p_j, q_j denote the processing time and the delivery time of job J_j, respectively. We use $1|online, r_j, \beta q_j \leq p_j|L_{max}$ to denote the problem for short, where L_{max} denote the time by which all jobs have been delivered. We prove a lower bound of competitive ratios for all online algorithms and propose an optimal online algorithm with a competitive ratio of $\frac{1}{2}(\sqrt{5+\beta^2+2\beta}+1-\beta)$.

3.1 Introduction

Makespan, the completion time of last job in the schedule, is commonly researched in the literature. This is an objective in the aspect of manufacturers. In order to consider the problems in the aspect of the clients, we focus our attention on the time when each client receive his demanded job (or product), that is the sum of completion time and delivery time of that job. In practice, a job's delivery time cannot be unlimited. These reasons motivate us to investigate online scheduling problem on a single machine with bounded delivery times.

The problem considered can be described as follows. There are n jobs, a single machine and sufficiently many vehicles. Each job has an arrival time, a processing time, and a bounded delivery time. These characteristics of a job are unknown until it arrives. Once the processing of a job is completed on a machine, we deliver it to the destination by a vehicle. No preemption is allowed. The objective is to minimize the time by which all jobs have been delivered. We use r_j, p_j and q_j to denote the arrival time, the processing time and the delivery time of job J_j, respectively. The bounded delivery time means that there is a certain positive number $\beta \geq \frac{1}{2}$, such that $\beta q_j \leq p_j$ for any j such that $1 \leq j \leq n$. Suppose that σ is a schedule of the jobs. We use $S_j(\sigma)$, $C_j(\sigma)$ and $L_j(\sigma)$ to denote the starting time of J_j, the completion time of J_j and the time by which

J_j has been delivered in schedule σ. The objective function of the considered problem can be expressed as follows,

$$L_{max}(\sigma) = \max\{L_j(\sigma) : L_j(\sigma) = C_j(\sigma) + q_j, 1 \leq j \leq n\}.$$

Under offline setting, there are many results about scheduling problem with the objective to minimize the time by which all jobs have been delivered. A well known algorithm called the LDT rule (largest delivery time first) is optimal for the problem $1||L_{max}$. For problem $1|r_j|L_{max}$, Kise et al. [61] proved that the LDT rule is 2-approximate and Lawer et al. [64] proved that the problem is strongly NP-hard. While under online setting, Hoogeveen et al. [52] provided an optimal online algorithm with a competitive ratio of $\frac{\sqrt{5}+1}{2}$ for the problem $1|online, r_j|L_{max}$. Based on these results, Tian et al. [105] showed a better online algorithm with competitive ratio of $\sqrt{2}$ for the problem $1|online, r_j, q_j \leq p_j|L_{max}$.

In this chapter, we consider the problem $1|online, r_j, \beta q_j \leq p_j|L_{max}$, where $\beta \geq \frac{1}{2}$. The rest of this chapter is organized as follows. In Section 2, we show that no online algorithm has a competitive ratio less than $\frac{1}{2}(\sqrt{5+\beta^2+2\beta} + 1-\beta)$. In section 3, we propose an online algorithm for this problem. In Section 4, we prove the algorithm proposed in section 3 is optimal.

3.2 A lower bound of competitive ratio

In this section, we present a lower bound of competitive ratio for the problem, i.e., $\varphi = \frac{1}{2}(\sqrt{5+\beta^2+2\beta}+1-\beta)$. Note that in the following proof, we consider the problem where $\beta > 0$. And the result is a general lower bound containing the case where $\beta \geq \frac{1}{2}$.

For simplicity of expression, let $\varphi = \frac{1}{2}(\sqrt{5+\beta^2+2\beta}+1-\beta)$, where $\beta > 0$. We derive a simple but important equality, which will be used in the following proofs.

Lemma 1 $\frac{1}{\varphi+\beta} = \varphi - 1 = \frac{1+\varphi-\varphi^2}{\beta} > 0$.

Proof.

$$\begin{aligned}\frac{1}{\varphi+\beta} &= \frac{2}{\sqrt{5+\beta^2+2\beta}+1+\beta} = \frac{2(\sqrt{5+\beta^2+2\beta}-(1+\beta))}{5+\beta^2+2\beta-(1+\beta)^2} \\ &= \frac{1}{2}(\sqrt{5+\beta^2+2\beta}-1-\beta) = \varphi - 1.\end{aligned}$$

$$\begin{aligned}\varphi^2 &= \frac{1}{4}[5+\beta^2+2\beta+1+\beta^2-2\beta+2(1-\beta)\sqrt{5+\beta^2+2\beta}] \\ &= \frac{1}{2}[3+\beta^2+(1-\beta)\sqrt{5+\beta^2+2\beta}].\end{aligned}$$

$$\frac{1+\varphi-\varphi^2}{\beta} = \frac{\beta\sqrt{5+\beta^2+2\beta}-\beta-\beta^2}{2\beta} = \frac{1}{2}(\sqrt{5+\beta^2+2\beta}-1-\beta) = \varphi - 1.$$

$$\varphi - 1 = \frac{1}{2}(\sqrt{5+\beta^2+2\beta}-1-\beta) = \frac{1}{2}[\sqrt{4+(1+\beta)^2}-(1+\beta)] > 0.$$

According to the above equalities and inequality, we have the desired result. ■

Theorem 1 *For the problem* $1|online, r_j, \beta q_j \leq p_j|L_{max}$*, no online algorithm has a competitive ratio less than* $\varphi = \frac{1}{2}(\sqrt{5+\beta^2+2\beta}+1-\beta)$*, where* $\beta > 0$*.*

Proof. For any online algorithm $\mathcal{A}$, we consider the following instance. Let π and σ denote an optimal offline schedule and the schedule produced by online algorithm $\mathcal{A}$ for the instance, respectively. We use $J_j = (p_j, q_j)$ to denote a job J_j, where p_j and q_j are the processing time and the delivery time of J_j, respectively. The first job $J_1 = (1, 0)$ arrives at time 0. We assume that algorithm $\mathcal{A}$ schedules $J_1 = (1, 0)$ at time $T_{\mathcal{A}} \geq 0$. From Lemma 1, we know that $\varphi - 1 > 0$. Depending on $T_{\mathcal{A}}$, we consider two cases as follows. (Job $J_1 = (1, 0)$ is constructed in order to avoid algorithm $\mathcal{A}$ schedules it too early, i.e., before time $\varphi - 1$).

Case 1: $T_{\mathcal{A}} \geq \varphi - 1$.

In the worst-case, no jobs arrive any more. In an optimal situation, $J_1 = (1, 0)$ is scheduled at time 0, which gives $L_{max}(\pi) = 1$. Then

$$\frac{L_{max}(\sigma)}{L_{max}(\pi)} = \frac{T_{\mathcal{A}} + 1}{1} \geq \varphi.$$

Case 2: $0 \leq T_{\mathcal{A}} < \varphi - 1$.

In the worst-case, the second job $J_2 = (\beta, 1)$ arrives at time $T_{\mathcal{A}}$. The optimal solution consists of scheduling J_2 then J_1. Thus

$$\frac{L_{max}(\sigma)}{L_{max}(\pi)} = \frac{T_{\mathcal{A}} + 1 + \beta + 1}{T_{\mathcal{A}} + \beta + 1} = 1 + \frac{1}{T_{\mathcal{A}} + \beta + 1} > 1 + \frac{1}{\varphi + \beta}.$$

From Lemma 1, it follows that $1 + \frac{1}{\varphi+\beta} = 1 + \varphi - 1 = \varphi$. Therefore, we have

$$\frac{L_{max}(\sigma)}{L_{max}(\pi)} > \varphi \quad (\varepsilon \to 0).$$

The theorem follows. ■

Remark 1 *For the problem* $1|online, r_j|L_{max}$*, let* $\beta \to 0$*, then we obtain the lower bound of* $\frac{\sqrt{5}+1}{2}$ *proved in [52].*

Remark 2 *For the problem* $1|online, r_j, q_j \leq p_j|L_{max}$*, which is a special case of our model with* $\beta = 1$*, we obtain the lower bound of* $\sqrt{2}$ *proposed in [105].*

The lower bound $\frac{1}{2}(\sqrt{5+\beta^2+2\beta}+1-\beta)$, where $\beta > 0$, is a generalized lower bound.

3.3 An online algorithm EX-D-LDT

The idea of the algorithm is originated from D-LDT algorithm (delayed LDT-rule) proposed in [52]. Considering the proof of the lower bound, we know that if an algorithm wants to guarantee a better performance bound, then it needs a waiting strategy. Therefore, we modify the LDT (Largest Delivery Time) rule and add some waiting times. Since we use the same idea as for D-LDT algorithm, we must mention the idea first. The basic idea behind D-LDT algorithm is that, if no jobs with a large processing requirement are available, then we schedule the

job with the largest delivery time; otherwise, we decide whether to schedule the large job, the job with the largest delivery time, or no job at all. The algorithm *EX-D-LDT* (Extended D-LDT) described in the following has a few differences from that in [52]. For the case where $\beta \geq \frac{1}{2}$, we have

$$1 < \varphi \leq \frac{3}{2}. \tag{3.1}$$

First, adopt some notations from [52] and [105].

-$p(S)$ denotes the total processing time of all jobs in the set S, i.e., $p(S) = \sum_{J_j \in S} p_j$;

-$J(t)$ is the set containing all jobs that arrived at or before time t;

-$U(t)$ is the set containing all jobs in $J(t)$ that have not been started at time t;

-t_1 denotes the start time of the last idle time period before time t; if there is no idle time, then define $t_1 = 0$;

-A job J_j is said to be big if $p_j > \frac{1}{\varphi} p((J(t) \backslash J(t_1)) \bigcup U(t_1))$;

-$p_{max}(t)$ denotes the index of the job with the largest processing time in $U(t)$;

-$q_{max}(t)$ denotes the index of the job with the largest delivery time in $U(t)$.

Note that $(J(t) \backslash J(t_1)) \bigcup U(t_1)$ contains the jobs which arrive from time t_1 to t and the jobs which were not started (or completed, since t_1 is the start time of idle time) at time t_1; if $t_1 = 0$, $(J(t) \backslash J(t_1)) \bigcup U(t_1) = J(t)$ for any t. And also, from equality (3.1), we have $\frac{1}{\varphi} \geq \frac{2}{3} > \frac{1}{2}$. Therefore, there is at most one big job at any time t. The online algorithm runs as follows.

Algorithm EX-D-LDT

Step 1: Wait until a decision point, where the machine is idle and a job is available (if all jobs have been scheduled, output the schedule). Suppose this happens at time t. Determine $p_{max}(t)$ and $q_{max}(t)$.

Step 2: If there is no big job available, then schedule $J_{q_{max}(t)}$. Go to **Step 1**.

Step 3: If $J_{p_{max}(t)}$ is the only available job, then wait (or keep idle) until a new job arrives or until time $r_{p_{max}(t)} + (\varphi - 1)p_{p_{max}(t)}$, whichever happens first;

otherwise,

If $t + p(U(t)) > r_{p_{max}(t)} + \varphi p_{p_{max}(t)}$,

(3.1) If $q_{q_{max}(t)} > (\varphi - 1)p_{p_{max}(t)}$, schedule $J_{q_{max}(t)}$;

(3.2) otherwise, schedule $J_{p_{max}(t)}$;

else,

(3.3) If $q_{max}(t) \neq p_{max}(t)$, then schedule $J_{q_{max}(t)}$;

(3.4) otherwise, schedule the job with the second largest delivery time, if any.

Step 4: Go to **Step 1**.

The lower bound φ is a key parameter in the algorithm. In Step 1 the algorithm wait to a decision point and find the job with the largest processing time $J_{p_{max}(t)}$ and the job with the largest delivery time $J_{q_{max}(t)}$. If all jobs in the job instance have been scheduled, the algorithm stop and output the schedule. In Step 2, we divide into two cases. If there is no big job available, schedule the job with the largest delivery time $J_{q_{max}(t)}$; otherwise, go to Step 3. In Step 3, we have a precondition that there is a big job $J_{p_{max}(t)}$. If this big job is the only available job, the algorithm wait until a new decision point or time $r_{p_{max}(t)} + (\varphi - 1)p_{p_{max}(t)}$. Because the algorithm cannot wait infinite time.

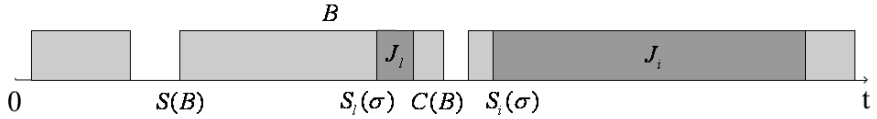

Figure 3.1: Structure of a counterexample

Note that the expression $r_{p_{max}(t)}+(\varphi-1)p_{p_{max}(t)}$ contains a parameter φ which is the lower bound we have proved. This time is chosen in order that when this job is the only one in job instance, the competitive ratio of the algorithm is

$$\begin{aligned}\rho &\leq \frac{r_{p_{max}(t)}+(\varphi-1)p_{p_{max}(t)}+p_{p_{max}(t)}+q_{p_{max}(t)}}{r_{p_{max}(t)}+p_{p_{max}(t)}+q_{p_{max}(t)}} \\ &\leq \max\left\{\frac{r_{p_{max}(t)}}{r_{p_{max}(t)}}, \frac{\varphi p_{p_{max}(t)}}{p_{p_{max}(t)}}, \frac{q_{p_{max}(t)}}{q_{p_{max}(t)}}\right\}=\varphi.\end{aligned}$$

Otherwise, i.e., the big job $J_{p_{max}(t)}$ is not the only available job. We divided into cases in order that the competitive ratio of the algorithm is not greater than φ. In Step 3.2, $J_{p_{max}(t)}$ is chosen in order that when no job arrives in future, the competitive ratio of the algorithm is

$$\rho \leq \frac{t+p_{p_{max}(t)}+p(U(t)\backslash p_{p_{max}(t)})+q_{q_{max}(t)}}{t+p_{p_{max}(t)}+p(U(t)\backslash p_{p_{max}(t)})} \leq \frac{p_{p_{max}(t)}+q_{q_{max}(t)}}{p_{p_{max}(t)}} \leq \varphi.$$

This is an example. The purpose of Step 3 is to guarantee the competitive ratio of the algorithm not more than φ. Readers can find more details in Theorem 2.

In order to prove that algorithm EX-D-LDT has a competitive ratio of φ by contradiction in the next section, we assume that there exists a smallest counterexample consisting of a minimum number of jobs. For this smallest counterexample (like for all other counterexample), the competitive ratio of EX-D-LDT algorithm is greater than φ.

In the following, we will show several properties of this smallest counterexample, denoted by $\mathcal{I}$. Let σ and π be the schedules produced by EX-D-LDT and offline optimal schedule, respectively. Let $S_j(\sigma)$ denote the start time of job J_j in σ. Let J_l denote the first job in σ that assumes (or determines) the value $L_{max}(\sigma)$, i.e., $L_l(\sigma)=L_{max}(\sigma)$. Similar to the lemmas proved in [52] and [105], we obtain several structural properties that schedule σ satisfies.

Lemma 2 *The schedule σ consists of a single block: it starts at a nonnegative time and after that all jobs are processed contiguously.*

Proof. By contradiction. Suppose that σ consists of more than one block (see Fig.3.1). Let B be the block that contains job J_l,

$$L_l(\sigma)=L_{max}(\sigma). \tag{3.2}$$

(1) The first step is to reduce the jobs' number of the smallest counterexample by its definition.

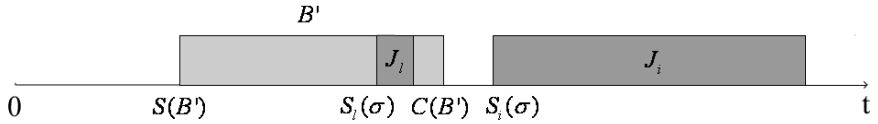

Figure 3.2: Structure of a smaller counterexample

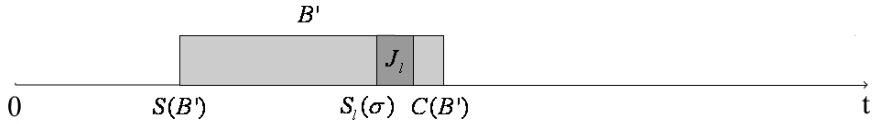

Figure 3.3: Structure of the smallest counterexample $\mathcal{I}$

Consider any block that precedes B. Since the algorithm bases its choices on the set $(J(t)\backslash J(t_1))\bigcup U(t_1)$ (considering the definition of the big job), the existence of the jobs that are completed before the start of block B does not influence the start time of B and the order in which the jobs are executed. Therefore, we delete all jobs completed before the start of block B without changing the value $L_{max}(\sigma)$ and increasing $L_{max}(\pi)$. Similarly, we can remove all jobs from $\mathcal{I}$ that are released after the start time $S_l(\sigma)$ of J_l in σ.

(2) The second step is to form a new counterexample.

From the minimization of $\mathcal{I}$, we assume that our counterexample consists of the jobs which arrives before time $S_l(\sigma)$ from block B, which form a new block B', and the jobs that are available at the start time $S_l(\sigma)$ of J_l in σ but that are scheduled in another block. We keep these jobs in the counterexample since they result in the start time $S_l(\sigma)$ of job J_l by the algorithm. Note that there indeed exists the jobs which are available at the start time $S_l(\sigma)$ of J_l in σ but are scheduled in another block. B' must be a continuous block, since the algorithm always starts a job if more than one job is available and the machine is idle (by Step 1 of the algorithm). For the same reason, there is at most one job that is available at time $S_l(\sigma)$ and does not belong to B' and this job must exist, otherwise the lemma holds. Moreover, this job, denoted by J_i, must be marked as big (by Step 2 of the algorithm). Otherwise, J_i must be scheduled following the last job in B'. Therefore, we have the new counterexample formed by B' and J_i (See Fig.3.2).

(3) The third step is to prove that there is a contradiction if J_i exists.

Let $S(B')$ and $C(B')$ denote the start time of the first job and the completion time of the last job in B'. We know that $C(B') \geq S_l(\sigma) + p_l$.

Since J_i is big and the only job at time $C(B')$, by Step 3 of the algorithm, we know that $S_i(\sigma) > C(B') \geq S_l(\sigma) + p_l$. Then, $L_i(\sigma) = S_i(\sigma) + p_i + q_i > S_l(\sigma) + p_l + p_i + q_i > S_l(\sigma) + p_l + p_i$. From equality (3.2), we have $L_{max}(\sigma) = S_l(\sigma) + p_l + q_l$. Therefore, $L_{max}(\sigma) - L_i(\sigma) < q_l - p_i$. Since J_i is a big job, $p_i > \frac{1}{\varphi}p(\mathcal{I}) > \frac{1}{\varphi}(p_i + p_l)$. Then, we have $p_l < (\varphi - 1)p_i$. Since $\beta q_l \leq p_l$, it follows that $L_{max}(\sigma) - L_i(\sigma) < q_l - p_i \leq \frac{1}{\beta}p_l - p_i < \frac{1}{\beta}(\varphi - 1)p_i - p_i = (\frac{\varphi - 1}{\beta} - 1)p_i$.

Since $\beta \geq \frac{1}{2}$, considering inequality (3.1), we have that

$$\frac{\varphi - 1}{\beta} - 1 \leq \frac{1}{2\beta} - 1 \leq 0.$$

Therefore, $L_{max}(\sigma) - L_i(\sigma) < 0$. This contradicts the assumption that J_l is the first job in σ assuming the value $L_{max}(\sigma)$. Therefore, The schedule σ consists of a single block B' (see Fig.3.3) and the lemma follows. ■

From now on, let J_0 be the job that arrives first in $\mathcal{I}$. Without loss of generality, we assume that $r_0 = 0$.

Lemma 3 *In the optimal schedule π,*

(a) if J_0 is the first scheduled job, we have $L_{max}(\sigma) - L_{max}(\pi) \leq (\varphi - 1)p_0 + q_l$;

(b) if J_0 is not the first scheduled job, we have $L_{max}(\sigma) - L_{max}(\pi) \leq q_l$.

Proof. By lemma 2, σ consists of a single block B'. Since J_l is the first job in σ that assumes the value $L_{max}(\sigma)$, then $L_{max}(\sigma) = C_l(\sigma) + q_l \leq C(B') + q_l = S(B') + p(\mathcal{I}) + q_l$. By the algorithm, we have

$$S(B') = \min\{(\varphi - 1)p_0, r_1\}, \tag{3.3}$$

where r_1 is the arrival time of the second available job. This equality will be used in the following lemmas. If J_0 is the first scheduled job in π, we have $L_{max}(\pi) \geq p(\mathcal{I})$. Hence, it follows that $L_{max}(\sigma) - L_{max}(\pi) \leq S(B') + q_l \leq (\varphi - 1)p_0 + q_l$. Therefore, (a) follows. If J_0 is not the first scheduled job in π, we have $L_{max}(\pi) \geq r_1 + p(\mathcal{I}) \geq S(B') + p(\mathcal{I})$. Hence, it follows that $L_{max}(\sigma) - L_{max}(\pi) \leq q_l$. Thus, (b) holds. ■

As in [52] and [105], let J_k be the last job scheduled in σ before J_l with a delivery time smaller than q_l, if any. If J_k exists, let $G(l)$ denote the set of all jobs between J_k and J_l in σ, including J_l. Note that each job in $G(l)$ has a delivery time greater than or equal to q_l. We name J_k as the *interference job* for schedule.

The purpose of the following lemma is to prove there there exists an interference job in σ. This job will be used in the proof of Theorem 2.

Lemma 4 *Schedule σ contains an interference job J_k scheduled before J_l such that $q_k \leq q_l$.*

Proof. By contradiction. We assume that this interference job J_k does not exist in σ. We also use $G(l)$ denote all jobs scheduling J_l, including J_l. Note that each job in $G(l)$ has a delivery time greater than or equal to q_l since J_k does not exist. We know that in the smallest counterexample $\mathcal{I}$, the competitive ratio of the algorithm is greater than φ.

Since each job in $G(l)$ has a delivery time greater than or equal to q_l, we have

$$L_{max}(\pi) \geq p(G(l)) + q_l.$$

That is to say, in set $G(l)$, job J_l must be the last job in σ according to their positions (or completion times); otherwise, the objective value must increase. By the algorithm, the first job in the block starts at time $(\varphi - 1)p_0$ at the latest, considering equality (3.3). Therefore, it follows that

$$L_{max}(\sigma) = C_l(\sigma) + q_l \leq (\varphi - 1)p_0 + p(G(l)) + q_l.$$

Therefore, $L_{max}(\sigma)-L_{max}(\pi) \leq (\varphi-1)p_0 \leq (\varphi-1)L_{max}(\pi)$. That is to say, the competitive ratio of the algorithm is not greater than φ. Thus, this contradicts to the fact that we consider a counterexample where the competitive ratio of the algorithm is greater than φ. Therefore, the lemma follows. ■

Lemma 5 *$p_k > (\varphi - 1)p(\mathcal{I})$.*

Proof. By contradiction. We assume that $p_k \leq (\varphi - 1)p(\mathcal{I})$. There are two possibilities for the algorithm to select J_k and not one of the jobs from $G(l)$.

(1) All jobs in $G(l)$ have a release date larger than $S_k(\sigma)$.

(2) There is one job from $G(l)$ available, which we denote by J_1, that is marked as big and cannot be started yet. Note that since J_1 cannot be started yet, corresponding to Step 3.4 of the algorithm, we must have that $S_k(\sigma)+p_k \leq r_1 + (\varphi - 1)p_1$.

For case (1), we have that

$$L_{max}(\pi) \geq \min_{J_j \in G(l)} r_j + p(G(l)) + q_l > S_k(\sigma) + p(G(l)) + q_l.$$

Since $L_{max}(\sigma) = C_l(\sigma) + q_l = S_k(\sigma) + p_k + p(G(l)) + q_l$, it follows that

$$L_{max}(\sigma) - L_{max}(\pi) < p_k \leq (\varphi - 1)p(\mathcal{I}) \leq (\varphi - 1)L_{max}(\pi).$$

This contradicts to the fact that we consider a counterexample.

Considering case (2), we have that

$$L_{max}(\pi) \geq \min_{J_j \in G(l)} r_j + p(G(l)) + q_l > r_1 + p(G(l)) + q_l.$$

Since $L_{max}(\sigma) = C_l(\sigma) + q_l = S_k(\sigma) + p_k + p(G(l)) + q_l$, it follows that

$$\begin{aligned} L_{max}(\sigma) - L_{max}(\pi) &< S_k(\sigma) + p_k - r_1 \leq r_1 + (\varphi - 1)p_1 - r_1 = (\varphi - 1)p_1 \\ &\leq (\varphi - 1)L_{max}(\pi). \end{aligned}$$

This also contradicts to the fact that we consider a counterexample. Therefore, the lemma follows. ■

Lemma 6 *For all $J_j \in \mathcal{I}\backslash\{J_0\}$, we have $p_j \leq \frac{1}{\varphi}p(\mathcal{I})$.*

Proof. By contradiction. We assume that there exists at least one job J_i with $r_i \geq r_0$, which satisfies that $p_i > \frac{1}{\varphi}p(\mathcal{I})$. Then we know that p_i is the only big job at any time after its arrival (by the definition of big job). We divide set $J(t)$ into two independent parts: $J(t)\backslash U(t)$ and $U(t)$.

Suppose that at time t such that $t \in [S(B'), S(B') + p(J(\mathcal{I}))]$, since $p_i > \frac{1}{\varphi}p(\mathcal{I}) \geq \frac{1}{\varphi}p(J(t))$, we have $\varphi p_i > p(J(t)\backslash U(t)) + p(U(t))$. Note that $J(t)\backslash U(t)$ denotes the set of jobs which have been scheduled at time t. By the algorithm, we have $S(B') \leq r_i$, considering equality (3.3). We know that the schedule σ consists of a single block. By the construction of this block, we know that $t = S(B') + p(J(t)\backslash U(t))$. Therefore,

$$t+p(U(t)) = S(B')+p(J(t)\backslash U(t))+p(U(t)) \leq r_i+p(J(t)) \leq r_i+p(\mathcal{I}) < r_i+\varphi p_i.$$

Note that among Step 3.1, 3.2, 3.3 and 3.4, the big job can be scheduled only in Step 3.2. Since $t + p(U(t)) < r_i + \varphi p_i$ always holds, Step 3.2 can not execute.

Thus, the big job can be scheduled only in the case where time $t \geq r_i + (\varphi - 1)p_i$ (by Step 3). That is to say, J_i cannot be scheduled before time $r_i + (\varphi - 1)p_i$.

Thus, $C_i(\sigma) \geq r_i + (\varphi - 1)p_i + p_i = r_i + \varphi p_i$. Since $C_i(\sigma) \leq S(B') + p(\mathcal{I}) < S(B') + \varphi p_i$, we have $r_i + \varphi p_i < S(B') + \varphi p_i$. Therefore, $r_i < S(B')$. This is a contradiction to $S(B') \leq r_i$. So the lemma follows. ■

Lemma 7 *In the case where each job in $G(l)$ arrives after time $S_k(\sigma)$ and J_k is a big job at time $S_k(\sigma)$, we have $S_k(\sigma) \geq r_k + (\varphi - 1)p_k$.*

Proof. By contradiction. Assume that job J_k's start time $S_k(\sigma) < r_k + (\varphi - 1)p_k$. We discuss the following two cases.

Case 1: J_k is the only available job at time $S_k(\sigma)$.

From Step 3 of the algorithm, we know that J_k cannot be scheduled before a new job arrives or $r_k + (\varphi - 1)p_k$. Since $S_k(\sigma) < r_k + (\varphi - 1)p_k$, J_k cannot be scheduled at time $S_k(\sigma)$. There is a contradiction.

Case 2: J_k is not the only available job at time $S_k(\sigma)$.

It follows that set $U(S_k(\sigma))\backslash\{J_k\} \neq \emptyset$ and the jobs in this set are scheduled after job J_l. We can delete these jobs in $\mathcal{I}$ without decreasing $L_{max}(\sigma)$ and increasing $L_{max}(\pi)$. Because J_l should be scheduled at last in $G(l)$; otherwise, $L_{max}(\sigma)$ increases. This contradicts to the minimization of the smallest counterexample $\mathcal{I}$.

Therefore, the lemma follows. ■

3.4 The proof of optimality of EX-D-LDT

By the lemmas proved in the above section, we can show the following theorem. The idea of proof of the following theorem originated from [105]. The idea is that we discuss three different possibilities which result in that algorithm EX-D-LDT select interface job J_k instead of one of the jobs from $G(l)$ at time $S_k(\sigma)$. The method is to prove that there are contradictions.

Theorem 2 *The online algorithm EX-D-LDT has an optimal competitive ratio of $\varphi = \frac{1}{2}(\sqrt{5 + \beta^2 + 2\beta} + 1 - \beta)$, when $\beta \geq \frac{1}{2}$.*

Proof. We assume that there exists a smallest counterexample $\mathcal{I}$ such that $L_{max}(\sigma) > \varphi L_{max}(\pi)$, where σ and π denote the schedule obtained by algorithm EX-D-LDT for instance $\mathcal{I}$ and the offline optimal schedule, respectively. Due to Lemma 2 and Lemma 4, we know that σ consists of a single block and contains an interference job J_k. In order to show that such a counterexample does not exist, we need to prove that $L_{max}(\sigma) \leq \varphi L_{max}(\pi)$ or $L_{max}(\sigma) - L_{max}(\pi) \leq (\varphi - 1)L_{max}(\pi)$.

Considering the interference job J_k, we have

$$L_{max}(\sigma) = S_k(\sigma) + p_k + p(G(l)) + q_l. \tag{3.4}$$

There are three possible reasons that algorithm EX-D-LDT selected J_k instead of one of the jobs from $G(l)$ at time $S_k(\sigma)$.
(1) No jobs in $G(l)$ is available at time $S_k(\sigma)$.
(2) There is a big job $J_i \in G(l)$ available at time $S_k(\sigma)$, and algorithm EX-D-LDT is not allowed to schedule it at time $S_k(\sigma)$. This corresponds to Step 3.4.

(J_k is not a big job.)
(3) J_k is a big job at time $S_k(\sigma)$, and all jobs from $G(l)$ that are available at time $S_k(\sigma)$ have a delivery time of at most $(\varphi-1)p_k$. This corresponds to Step 3.2.

In the following, we discuss these three cases, respectively.

Case 1: No jobs in $G(l)$ is available at time $S_k(\sigma)$.

Since all jobs in $G(l)$ arrive after time $S_k(\sigma)$, we have

$$L_{max}(\pi) > S_k(\sigma) + p(G(l)) + q_l. \tag{3.5}$$

From equality (3.4), we have $L_{max}(\sigma) - L_{max}(\pi) < p_k$. From Lemma 5, we have

$$p_k > (\varphi-1)p(\mathcal{I}). \tag{3.6}$$

In the following, considering whether J_k is or not a big job at time $S_k(\sigma)$, we consider two subcases.

Case 1.1: J_k is not a big job at time $S_k(\sigma)$.

We have $p_k \leq \frac{1}{\varphi}p(J(S_k(\sigma)))$. It follows that

$$p(J(S_k(\sigma))) \geq \varphi p_k. \tag{3.7}$$

It can be observed that

$$L_{max}(\pi) \geq p(\mathcal{I}) \geq p(J(S_k(\sigma))) + p(G(l)) \tag{3.8}$$

Therefore, from inequalities (3.6), (3.7) and (3.8), we must have that

$$p_k > (\varphi-1)[p(J(S_k(\sigma))) + p(G(l)] \geq (\varphi-1)[\varphi p_k + p(G(l)]. \tag{3.9}$$

Therefore, it follows that

$$p(G(l)) < \frac{1+\varphi-\varphi^2}{\varphi-1}p_k. \tag{3.10}$$

Note that $\beta q_l \leq p_l$ and $p_l \leq p(G(l))$. From inequality (3.10) and Lemma 1, we have

$$q_l \leq \frac{1}{\beta}p_l \leq \frac{1}{\beta}p(G(l)) < \frac{1+\varphi-\varphi^2}{\beta(\varphi-1)}p_k = p_k. \tag{3.11}$$

We distinguish two possibilities according to where J_0 is scheduled in π.

Case 1.1.1: J_0 is not the first scheduled job in π.

By lemma 3, we have $L_{max}(\sigma) - L_{max}(\pi) \leq q_l$. From inequalities (3.7), (3.8) and (3.11) and Lemma 1, it follows that

$$\frac{L_{max}(\sigma) - L_{max}(\pi)}{L_{max}(\pi)} \leq \frac{q_l}{p(J((S_k(\sigma))) + p(G(l))} \leq \frac{q_l}{\varphi p_k + \beta q_l} < \frac{1}{\varphi+\beta} = \varphi - 1.$$

Case 1.1.2: J_0 is the first scheduled job in π.

We use $B_k(\sigma)$ to denote the set of jobs which arrive at or before time $S_k(\sigma)$ and are scheduled before job J_k in σ. Let $A_l(\sigma)$ denote the set of jobs which arrive at or before time $S_k(\sigma)$ and are scheduled after job J_l in σ. Then we can divide $J(S_k(\sigma))$ into 3 parts, $B_k(\sigma)$, p_k, $A_l(\sigma)$. Therefore, we know

$$S_k(\sigma) = S(B') + p(B_k(\sigma)). \tag{3.12}$$

and $J(S_k(\sigma)) = B_k(\sigma) \bigcup J_k \bigcup A_l(\sigma)$, which deduces that

$$p(J_k(\sigma)) = p(B_k(\sigma)) + p_k + p(A_l(\sigma)). \tag{3.13}$$

Note that $B_k(\sigma)$ and $A_l(\sigma)$ may be empty. Then $p(B_k(\sigma)) \geq 0$ and $P(A_l(\sigma)) \geq 0$. We will consider two subcases divided by whether J_k is scheduled after all jobs in $G(l)$ in π or not.

Case 1.1.2.1: J_k is not scheduled after all jobs in $G(l)$ in π.

Considering that $S(B') = \{(\varphi - 1)p_0, r_1\}$, where r_1 is the arrival time of the second available job, we have

$$S(B') \leq (\varphi - 1)p_0. \tag{3.14}$$

Since it is possible that $J_k \equiv J_0$, we further discuss the following two subcases.

Subcase 1: $J_k \neq J_0$.

It follows that $B_k(\sigma)$ must contain J_0. Therefore, $p(B_k(\sigma)) \geq p_0$. Then we have

$$L_{max}(\pi) \geq p_0 + p_k + p(G(l)) + q_l \geq p_0 + p_k + p_l + q_l. \tag{3.15}$$

Considering equality (3.4) and inequality (3.8), we have

$$L_{max}(\sigma) - L_{max}(\pi) \leq S_k(\sigma) + p_k + q_l - p(J_k(\sigma)).$$

From equalities (3.12) and (3.13), it follows that

$$L_{max}(\sigma) - L_{max}(\pi) \leq S(B') + q_l - p(A_l(\sigma)) \leq S(B') + q_l. \tag{3.16}$$

Therefore, from inequalities (3.16) and (3.14), we have

$$L_{max}(\sigma) - L_{max}(\pi) \leq (\varphi - 1)p_0 + q_l. \tag{3.17}$$

Considering inequalities (3.15), we have

$$\frac{L_{max}(\sigma) - L_{max}(\pi)}{L_{max}(\pi)} \leq \frac{(\varphi - 1)p_0 + q_l}{p_0 + p_k + p_l + q_l} \leq \max\left\{\varphi - 1, \frac{q_l}{p_k + p_l + q_l}\right\}.$$

From inequalities (3.1) and (3.11) and Lemma 1, it follows that

$$\frac{q_l}{p_k + p_l + q_l} < \frac{1}{2 + \beta} < \frac{1}{\varphi + \beta} = \varphi - 1.$$

Therefore, we have

$$\frac{L_{max}(\sigma) - L_{max}(\pi)}{L_{max}(\pi)} \leq \varphi - 1.$$

Subcase 2: $J_k = J_0$.

In this case J_k is the first job in σ, we have $B_k(\sigma) = \emptyset$ and $S_k(\sigma) = S(B')$. Considering inequality (3.14), it follows that $S_k(\sigma) \leq (\varphi - 1)p_0$. From equality (3.4) and $L_{max}(\pi) \geq p_k + p(G(l)) + q_l$, we have

$$L_{max}(\sigma) - L_{max}(\pi) \leq S_k(\sigma) \leq (\varphi - 1)p_0 < (\varphi - 1)L_{max}(\pi).$$

Case 1.1.2.2: J_k is scheduled after all jobs in $G(l)$ in π.

We have

$$L_{max}(\pi) > S_k(\sigma) + p(G(l)) + p_k + q_k. \tag{3.18}$$

Since J_k is not a big job, we have

$$p(J(S_k(\sigma))) \geq \varphi p_k. \tag{3.19}$$

Considering equality (3.4) and inequality (3.18), we have

$$L_{max}(\sigma) - L_{max}(\pi) < q_l - q_k \leq q_l. \tag{3.20}$$

From inequalities (3.8), (3.11), (3.19) and (3.20) and Lemma 1, it follows that

$$\frac{L_{max}(\sigma) - L_{max}(\pi)}{L_{max}(\pi)} < \frac{q_l}{p(J(S_k(\sigma))) + p(G(l))} < \frac{q_l}{\varphi p_k + \beta q_l} < \frac{1}{\varphi + \beta} = \varphi - 1.$$

Case 1.2: J_k is a big job at time $S_k(\sigma)$.

From Lemma 7, we have

$$S_k(\sigma) \geq r_k + (\varphi - 1)p_k \geq (\varphi - 1)p_k. \tag{3.21}$$

We consider the following two subcases divided by whether J_k is scheduled after all jobs in $G(l)$ in π or not.

Case 1.2.1: J_k is scheduled after all jobs in $G(l)$ in π.

If $q_l \geq p_k$, from inequality (3.5), we have

$$\begin{aligned} L_{max}(\pi) &> S_k(\sigma) + p(G(l)) + q_l \geq S_k(\sigma) + p_l + q_l = S_k(\sigma) + (\beta + 1)q_l \\ &\geq S_k(\sigma) + (\beta + 1)p_k. \end{aligned} \tag{3.22}$$

From equality (3.4), inequalities (3.5), (3.21) and (3.22) and Lemma 1, we have

$$\frac{L_{max}(\sigma) - L_{max}(\pi)}{L_{max}(\pi)} < \frac{p_k}{S_k(\sigma) + (\beta + 1)p_k} \leq \frac{p_k}{(\varphi + \beta)p_k} = \varphi - 1.$$

Otherwise, i.e., $q_l < p_k$. Then we have

$$L_{max}(\pi) > S_k(\sigma) + p(G(l)) + p_k + q_k. \tag{3.23}$$

From equation (3.4), inequalities (3.11), (3.21) and (3.23), and Lemma 1, it follows that,

$$\begin{aligned} \frac{L_{max}(\sigma) - L_{max}(\pi)}{L_{max}(\pi)} &< \frac{q_l - q_k}{S_k(\sigma) + p(G(l)) + p_k + q_k} \leq \frac{q_l}{S_k(\sigma) + p_l + p_k} \\ &< \frac{q_l}{\varphi p_k + p_l} < \frac{1}{\varphi + \beta} = \varphi - 1. \end{aligned}$$

Case 1.2.2: J_k is not scheduled after all jobs in $G(l)$ in π.

Recall the definition of $B_k(\sigma)$, we have $B_k(\sigma) \bigcup \{J_k\} = U(J(S_k(\sigma)))$. We know that

$$L_{max}(\pi) \geq p_k + p(G(l)) + q_l. \tag{3.24}$$

If $B_k(\sigma) = \emptyset$, we know

$$S_k(\sigma) = r_k + (\varphi - 1)p_k. \tag{3.25}$$

Considering that J_k is not scheduled after all jobs in $G(l)$ in π, we know that J_l must be scheduled at last in the set $G(l) \bigcup J_k$ in π.

If J_k is scheduled before all jobs in $G(l)$ in π, we have $L_{max}(\pi) \geq r_k + p_k + p(G(l)) + q_l$. Otherwise, i.e., if J_k is not scheduled before all jobs in $G(l)$ in π, there must be $L_{max}(\pi) > r_k + p_k + p(G(l)) + q_l$.

Considering above two situations, we have

$$L_{max}(\pi) > r_k + p_k + p(G(l)) + q_l. \tag{3.26}$$

From equalities (3.4) and (3.25) and inequality (3.26), it follows that

$$L_{max}(\sigma) - L_{max}(\pi) \leq S_k(\sigma) - r_k = (\varphi - 1)p_k < (\varphi - 1)L_{max}(\pi).$$

Then, we suppose that $B_k(\sigma) \neq \emptyset$. So we have that $J_k \neq J_0$. Let J_i be the job completed at time $S_k(\sigma)$ in σ. So we have

$$S_k(\sigma) = S_i(\sigma) + p_i. \tag{3.27}$$

It follows that

$$L_{max}(\sigma) = S_i(\sigma) + p_i + p_k + p(G(l)) + q_l. \tag{3.28}$$

Since J_k is big, from equality (3.13), we have $\varphi p_k > p(B_k(\sigma)) + p_k + p(A_l(\sigma))$. It follows that

$$p(B_k(\sigma)) < (\varphi - 1)p_k - p(A_l(\sigma)) \leq (\varphi - 1)p_k. \tag{3.29}$$

We consider two possibilities distinguished by whether $r_k \leq S_i(\sigma)$ or not.

Case 1.2.2.1: $r_k \leq S_i(\sigma)$.

From inequality (3.21) and equality (3.27), we have $S_i(\sigma)+p_i \geq r_k+(\varphi-1)p_k$. Therefore, $S_i(\sigma) + p_i + p_k \geq r_k + \varphi p_k$. If $S_k(\sigma) = r_k + (\varphi - 1)p_k$, from equality (3.4) and inequality (3.26), we have

$$\frac{L_{max}(\sigma) - L_{max}(\pi)}{L_{max}(\pi)} \leq \frac{S_k(\sigma) - r_k}{r_k + p_k + p(G(l)) + q_l} < \frac{(\varphi - 1)p_k}{p_k} = \varphi - 1.$$

Then we consider the case where $S_k(\sigma) > r_k + (\varphi - 1)p_k$. By the algorithm, Step 3.1 must occur at time $S_i(\sigma)$. Therefore, we have

$$q_i > (\varphi - 1)p_k. \tag{3.30}$$

Then we consider the following two subcases divided by whether J_i is scheduled after all jobs in $G(l)$ in π or not.

Subcase 1: J_i is scheduled after all jobs in $G(l)$ in π.

Since $B_k(\sigma) \neq \emptyset$, we know that $J_0 \neq J_k$. It follows that

$$r_k \geq S(B') \tag{3.31}$$

and that

$$S_k(\sigma) = S(B') + p(B_k(\sigma)). \tag{3.32}$$

In this subcase, J_i is scheduled after all jobs in the set $G(l) \bigcup J_k$ in π. Considering that J_k is not scheduled after all jobs in $G(l)$ in π, we know that J_l must be scheduled at last in the set $G(l) \bigcup J_k$ in π. If $p_i + q_i \geq q_l$, we have

$L_{max}(\pi) \geq r_k + p_k + p(G(l)) + p_i + q_i$. Otherwise, if $p_i + q_i < q_l$, we have $L_{max}(\pi) \geq r_k + p_k + p(G(l)) + q_l$. Therefore, we have

$$L_{max}(\pi) \geq r_k + p_k + p(G(l)) + \max\{p_i + q_i, q_l\}. \tag{3.33}$$

From equalities (3.4) and (3.32), we know

$$L_{max}(\sigma) = S(B') + p(B_k(\sigma)) + p_k + p(G(l)) + q_l. \tag{3.34}$$

From inequalities (3.31) and (3.33) and equality (3.34), it follows that

$$\begin{aligned} L_{max}(\sigma) - L_{max}(\pi) &\leq p(B_k(\sigma)) + q_l - p_i - q_i + (S(B') - r_k) \\ &\leq p(B_k(\sigma)) + q_l - p_i - q_i. \end{aligned} \tag{3.35}$$

Considering inequality (3.30) and $p_i \geq \beta q_i$, we have

$$p_i > \beta(\varphi - 1)p_k. \tag{3.36}$$

From inequalities (3.29), (3.30), (3.35) and (3.36), we have

$$\begin{aligned} L_{max}(\sigma) - L_{max}(\pi) &< (\varphi - 1)p_k + q_l - \beta(\varphi - 1)p_k - (\varphi - 1)p_k \\ &= q_l - \beta(\varphi - 1)p_k < q_l. \end{aligned}$$

From inequality (3.24), we know

$$L_{max}(\pi) \geq p_k + p(G(l)) + q_l \geq p_k + p_l + q_l \geq p_k + (\beta + 1)q_l.$$

Since $\mathcal{I}$ is a counterexample, we must have

$$q_l > (\varphi - 1)L_{max}(\pi) \geq (\varphi - 1)p_k + (\varphi - 1)(\beta + 1)q_l. \tag{3.37}$$

Therefore, Considering Lemma 1, we have

$$p_k < \left(\frac{1}{\varphi - 1} - \beta - 1\right) q_l = (\varphi + \beta - \beta - 1)q_l = (\varphi - 1)q_l. \tag{3.38}$$

From equality (3.4) and inequalities (3.5) and (3.38), we have

$$L_{max}(\sigma) - L_{max}(\pi) < p_k < (\varphi - 1)q_l < (\varphi - 1)L_{max}(\pi).$$

Subcase 2: J_i is not scheduled after all job in $G(l)$ in π.
We have

$$L_{max}(\pi) \geq p_k + p_i + p(G(l)) + q_l. \tag{3.39}$$

Since $B_k(\sigma) \neq \emptyset$, we know that $J_0 \neq J_k$. It follows that

$$L_{max}(\sigma) = S(B') + p(B_k(\sigma)) + p_k + p(G(l)) + q_l \tag{3.40}$$

and that

$$L_{max}(\pi) > p_0 + p_k. \tag{3.41}$$

From inequalities (3.39) and (3.40), we have

$$L_{max}(\sigma) - L_{max}(\pi) \leq S(B') + p(B_k(\sigma)) - p_i \leq S(B') + p(B_k(\sigma)). \tag{3.42}$$

Since $S(B') = \min\{(\varphi-1)p_0, r_1\}$, we have

$$S(B') \leq (\varphi-1)p_0. \tag{3.43}$$

From inequalities (3.29), (3.41), (3.42) and (3.43), it follows that

$$L_{max}(\sigma) - L_{max}(\pi) < (\varphi-1)p_0 + (\varphi-1)p_k < (\varphi-1)(p_0+p_k) < (\varphi-1)L_{max}(\pi).$$

Case 1.2.2.2: $r_k > S_i(\sigma)$.

We know that

$$L_{max}(\pi) \geq r_k + p_k + p(G(l)) + q_l. \tag{3.44}$$

From equality (3.28) and inequality (3.44), we have

$$L_{max}(\sigma) - L_{max}(\pi) \leq S_i(\sigma) + p_i - r_k < p_i \leq p(B_k(\sigma)). \tag{3.45}$$

From inequalities (3.29) and (3.45), it follows that

$$L_{max}(\sigma) - L_{max}(\pi) < p(B_k(\sigma)) < (\varphi-1)p_k < (\varphi-1)L_{max}(\pi).$$

Case 2: There is a big job $J_i \in G(l)$ available at time $S_k(\sigma)$ and algorithm EX-D-LDT is not allowed to schedule it at time $S_k(\sigma)$. This corresponds to Step 3.4. (J_k is not a big job.)

We claim that J_i must be the only job from $G(l)$ available at time $S_k(\sigma)$. Suppose to the contrary that there is another job J_j from $G(l)$ except J_i available at time $S_k(\sigma)$. According to the definition of $G(l)$, we have $q_k < \min\{q_i, q_j\}$. By Step 3.3 and 3.4 of the algorithm, J_k must not be selected to schedule at time $S_k(\sigma)$. There is a contradiction.

Therefore, we have that

$$L_{max}(\pi) \geq r_i + p(G(l)) + q_l. \tag{3.46}$$

By the above claim, we know that J_i is both the job with largest delivery time and the job with largest processing time at time $S_k(\sigma)$. So, we know the index $p_{max}(S_k(\sigma)) = q_{max}(S_k(\sigma))$. Therefore, Step 3.4 of the algorithm must occur at time $S_k(\sigma)$. This implies that $S_k(\sigma) + p_k + p_i \leq r_i + \varphi p_i$. Therefore, we have

$$S_k(\sigma) + p_k \leq r_i + (\varphi-1)p_i. \tag{3.47}$$

From equality (3.4) and inequalities (3.46) and (3.47) we have

$$\begin{aligned} L_{max}(\sigma) - L_{max}(\pi) &\leq S_k(\sigma) + p_k - r_i \leq r_i + (\varphi-1)p_i - r_i = (\varphi-1)p_i \\ &\leq (\varphi-1)L_{max}(\pi). \end{aligned}$$

Case 3: J_k is a big job at time $S_k(\sigma)$, and all of the jobs from $G(l)$ that are available at time $S_k(\sigma)$ have a delivery time of at most $(\varphi-1)p_k$. This corresponds to Step 3.2.

Since J_l is the job with minimum delivery time in $G(l)$, we have

$$q_l \leq (\varphi-1)p_k. \tag{3.48}$$

Consider the following two cases.

Case 3.1: J_0 is not the first job in π.

From Lemma 3 and inequality (3.48), we have that

$$L_{max}(\sigma) - L_{max}(\pi) \leq q_l \leq (\varphi - 1)p_k < (\varphi - 1)L_{max}(\pi).$$

Case 3.2: J_0 is the first job in π.
From Lemma 3 and inequality (3.48), we have

$$L_{max}(\sigma) - L_{max}(\pi) \leq (\varphi - 1)p_0 + q_l \leq (\varphi - 1)(p_0 + p_k). \tag{3.49}$$

If $J_0 \neq J_k$, we have $L_{max}(\pi) > p_0 + p_k$. Therefore, from inequality (3.49), it follows that

$$L_{max}(\sigma) - L_{max}(\pi) \leq (\varphi - 1)(p_0 + p_k) < (\varphi - 1)L_{max}(\pi).$$

If $J_0 = J_k$, then J_k is the first job in π. therefore, we have

$$L_{max}(\pi) \geq p_k + p(G(l)) + q_l. \tag{3.50}$$

Therefore, from equality (3.4), we have

$$L_{max}(\sigma) - L_{max}(\pi) \leq S_k(\sigma). \tag{3.51}$$

Since $\mathcal{I}$ is a counterexample, we must have $S_k(\sigma) > (\varphi - 1)p_k$. (Note that $r_k = 0$.) This implies that the algorithm must run Step 3.1 at least once. Let T be the starting time of the first job completed after time $(\varphi - 1)p_k$. Let Q denote the set of jobs processed from time T to time $S_k(\sigma)$ in σ. So we know that

$$T \leq (\varphi - 1)p_k \tag{3.52}$$

and $Q \bigcap G(l) = \emptyset$. Since J_k is a big job and the jobs in Q are scheduled before J_k in σ, from Step 3.1, we have that each job in Q has a delivery time more than $(\varphi - 1)p_k$. From inequality (3.48), $q_l \leq (\varphi - 1)p_k$, we know that each job in $Q \bigcup G(l)$ has a delivery time at least q_l. Therefore, we have

$$L_{max}(\pi) \geq p_k + p(Q) + p(G(l)) + \min_{J_j \in Q \bigcup G(l)} q_j = p_k + p(Q) + p(G(l)) + q_l. \tag{3.53}$$

We also know that

$$L_{max}(\sigma) = T + p(Q) + p_k + p(G(l)) + q_l. \tag{3.54}$$

From equalities (3.52), (3.53) and (3.54), it follows that

$$L_{max}(\sigma) - L_{max}(\pi) \leq T \leq (\varphi - 1)p_k < (\varphi - 1)L_{max}(\pi).$$

The theorem follows. ■

Chapter 4

Two identical parallel machine model

We study the problem of semi-online scheduling on 2 machines under a grade of service (GoS). GoS means that some jobs have to be processed by some machines to be guaranteed a high quality. The problem is online in the sense that jobs are presented one by one, and each job shall be assigned to a time slot on its arrival. Assume that the processing time p_i of every job J_i is bounded by an interval $[a, \alpha a]$, where $a > 0$ and $\alpha > 1$ are two constant numbers. By knowing the bound of jobs' processing times, we denote it by semi-online problem. We deal with two semi-online problems.

The first one concerns about bounded processing time constraint. First, we show that a lower bound of competitive ratio is: (1) $\frac{1+\alpha}{2}$ in the case where $1 < \alpha < 2$; (2) $\frac{3}{2}$ in the case where $2 \leq \alpha < 5$; and (3) $\frac{4+\alpha}{6}$ in the case where $5 \leq \alpha < 6$. We further propose an algorithm, called *B-ONLINE*, and prove that in the case where $\frac{25}{14} \leq \alpha$ and the optimal makespan $C_{OPT} \geq 20a$, *B-ONLINE* algorithm is optimal.

For the second problem, we further know the sum of jobs' processing times Σ in advance. We first show a lower bound $\frac{1+\alpha}{2}$ in the case where $1 < \alpha < 2$, then we propose an algorithm *B-SUM-ONLINE* which is optimal in the case where $\Sigma \geq \frac{2\alpha}{\alpha-1}a$ and $1 < \alpha < 2$.

4.1 Introduction

Grade of service (GoS) is a qualitative concept, and it's often translated into the level of access privilege of different service provision. For example, suppose we have 2 machines (or processors). One of them can provide high quality service (or high GoS) while the other one provides normal service (or low GoS). Some jobs which request high quality must be processed by high GoS machine, while other jobs with low quality requests can be processed by both machines whenever they are available. For more recent development on GoS, see [55]. The problem is *online* in the sense that when receiving a job and before the next job is presented, we must irrevocably assign the job to a time slot of the schedule. The assignment of each job shall be made on its arrival and the next job arrives immediately after the assignment, i.e., the difference of arrival

times of two continual jobs is ignorable. Preemption and re-assignment are not allowed. The objective is to minimize the makespan, that is the completion time of the last job in the schedule. When all information is available at one time before scheduling, the problem is called *offline*. We call a problem semi-online if we know some information of jobs in advance, i.e., jobs' total processing time.

Hwang et al. [55] first study the (offline) problem of parallel machine scheduling with GoS eligibility. They proposed an approximation algorithm LG-LPT, and proved that its makespan is not greater than $\frac{5}{4}$ times the optimal makespan for $m = 2$ and not greater than $2 - \frac{1}{m-1}$ times the optimal makespan for $m \geq 3$. However, online scheduling under GoS eligibility was first studied by Park et al. [82] and Jiang et al. [**?**]. For the problem of online scheduling on 2 machines with GoS constraint, they respectively proposed an optimal algorithm with a competitive ratio of $\frac{5}{3}$. Jiang [57] further investigated the problem of online scheduling on parallel machines with two GoS levels. He assumed that the number of machines providing high GoS is not known before scheduling and decisions must be made without knowledge of the exact number of machines providing high GoS. I.e., we only know that in 10 parallel machines there are k ($1 \leq k \leq 9$) machines which can provide high GoS. Under this consideration, he proved that 2 is a lower bound of online algorithms and proposed an online algorithm with a competitive ratio of $\frac{12+4\sqrt{2}}{7}$.

He et al. [46] investigated two different semi on-line scheduling problems on a two-machine system. In the first problem, they assumed that all jobs have their processing times in between p and rp ($p > 0$, $r \geq 1$). They showed that LS is optimal with a competitive ratio $(r + 1)/2$ in the case where $1 \leq r < 2$ and $3/2$ in the case where $r \geq 2$. In the second problem, they supposed that the largest processing time is known in advance. They showed that PLS algorithm is optimal with a competitive ratio $4/3$.

In this chapter we will consider semi-online scheduling on two machines under a grade of service provision with jobs' processing times bounded by an interval $[a, \alpha a]$, where $a > 0$ and $\alpha > 1$ are two constant numbers. For simplicity, we use online algorithm to denote semi-algorithm in the remainder.

The rest of this chapter is organized as follows. In Section 2.1, we will describe the problem and introduce some basic notations. Section 2.2 will show some lower bounds of competitive ratio considering different values of α. In section 2.3, we will propose an algorithm *B-ONLINE*, and prove that in the case where $\frac{25}{14} \leq \alpha$ and the optimal makespan $C_{OPT} \geq 20a$, *B-ONLINE* is optimal. In section 3.1, we will give some problem definitions and notations. Section 3.2 will present a lower bound of competitive ratio. In section 3.3, we will show an algorithm *B-SUM-ONLINE* and prove that it is optimal in the case $\Sigma \geq \frac{2\alpha}{\alpha-1}a$ and $1 < \alpha < 2$.

4.2 Online scheduling on 2 machines with GoS and bounded processing times

In this section, we study the problem of online scheduling on 2 machines under a grade of service provision with bounded processing times.

4.2.1 Problem definitions and notations

We are given 2 machines with speed of 1. Without loss of generality, we denote the one that can provide both high and low GoSs by M_1, and the other one that only provides low GoS by M_2. We denote each job by $J_i = (p_i, g_i)$, where p_i is the processing time of J_i and $g_i \in \{1, 2\}$ is the GoS of the job. $g_i = 1$ if J_i must be processed by M_1 and $g_i = 2$ if it can be processed by either of the two machines. p_i and g_i are not known unless J_i arrives. A sequence of jobs $\sigma = \{J_1, J_2, ..., J_n\}$ which arrive online have to be scheduled irrevocably on one of the two machines on their arrivals. Each job J_i is presented immediately after J_{i-1} is scheduled. The schedule can be seen as a partition of job sequence σ into two subsequences, denoted by S_1 and S_2, where S_1 and S_2 consist of jobs assigned to M_1 and M_2, respectively. Let $L_1 = t(S_1) = \sum_{J_i \in S_1} p_i$ and $L_2 = t(S_2) = \sum_{J_i \in S_2} p_i$ denote the loads (or total processing times) of M_1 and M_2, respectively. Hence, the makespan of one schedule is $max\{L_1, L_2\}$. The online problem can be written as:

Given σ, find S_1 and S_2 to minimize $max\{L_1, L_2\}$.

Let C_{ON} and C_{OPT} denote the makespan of online algorithm and offline optimal algorithm (for short, offline algorithm), respectively.

4.2.2 Lower bounds of competitive ratio

In this section, we will show some lower bounds of the competitive ratio of online algorithms for different values of α. If $\alpha = 1$, online algorithms can reach the optimal makespan. An algorithm is optimal if it assigns the jobs as many as possible on M_2 when the difference of the loads of two machines is not greater than 1. Moreover, for the case where $\alpha \geq 6$, the lower bound of $\frac{5}{3}$ has been proved in [82]. So, we will focus on the case where $1 < \alpha < 6$.

Theorem 3 *For the problem of online scheduling on two machines under GoS constraint with jobs' processing times bounded within interval $[a, \alpha a]$, there exists no algorithm with a competitive ratio less than: (1) $\frac{1+\alpha}{2}$ in the case where $1 < \alpha < 2$; (2) $\frac{3}{2}$ in the case where $2 \leq \alpha < 5$; and (3) $\frac{4+\alpha}{6}$ in the case where $5 \leq \alpha < 6$.*

Proof.

Without loss of generality, let $a = 1$. We will discuss the three cases of α in the following.

Case 1. $1 < \alpha < 2$.

Let $\varphi = \frac{1+\alpha}{2}$. We will generate a job sequence ρ consisting of at most 3 jobs, which arrive one by one. Once the ratio of makespans between online and offline algorithms is at least φ after some job is assigned, no more jobs will be presented and we stop. We begin with job $J_1 = (1, 2)$. If online algorithm assigns J_1 to M_1, we further generate job $J_2 = (1, 1)$. Since offline algorithm will assign J_1 to M_2, $\frac{C_{ON}}{C_{OPT}} = 2 > \varphi$ and we stop. Otherwise if online algorithm assigns J_1 to M_2, we generate job $J_2 = (1, 2)$. If J_2 is assigned to M_2, we have $\frac{C_{ON}}{C_{OPT}} = 2 > \varphi$. Otherwise if J_2 is assigned to M_1, we generate job $J_3 = (\alpha, 1)$. We have $\frac{C_{ON}}{C_{OPT}} = \frac{1+\alpha}{2} = \varphi$ in this case.

Case 2. $2 \leq \alpha < 5$.

Similar to the analysis in Case 1, we begin with jobs $J_1 = J_2 = (1, 2)$. If online algorithm assigns both of them to one of the two machines, we have

$\frac{C_{ON}}{C_{OPT}} = 2 > \frac{3}{2}$. Otherwise, we further generate job $J_3 = (2, 1)$. Then we have $C_{ON} = 3$ and $C_{OPT} = 2$, since the optimal solution consists of scheduling J_1, J_2 on M_2 and J_3 on M_1. It follows that $\frac{C_{ON}}{C_{OPT}} = \frac{3}{2}$.

Case 3. $5 \leq \alpha < 6$.

Let $\varphi = \frac{4+\alpha}{6}$. We will generate a job sequence which consists of at most 5 jobs in this case. Similarly, we begin with jobs $J_1 = J_2 = (1, 2)$ and observe the behavior of online algorithm. If both of them are assigned to one of the two machines, $\frac{C_{ON}}{C_{OPT}} = 2 > \varphi$ and we stop. Otherwise, we further generate job $J_3 = (1, 2)$. If J_3 is assigned to M_1, we give job $J_4 = (3, 1)$ and then $\frac{C_{ON}}{C_{OPT}} = \frac{1+1+3}{3} = \frac{5}{3} > \varphi$. Otherwise if J_3 is assigned to M_2, we further generate job $J_4 = (3, 2)$. If J_4 is assigned to M_2, it follows that $\frac{C_{ON}}{C_{OPT}} = \frac{1+1+3}{3} = \frac{5}{3} > \varphi$. Otherwise if J_4 is assigned to M_1, we give the last job $J_5 = (\alpha, 1)$, and then $\frac{C_{ON}}{C_{OPT}} = \frac{1+3+\alpha}{6} = \frac{4+\alpha}{6} = \varphi$.

Therefore, the theorem follows. ■

Remark 3 *In the case where $\alpha \geq 6$, the tight lower bound of competitive ratio is $\frac{5}{3}$ (see [82]).*

4.2.3 *B-ONLINE* algorithm

Combining the *ONLINE* algorithm proposed in [82] with bounded jobs' processing time of job, we will give a modified algorithm called *B-ONLINE*. Before describing the algorithm, we give some notations on *B-ONLINE*'s schedules as follows.

At the arrival of each job, P and T are updated to become the maximum processing time and a half of total processing times of all arrived jobs, respectively. D is updated to be the total processing time of all arrived jobs with $g_i = 1$. Let $L = max\{T, P, D\}$. Thus, we have $C_{OPT} \geq L$. For analysis convenience, we define $P^i, T^i, D^i, L^i, S_1^i$ and S_2^i to be the P, T, D, L, S_1 and S_2 respectively immediately after job J_i is assigned, and let $P^0 = T^0 = D^0 = L^0 = 0$ and $S_1^0 = S_2^0 = \emptyset$.

According to Theorem 1 on lower bounds, we define various values of parameter φ as follows. (1) $\varphi = \frac{1+\alpha}{2}$ in the case where $\frac{25}{14} \leq \alpha \leq 2$; (2) $\varphi = \frac{3}{2}$ in the case where $2 \leq \alpha \leq 5$; (3) $\varphi = \frac{4+\alpha}{6}$ in the case where $5 \leq \alpha < 6$; and (4) $\varphi = \frac{5}{3}$ in the case where $\alpha \geq 6$.

B-ONLINE behaves as follows:

Step 1: Let $S_1 = \emptyset, S_2 = \emptyset, P = 0, T = 0, D = 0$;

Step 2: Receive job $J_i = (p_i, g_i)$. $P = max\{P, p_i\}$ and $T = T + \frac{p_i}{2}$;

Step 3: If $g_i = 1$, let $S_1 = S_1 \bigcup \{J_i\}$ and $D = D + p_i$. Go to **Step 5**;

Step 4: Let $L = max\{T, D, P\}$, if $t(S_2) + p_i \leq \varphi L$, let $S_2 = S_2 \bigcup \{J_i\}$; else , let $S_1 = S_1 \bigcup \{J_i\}$;

Step 5: If no more jobs arrive, stop and output S_1 and S_2; Else, let $i = i + 1$ and go to **Step 2**.

B-ONLINE has the same performance in competitiveness as *ONLINE* in the case where $\alpha \geq 6$, i.e., both of them have a competitive ratio of $\frac{5}{3}$ in the case (please refer to [82] for details). So, we will focus our attention on the case where $\frac{25}{14} \leq \alpha < 6$ later on.

Lemma 8 *Given a constant number a, in the case where $\frac{25}{14} \leq \alpha < 6$, if job $J_i = (p_i, 2)$ is scheduled on M_1 by* B-ONLINE *algorithm, there must be $t(S_1^i) <$*

$\frac{2-\varphi}{\varphi}t(S_2^i) + \frac{2\alpha a}{\varphi}$, *where (1)* $\varphi = \frac{1+\alpha}{2}$ *in the case where* $\frac{25}{14} \leq \alpha \leq 2$*; (2)* $\varphi = \frac{3}{2}$ *in the case where* $2 \leq \alpha \leq 5$*; (3)* $\varphi = \frac{4+\alpha}{6}$ *in the case where* $5 \leq \alpha < 6$.

Proof. If $J_i = (p_i, 2)$ is scheduled on M_1, there must be $t(S_2^{i-1}) + p_i > \varphi L^i$. Since $t(S_2^i) = t(S_2^{i-1})$ and $p_i \in [a, \alpha a]$, we have $t(S_2^i) = t(S_2^{i-1}) > \varphi L^i - \alpha a$. In another aspect, because $L^i \geq T^i = \frac{1}{2}[t(S_1^i) + t(S_2^i)]$, we have

$$t(S_2^i) > \frac{\varphi}{2}[t(S_1^i) + t(S_2^i)] - \alpha a. \tag{4.1}$$

It follows that $t(S_1^i) < \frac{2-\varphi}{\varphi}t(S_2^i) + \frac{2\alpha a}{\varphi}$. The lemma follows. ■

Theorem 4 *Given a constant number a, in the case where* $\frac{25}{14} \leq \alpha < 6$ *and* $C_{OPT} \geq 20a$ *,* B-ONLINE *is optimal with a competitive ratio* φ *such that (1)* $\varphi = \frac{1+\alpha}{2}$ *in the case where* $\frac{25}{14} \leq \alpha \leq 2$*; (2)* $\varphi = \frac{3}{2}$ *in the case where* $2 \leq \alpha \leq 5$*; (3)* $\varphi = \frac{4+\alpha}{6}$ *in the case where* $5 \leq \alpha < 6$.

Proof. We define C_{ON}^i and C_{OPT}^i to be C_{ON} and C_{OPT} respectively immediately after job J_i is scheduled. Thus, $C_{OPT}^i \geq C_{OPT}^{i-1}$ for $i \geq 1$.

We first assume that the theorem is false, implying that there must exist at least one instance, called *counter example*, which derives $\frac{C_{ON}}{C_{OPT}} > \varphi$. Among all such counter examples, let ς be the one with the least number n of jobs, called *minimal counter example*. By the definition of minimal counter example, the makespan of ς is not determined until the arrival of job J_n. Therefore,

$$C_{ON}^n = max\{t(S_1^n), t(S_2^n)\} > \varphi C_{OPT}^n; \tag{4.2}$$

$$max\{t(S_1^{n-1}), t(S_2^{n-1})\} \leq \varphi C_{OPT}^{n-1}. \tag{4.3}$$

Case 1. $g_n = 2$.
If $J_n = (p_n, 2)$ is scheduled on M_2, then $t(S_2^{n-1}) + p_n \leq \varphi L^n \leq \varphi C_{OPT}^n$. This implies that $\varphi C_{OPT}^n < C_{ON}^n = t(S_1^n) = t(S_1^{n-1}) \leq \varphi C_{OPT}^{n-1}$. Since $C_{OPT}^n \geq C_{OPT}^{n-1}$, There is a contradiction. Therefore, J_n must be assigned to M_1 and we have

$$t(S_2^{n-1}) + p_n > \varphi L^n. \tag{4.4}$$

Since $T^n = \frac{1}{2}[t(S_1^{n-1}) + t(S_2^{n-1}) + p_n] \leq L^n$, together with inequality (4.4), we have $t(S_1^{n-1}) < (2-\varphi)L^n$. Since $L^n \leq C_{OPT}^n$, we have $t(S_1^{n-1}) < (2-\varphi)C_{OPT}^n$. Considering $p_n \in [a, \alpha a]$, it follows $C_{ON}^n = t(S_1^n) = t(S_1^{n-1}) + p_n < (2 - \varphi)C_{OPT}^n + \alpha a$. Since $C_{OPT}^n \geq 20a$, we have

$$\frac{C_{ON}^n}{C_{OPT}^n} = (2-\varphi) + \frac{\alpha a}{C_{OPT}^n} \leq 2 - \varphi + \frac{\alpha}{20}.$$

To prove the theorem, we need to derive a contradiction to the assumption, i.e., to prove $\frac{C_{ON}^n}{C_{OPT}^n} \leq \varphi$. That means $2 - \varphi + \frac{\alpha}{20} \leq \varphi$ or $2 + \frac{\alpha}{20} - 2\varphi \leq 0$.

Case 1.1. $\frac{25}{14} \leq \alpha \leq 2$.
In this subcase, $\varphi = \frac{1+\alpha}{2}$, and then

$$2 + \frac{\alpha}{20} - 2\varphi = 2 + \frac{\alpha}{20} - 1 - \alpha = 1 - \frac{19\alpha}{20} < 0.$$

It contradicts to the assumption.

Case 1.2. $2 \leq \alpha \leq 5$.
In this subcase, $\varphi = \frac{3}{2}$ and then

$$2 + \frac{\alpha}{20} - 2\varphi = 2 + \frac{\alpha}{20} - 3 = \frac{\alpha}{20} - 1 < -\frac{3}{4} < 0.$$

Again, there is a contradiction to the assumption.

Case 1.3. $5 \leq \alpha < 6$.
In this subcase, $\varphi = \frac{4+\alpha}{6}$.

$$2 + \frac{\alpha}{20} - 2\varphi = 2 + \frac{\alpha}{20} - \frac{4+\alpha}{3} = \frac{40 - 17\alpha}{60} \leq -\frac{3}{4} < 0.$$

Again, there is a contradiction.

Case 2. $g_n = 1$.
According to the definition of minimal counter example,

$$C_{ON}^n = t(S_1^n) > \varphi C_{OPT}^n \tag{4.5}$$

Since C_{OPT}^n is at least the sum of processing times of jobs with $g_i = 1$, inequality (4.5) means that S_1 must contain at least one job, named J_k, with $g_k = 2$. Let J_K be the last job with $g_K = 2$ assigned to M_1. Let $A_K(S_1)$ be the set of jobs assigned to M_1 after J_K has been assigned to M_1. Thus, $t(S_1^n) = t(S_1^k) + t(A_K(S_1))$. Since C_{OPT}^n cannot be less than the sum of processing times of jobs with $g_i = 1$, we have $t(A_K(S_1)) \leq C_{OPT}^n$. Therefore, $t(S_1^n) \leq t(S_1^k) + C_{OPT}^n$. Together with Lemma 8,

$$t(S_1^n) \leq \frac{2-\varphi}{\varphi} t(S_2^k) + \frac{2\alpha a}{\varphi} + C_{OPT}^n \tag{4.6}$$

Since the total processing time of jobs doesn't vary among different algorithms, we have $t(S_1^n) + t(S_2^n) \leq 2C_{OPT}^n$. Considering inequality (4.5), we have $t(S_2^n) < (2-\varphi)C_{OPT}^n$. Since $t(S_2^k) \leq t(S_2^n)$, together with inequality (4.6), we have

$$t(S_1^n) < \left(\frac{(2-\varphi)^2}{\varphi} + 1 \right) C_{OPT}^n + \frac{2\alpha a}{\varphi}$$

Therefore,

$$\frac{C_{ON}^n}{C_{OPT}^n} < \frac{(2-\varphi)^2}{\varphi} + 1 + \frac{2\alpha a}{\varphi C_{OPT}^n} \leq \frac{(2-\varphi)^2}{\varphi} + 1 + \frac{\alpha}{10\varphi}.$$

Similar to Case 1, we need to derive a contradiction to the assumption to prove the theorem, i.e., to prove $\frac{C_{ON}^n}{C_{OPT}^n} \leq \varphi$. That means $\frac{(2-\varphi)^2}{\varphi} + 1 + \frac{\alpha}{10\varphi} \leq \varphi$ or $4 + \frac{\alpha}{10} - 3\varphi \leq 0$.

Case 2.1. $\frac{25}{14} \leq \alpha \leq 2$.
In this subcase, $\varphi = \frac{1+\alpha}{2}$. Therefore,

$$4 + \frac{\alpha}{10} - 3\varphi = 4 + \frac{\alpha}{10} - \frac{3(1+\alpha)}{2} = \frac{25 - 14\alpha}{10} \leq 0.$$

It contradicts to the assumption.

Case 2.2. $2 \leq \alpha \leq 5$.
In this subcase, $\varphi = \frac{3}{2}$, and then

$$4 + \frac{\alpha}{10} - 3\varphi = 4 + \frac{\alpha}{10} - \frac{9}{2} = \frac{\alpha - 5}{10} < 0.$$

Again, there is a contradiction.

Case 2.3. $5 \leq \alpha < 6$.
In this subcase, $\varphi = \frac{4+\alpha}{6}$ and thus

$$4 + \frac{\alpha}{10} - 3\varphi = 4 + \frac{\alpha}{10} - \frac{4+\alpha}{2} = \frac{2(5-\alpha)}{5} \leq 0.$$

There is a contradiction to the assumption.

According to the analysis in Cases 1 and 2, the theorem follows. ■

4.3 Online scheduling on 2 machines with the total processing time

In this section, we further know jobs' total processing time in advance. We also study the problem of online scheduling on 2 machines with GoS and bounded processing times.

4.3.1 Problem definitions and notations

We are given 2 machines with speed of 1 and jobs' total processing time Σ. Without loss of generality, we denote the machine that can provide both high and low GoSs by M_1, and the other one that only provides low GoS by M_2. We denote each job by $J_i = (p_i, g_i)$, where p_i is the processing time of J_i and $g_i \in \{1, 2\}$ is the GoS of the job. $g_i = 1$ if J_i must be processed by M_1 and $g_i = 2$ if it can be processed by either of the two machines. p_i and g_i are not known unless J_i arrives. A sequence of jobs $\sigma = \{J_1, J_2, ..., J_n\}$ which arrive online have to be scheduled irrevocably on one of the two machines on their arrivals. Each job J_i is presented immediately after J_{i-1} is scheduled. The schedule can be seen as a partition of job sequence σ into two subsequences, denoted by S_1 and S_2, where S_1 and S_2 consist of jobs assigned to M_1 and M_2, respectively. Note that $\Sigma = t(S_1) + t(S_2)$. Let $L_1 = t(S_1) = \sum_{J_i \in S_1} p_i$ and $L_2 = t(S_2) = \sum_{J_i \in S_2} p_i$ denote the loads (or total processing times) of M_1 and M_2, respectively. Hence, the makespan of one schedule is $max\{L_1, L_2\}$. The online problem can be written as:

Given σ, find S_1 and S_2 to minimize $max\{L_1, L_2\}$.

Let C_{S-ON} and C_{OPT} denote the makespan of online algorithm and offline optimal algorithm (for short, offline algorithm), respectively.

4.3.2 Lower bounds of competitive ratio

In this section, we will show some lower bounds of competitive ratio of online algorithm considering difference values of α. If $\alpha = 1$, online algorithms can reach the optimal makespan. I.e., one algorithm is optimal if it assigns the jobs as many as possible on M_2 when the difference of the loads of two machines is

not greater than 1. Moreover, for the case where $\alpha \geq 2$, the lower bound of $\frac{3}{2}$ has been proved in [82]. So, we will focus on the case where $1 < \alpha < 2$.

Theorem 5 *For the problem of online scheduling on two machines under GoS constraint with (1) knowledge of total processing time and (2) jobs' processing times bounded within interval $[a, \alpha a]$, there exists no online algorithm with a competitive ratio less than $\frac{1+\alpha}{2}$ in the case where $1 < \alpha < 2$.*

Proof. Without loss of generality, let $a = 1$ and the sum of jobs' processing times be $2 + \alpha$. We will generate a job sequence which consists of at most 3 jobs, which arrive one by one. Once the ratio of makespans between online and offline algorithms is at least $1 + \frac{1}{\alpha}$ after some job is assigned, no more jobs will be presented and we stop. We begin with job $J_1 = (1, 2)$. If online algorithm assigns J_1 to M_1, we further generate job $J_2 = (\alpha, 1)$ and $J_3 = (1, 2)$. Since offline algorithm will assign J_1, J_3 to M_2 and J_2 to M_1, it follows $\frac{C_{ON}}{C_{OPT}} \geq \frac{1+\alpha}{2}$ and we stop. Otherwise if online algorithm assigns J_1 to M_2, we generate job $J_2 = (\alpha, 2)$ and $J_3 = (1, 1)$. If J_2 is assigned to M_2, since offline algorithm will scheduling J_1, J_3 on M_1 and J_2 on M_2, we have $\frac{C_{ON}}{C_{OPT}} \geq \frac{1+\alpha}{2}$. Otherwise if J_2 is assigned to M_1, we also generate job $J_3 = (1, 1)$. We have $\frac{C_{ON}}{C_{OPT}} = \frac{1+\alpha}{2}$ in this case, since offline algorithm will assign J_1, J_3 on M_1 and J_2 on M_2. ■

Remark 4 *In the case where $\alpha \geq 2$, the tight lower bound of competitive ratio is $\frac{3}{2}$ (see [82]).*

4.3.3 *B-SUM-ONLINE* algorithm

Combining the *SEMI-ONLINE* algorithm proposed in [82] with bounded jobs' processing times, we will give a modified algorithm called *B-SUM-ONLINE*. Before describing the algorithm, we give some notations on *B-SUM-ONLINE*'s schedules as follows.

Let Σ be the sum of jobs' processing times. Let $L = \frac{\Sigma}{2}$. Thus, we have $C_{OPT} \geq L$. *B-SUM-ONLINE* behaves as follows:

Step 1: Let $S_1 = \emptyset, S_2 = \emptyset$;
Step 2: Receive job $J_i = (p_i, g_i)$;
Step 3: If $g_i = 1$, let $S_1 = S_1 \bigcup \{J_i\}$. Go to **Step 5**;
Step 4: If $t(S_2) + p_i \leq \frac{1+\alpha}{2} L$, let $S_2 = S_2 \bigcup \{J_i\}$; else , let $S_1 = S_1 \bigcup \{J_i\}$;
Step 5: If no more jobs arrive, stop and output S_1 and S_2; Else, let $i = i + 1$ and go to **Step 2**.

B-SUM-ONLINE has the same performance in competitiveness as *SEMI-ONLINE* in the case where $\alpha \geq 2$, i.e., both of them have competitive ratio of $\frac{3}{2}$ in the case (please refer to [82] for details). So, we will focus our attention on the case where $1 < \alpha < 2$ later on.

The proof of the competitive ratio of B-SUM-ONLINE algorithm is by contradiction. We assume that there exists a job instance, called counter example, for which B-SUM-ONLINE algorithm yields a schedule with makespan greater than $\frac{1+\alpha}{2}$ times of the optimum. We further define the counter example with the least number of jobs as minimal counter example. For a minimal counter example $\mathcal{I}$, let $\mathcal{J}_1 = \{J_i | g_i = 1, J_i \in \mathcal{I}\}$ and $\mathcal{J}_2 = \{J_i | g_i = 2, J_i \in \mathcal{I}\}$. We use σ to denote the schedule generated by B-SUM-ONLINE algorithm for $\mathcal{I}$. Let C_{S-ON} and C_{OPT} denote the makespan of B-SUM-ONLINE algorithm and the

optimal algorithm for $\mathcal{I}$, respectively. We define S_1^j and S_2^j to be S_1 and S_2 that we have immediately after we schedule job J_j. $S_1^0 = S_2^0 = \emptyset$.

Lemma 9 $t(\mathcal{J}_2) > \frac{1+\alpha}{2}L$.

Proof. Suppose $t(\mathcal{J}_2) \leq \frac{1+\alpha}{2}L$. For any job $J_j = (p_j, 2)$, since $t(S_2^{j-1}) + p_j \leq t(\mathcal{J}_2)$ and $t(\mathcal{J}_2) \leq \frac{1+\alpha}{2}L$, we have $t(S_2^{j-1}) + p_j \leq \frac{1+\alpha}{2}L$. From Step 4 of the algorithm, we get $S_1 = \mathcal{J}_1$ and $S_2 = \mathcal{J}_2$. Since $t(S_1) = t(\mathcal{J}_1) \leq C_{OPT}$, we know $C_{S-ON} \neq t(S_1)$. Therefore, $C_{S-ON} = t(S_2) = t(\mathcal{J}_2) \leq \frac{1+\alpha}{2}L \leq \frac{1+\alpha}{2}C_{OPT}$. There exists a contradiction. ■

Theorem 6 *Given $\Sigma \geq \frac{2\alpha}{\alpha-1}a$, B-SUM-ONLINE algorithm is optimal with a competitive ratio of $\frac{1+\alpha}{2}$ in the case $1 < \alpha < 2$.*

Proof. Let $\varphi = \frac{1+\alpha}{2}$. We assume that the theorem is false and there exists a minimal counter example $\mathcal{I} = \{J_1, ..., J_n\}$. Therefore, we have

$$C_{S-ON}^n = \max\{t(S_1^n), t(S_2^n)\} > \varphi C_{OPT}^n, \tag{4.7}$$

$$\max\{t(S_1^{n-1}), t(S_2^{n-1})\} \leq \varphi C_{OPT}^{n-1}. \tag{4.8}$$

Our aim is to prove for this instance, $\frac{C_{S-ON}^n}{C_{OPT}^n} \leq \varphi$ holds.

Case 1. $g_n = 2$.

If J_n is assigned to M_2, we have $t(S_2^{n-1}) + p_n \leq \varphi L \leq \varphi C_{OPT}^n$ and $t(S_1^{n-1}) = t(S_1^n)$. By inequality (4.7), it follows that $t(S_1^{n-1}) > \varphi C_{OPT}^n \geq \varphi C_{OPT}^{n-1}$. This contradicts inequality (4.8).

So J_n must be assigned to M_1, which implies $t(S_2^{n-1}) + p_n > \varphi L$, $C_{S-ON}^n = t(S_1^n) > \varphi C_{OPT}^n$ and $t(S_2^n) = t(S_2^{n-1})$. Since $t(S_1^n) + t(S_2^n) = 2L$ and $C_{OPT}^n \geq L$,

$$C_{S-ON}^n - C_{OPT}^n \leq t(S_1^n) - L = L - t(S_2^n) = L - t(S_2^{n-1}). \tag{4.9}$$

Since $t(S_2^{n-1}) + p_n > \varphi L$, together with the above inequality,

$$C_{S-ON}^n - C_{OPT}^n < L - (\varphi L - p_n) = p_n - (\varphi - 1)L. \tag{4.10}$$

If $p_n > (\alpha - 1)L$, considering $p_n \in [a, \alpha a]$, it follows $\alpha a > (\alpha - 1)L$. This implies $L < \frac{\alpha}{\alpha-1}a$. Since $\Sigma = 2L$, we have $\Sigma < \frac{2\alpha}{\alpha-1}a$. This contradicts the assumption $\Sigma \geq \frac{2\alpha}{\alpha-1}$. So

$$p_n \leq (\alpha - 1)L. \tag{4.11}$$

Together with inequality (4.10), it follows that

$$C_{S-ON}^n - C_{OPT}^n < (\alpha - 1)L - (\varphi - 1)L < \frac{\alpha - 1}{2}L = (\varphi - 1)L < (\varphi - 1)C_{OPT}^n.$$

Therefore, $\frac{C_{S-ON}^n}{C_{OPT}^n} < \varphi$, there exists a contradiction.

Case 2. $g_n = 1$.

From the minimality, we have $C_{S-ON}^n = t(S_1^n)$. By Lemma 9, we know that there is at least one job in $\mathcal{J}_2$ scheduled on M_1. Otherwise, we have $\frac{C_{S-ON}^n}{C_{OPT}^n} < \varphi$ and there exists a contradiction. Let $J_k = (p_k, 2)$ denote the last job with $g_i = 2$ in $\mathcal{I}$ scheduled on M_1, i.e., $\mathcal{I} = \{J_1, ..., J_{k-1}, J_k, J_{k+1}..., J_n\}$. In this case, by

taking out J_k and putting it at last position in $\mathcal{I}$, we get a new instance $\mathcal{I}'$, i.e., $\mathcal{I}' = \{J_1, ..., J_{k-1}, J_{k+1}, ..., J_n, J_k\}$. Note that for this new instance $\mathcal{I}'$, the performance of B-SUM-ONLINE algorithm does not become worse, by Step 4 of the algorithm. Now, we renew the indexes of jobs in $\mathcal{I}'$ by their positions, i.e., $\mathcal{I}' = \{J'_1 = J_1, .., J'_{k-1} = J_{k-1}, J'_k = J_{k-1}, ..., J'_{n-1} = J'_n, J'_n = J_k\}$. Since $g'_n = 1$, we have the result of **Case 1**.

Therefore, the theorem follows. ■

Chapter 5

Two uniform parallel machine models

We consider two problems of online scheduling on two uniform machines: online scheduling under a grade of service (GoS) and online scheduling with reassignment. These problems are online in the sense that when a job presents, we have to irrevocably assign it to one of the machines before the next job is seen. The objective is to minimize the makespan.

In the first problem, GoS means that some jobs have to be processed by some machine so that they can be guaranteed a higher quality. Assume that the speed of the higher GoS machine is normalized to 1, while the speed of the other one is s. We show that a lower bound of competitive ratio is $1+\frac{2s}{s+2}$ in the case $0 < s \leq 1$ and $1+\frac{s+1}{s(2s+1)}$ in the case $s > 1$. Then we propose and analyze an online algorithms HSF. HSF is optimal in the case where $s > 1$ and $\Sigma_1 \geq \frac{\Sigma_2}{s}$, where Σ_1 and Σ_2 denote the total processing time of jobs which request higher GoS machine and the total processing time of jobs which request the normal one, respectively.

In the second problem, we study two subproblems $\mathscr{P}_L$ and $\mathscr{P}_A$ proposed in [103]. Assume that the speeds of 2 uniform machines are 1 and $s \geq 1$, respectively. For $\mathscr{P}_L$ where we can reassign the last k jobs of the sequence, we show a lower bound of competitive ratio $1+\frac{1}{1+s}$. For $\mathscr{P}_A$ where we can reassign arbitrary k jobs, we show a lower bound of competitive ratio $\frac{(s+1)^2}{s^2+s+1}$. We propose a $\frac{s+1}{s}$-competitive algorithm HSF-1 for both $\mathscr{P}_L$ and $\mathscr{P}_A$. For $\mathscr{P}_A$, we propose a $\frac{(s+1)^2}{s+2}$-competitive algorithm EX-RA, which is superior to HSF-1 when $1 \leq s \leq \sqrt{2}$.

5.1 Introduction

In the classical uniform machine scheduling problem, there are $m \geq 2$ machines with different speeds. A list of n independent jobs with nonnegative processing times has to be scheduled non-preemptively on these machines with the objective of minimizing the makespan, which is the completion time of the last job in the schedule. This chapter studies the *online* version of this problem with two

uniform machines. We are given a sequence of independent jobs which arrive in a list. We have to assign a job to one of the two uniform machines before the next job shows up.

In the first problem, we deal with online scheduling under a grade of service (GoS). Within an offline context, the concept of GoS was introduced and analyzed in [55]. For the online problem, the well-known List algorithm introduced by Graham [40] has been proved to be the best possible for the case of identical machines. The List algorithm assigns the incoming job to the least loaded machine. Cho and Shani [24] proved that List algorithm is $\frac{1+\sqrt{5}}{2}$-competitive for all s and the bound is tight when $s = \frac{1+\sqrt{5}}{2}$. Epstein et al. [30] provided randomized algorithms with better performance than List algorithm in the case where no preemption is allowed. In the same paper, for the problem with preemption, they proposed an optimal $(1 + \frac{s}{s^2+s+1})$-competitive algorithm for all $s \geq 1$, which cannot be beaten by any randomized algorithm [109]. Angelelli et al. [3] considered the semi-online scheduling on two uniform processors in the case where the total processing time of jobs is known in advance and presented algorithms which are optimal for $s \geq \sqrt{3}, s = 1$ and $\frac{1+\sqrt{17}}{4} \leq s \leq \frac{1+\sqrt{3}}{2}$. Jongho Park et al. [82] studied the online scheduling of two machines under a grade of service (GoS) provision and its semi-online variant where the total processing time is known. They gave an optimal online algorithm whose competitive ratio is $\frac{5}{3}$ and an optimal semi-online algorithm with a competitive ratio $\frac{3}{2}$.

In the second problem, we consider online scheduling with reassignment. In the classical online scheduling problem, jobs cannot be reassigned after they have been assigned to machines. However, it may not be the case in the real world, such as hotel or restaurant reservation and reception. In order to minimize the makespan, we need to reassign some jobs to gain a better effect. Three problems of online scheduling with reassignment have been proposed in [103]. In the first problem $\mathscr{P}_L$, we can reassign the last k jobs of the sequence. Tan et al. [103] proved that List algorithm [40] is optimal with a competitive ratio of $\frac{3}{2}$. In the second problem $\mathscr{P}_E$ where we can reassign the last job of each machine, they proposed an optimal algorithm RE with a competitive ratio of $\sqrt{2}$. In the third problem $\mathscr{P}_A$, we can reassign arbitrary k jobs. They obtained lower bound $\frac{4}{3}$ and presented an optimal algorithm RA. Kellerer et al. [60] studied two models. In one of them, a buffer is available to maintain those arrived but unassigned jobs. In the other, there exists two parallel processors and we need to choose the better one as output. They obtained two optimal algorithms which are $\frac{4}{3}$-competitive. Sanders et al. [90] discussed another kind of reassignment model. When a new job is arriving, we are allowed to migrate some pervious jobs to other machines provided that the total processing time of migrated jobs is not greater than β times of the processing of the new job. They gave an algorithm with a competitive ratio $\frac{3}{2}$ for $\beta = \frac{4}{3}$ on m machines, and an algorithm with a competitive ratio $\frac{7}{6}$ for $\beta = 1$ on two machines. Moreover, they also gave a family of algorithms with competitive ratio $1 + \varepsilon$ for a constant $\beta(\varepsilon)$ on m machines for any fixed ε.

The rest of this chapter is organized as follows. In Section 2, we deal with the problem of online scheduling under a grade of service. In subsection 2.1, we describe the problem and introduce some notations. In subsection 2.2, we show that $1 + \frac{2s}{s+2}$ is a lower bound of competitive ratio in the case where $0 < s \leq 1$ and $1 + \frac{s+1}{s(2s+1)}$ in the case where $s > 1$. In subsection 2.3, we

show some upper bounds of competitive ratio by proposing and analyzing an online algorithms HSF algorithm. In section 3, we deal with the problem of online scheduling with reassignment. In subsection 3.1, we show that a lower bound of competitive ratio for $\mathscr{P}_L$ is $1 + \frac{1}{1+s}$. In subsection 3.2, we show that a lower bound of competitive ratio for $\mathscr{P}_A$ is $\frac{(s+1)^2}{s^2+s+1}$. In subsection 3.3, we give an upper bound of competitive ratio for both $\mathscr{P}_L$ and $\mathscr{P}_A$ by proposing and analyzing HSF-1 algorithm. In subsection 3.4, we prove that EX-RA algorithm is $\frac{(s+1)^2}{s+2}$-competitive.

5.2 Online scheduling under GoS

In this section we consider the problem of online scheduling on two uniform machines under GoS.

5.2.1 Problem definition and notations

We are given two uniform machines. Without loss of generality, we denote the one which can provide higher GoS and whose speed is 1 by M_1, while the other one with lower GoS and speed s by M_s. Note that M_1 can also provide lower GoS, instead of M_s, whenever it is available. A sequence of jobs $\mathcal{I} = \{J_1, J_2, ..., J_n\}$ which arrive online have to be scheduled irrevocably on one of the machines at the time of their arrivals. The new job shows up only after the current job is scheduled. We denote a job by $J_i = (p_i, g_i)$, where the first item p_i is the processing time of job J_i and the second item $g_i \in \{1, 2\}$ denotes the GoS assigned to the job J_i, which is 1 if the job must be processed only by M_1 and 2 if it can be processed by either of two machines. p_i and g_i are not known until the previous job J_{i-1} has been scheduled, except job J_1. The schedule can be seen as a partition of job sequence $\mathcal{I}$ into two subsets, denote by S_1 and S_2, where S_1 and S_2 consist of jobs assigned to M_1 and M_s, respectively. Let $L_1 = \sum_{J_i \in S_1} p_i$ and $L_2 = \frac{1}{s} \sum_{J_i \in S_2} p_i$ denote the workload of M_1 and M_s, respectively. The makespan of the schedule is $max\{L_1, L_2\}$. The online problem can be written as:

Given $\mathcal{I}$, find S_1 and S_2 to minimize $max\{L_1, L_2\}$.

5.2.2 Lower bounds of competitive ratio

In this subsection, we show some lower bounds of competitive ratio in different cases.

Theorem 7 *For the problem of scheduling two uniform machines(with different speeds, s and 1) under GoS, there is no online algorithm with competitive ratio less than: (1)* $1 + \frac{2s}{s+2}$ *in the case where* $0 < s \leq 1$*; (2)* $1 + \frac{s+1}{s(2s+1)}$ *in the case where* $s > 1$.

Proof. We discuss two cases in the theorem according to the value of s.

(1) $0 < s \leq 1$.

For simplicity of expression, let $\varphi = 1 + \frac{2s}{s+2}$ and $\theta = \frac{1}{s}$. It follows $\theta \geq 1$. The problem is then converted to that the speeds of M_1 and M_s are θ and 1, respectively. The desired lower bound is $\varphi = 1 + \frac{2}{2\theta+1}$. We give a job input

sequence to show that there is no online algorithm whose competitive ratio is less than φ. The job sequence consists of at most 5 jobs which arrive in the order $\{J_1, J_2, J_3, J_4, J_5\}$. Let $J_1 = (\theta, 2)$.

Case 1. J_1 is scheduled on M_1.

Let $J_2 = (\theta^2, 1)$. Therefore, $C_{ON} \geq \frac{1}{\theta}(\theta + \theta^2) = 1 + \theta$. An optimal solution consists of assigning J_1 and J_2 to M_s and M_1, respectively. Thus, $C_{OPT} = \theta$ and $\frac{C_{ON}}{C_{OPT}} \geq \frac{1+\theta}{\theta} > \varphi$.

Case 2. J_1 is scheduled on M_s.

Then we further generate $J_2 = (\theta^2, 2)$.

Case 2.1. J_2 is assigned to M_s.

It follows $C_{ON} \geq \theta + \theta^2$. In an optimal solution, J_1 and J_2 are scheduled on M_s and M_1, respectively. Thus, $C_{OPT} = \theta$ and $\frac{C_{ON}}{C_{OPT}} \geq \frac{\theta+\theta^2}{\theta} = 1 + \theta > \varphi$.

Case 2.2. J_2 is assigned to M_1.

We generate $J_3 = (\theta^2, 2)$.

Case 2.2.1. J_3 is scheduled on M_1.

Let $J_4 = (2\theta^3 + \theta^2, 1)$. Therefore, $C_{ON} \geq \frac{1}{\theta}(\theta^2 + \theta^2 + 2\theta^3 + \theta^2) = 3\theta + 2\theta^2$. An optimal schedule consists of assigning J_1, J_2, J_3 to M_s and J_4 to M_1. Thus, $C_{OPT} = \frac{1}{\theta}(2\theta^3 + \theta^2) = 2\theta^2 + \theta$. It follows

$$\frac{C_{ON}}{C_{OPT}} \geq \frac{3\theta + 2\theta^2}{2\theta^2 + \theta} = 1 + \frac{2}{2\theta + 1} = \varphi.$$

Case 2.2.2. J_3 is scheduled on M_s.

We further generate $J_4 = (2\theta^3 + \theta^2, 2)$.

Case 2.2.2.1. J_4 is assigned to M_s.

It follows $C_{ON} \geq \theta + \theta^2 + 2\theta^3 + \theta^2 = \theta + 2\theta^2 + 2\theta^3$. An optimal solution can assign J_1, J_2, J_3 to M_s and J_4 to M_1. Therefore, it follows that $C_{OPT} = \frac{1}{\theta}(2\theta^3 + \theta^2) = 2\theta^2 + \theta$. Then

$$\frac{C_{ON}}{C_{OPT}} \geq \frac{\theta + 2\theta^2 + 2\theta^3}{2\theta^2 + \theta} = 1 + \frac{2\theta^2}{2\theta + 1} \geq \varphi.$$

Case 2.2.2.2. J_4 is assigned to M_1.

Let $J_5 = (2\theta^4 + 3\theta^3 + \theta^2, 1)$. It follows $C_{ON} \geq \frac{1}{\theta}(\theta^2 + 2\theta^3 + \theta^2 + 2\theta^4 + 3\theta^3 + \theta^2) = 2\theta^3 + 5\theta^2 + 3\theta$. An optimal schedule can assign J_1, J_2, J_3, J_4 to M_s and J_5 to M_1. Thus, $C_{OPT} = \frac{1}{\theta}(2\theta^4 + 3\theta^3 + \theta^2) = 2\theta^3 + 3\theta^2 + \theta$. It follows

$$\frac{C_{ON}}{C_{OPT}} \geq \frac{2\theta^3 + 5\theta^2 + 3\theta}{2\theta^3 + 3\theta^2 + \theta} = 1 + \frac{2}{2\theta + 1} = \varphi.$$

(2) $s > 1$.

In this case, we also give a job sequence which consists of at most 5 jobs to show that the competitive ratio of any online algorithm cannot be less than $1 + \frac{s+1}{s(2s+1)}$. Let $\varphi = 1 + \frac{s+1}{s(2s+1)}$. We begin with $J_1 = (s, 2)$.

Case 1. J_1 is assigned to M_1.

Let $J_2 = (1, 1)$. It follows $C_{ON} \geq s + 1$. An optimal schedule can assign J_1 to M_s and J_2 to M_1. Thus, $C_{OPT} = 1$ and $\frac{C_{ON}}{C_{OPT}} \geq s + 1 \geq \varphi$.

Case 2. J_1 is assigned to M_s.

We further generate $J_2 = (1, 2)$.

Case 2.1. J_2 is scheduled on M_s.

It follows $C_{ON} \geq \frac{1}{s}(s + 1) = 1 + \frac{1}{s}$. An optimal solution can assign J_1 to M_s and J_2 to M_1. Therefore, $C_{OPT} = 1$ and $\frac{C_{ON}}{C_{OPT}} \geq 1 + \frac{1}{s} > \varphi$.

Case 2.2. J_2 is scheduled on M_1.
We further generate $J_3 = (s, 2)$.

Case 2.2.1. J_3 is assigned to M_1.
Let $J_4 = (2+\frac{1}{s}, 1)$. Thus, $C_{ON} \geq 1+s+2+\frac{1}{s} = 3+s+\frac{1}{s}$. An optimal schedule can assign J_1, J_2, J_3 to M_s and J_4 to M_1. It follows $C_{OPT} = 2+\frac{1}{s}$. Since $s > 1$,

$$\frac{C_{ON}}{C_{OPT}} \geq \frac{3+s+\frac{1}{s}}{2+\frac{1}{s}} = 1 + \frac{s(s+1)}{2s+1} > \varphi.$$

Case 2.2.2. J_3 is assigned to M_s.
Then we generate $J_4 = (2+\frac{1}{s}, 2)$.

Case 2.2.2.1. J_4 is scheduled on M_s.
It follows $C_{ON} \geq \frac{1}{s}(s+s+2+\frac{1}{s}) = 2+\frac{2}{s}+\frac{1}{s^2}$. An optimal schedule can assign J_1, J_2, J_3 to M_s and J_4 to M_1. Thus, $C_{OPT} = 2+\frac{1}{s}$, and

$$\frac{C_{ON}}{C_{OPT}} \geq \frac{2+\frac{2}{s}+\frac{1}{s^2}}{2+\frac{1}{s}} = 1 + \frac{s+1}{s(2s+1)} = \varphi.$$

Case 2.2.2.2. J_4 is scheduled on M_1.
We further generate $J_5 = (2+\frac{3}{s}+\frac{1}{s^2}, 1)$. Therefore $C_{ON} \geq 1+2+\frac{1}{s}+2+\frac{3}{s}+\frac{1}{s^2} = 5+\frac{4}{s}+\frac{1}{s^2}$. An optimal solution can assign J_1, J_2, J_3, J_4 to M_s and J_5 to M_1. Thus, $C_{OPT} = 2+\frac{3}{s}+\frac{1}{s^2}$. Since $s > 1$,

$$\frac{C_{ON}}{C_{OPT}} \geq \frac{5+\frac{4}{s}+\frac{1}{s^2}}{2+\frac{3}{s}+\frac{1}{s^2}} = 1 + \frac{s(3s+1)}{(2s+1)(s+1)} > 1 + \frac{s+1}{s(2s+1)} = \varphi.$$

The theorem follows. ■

Remark 5 *In the case where $s = 1$, a lower bound of competitive ratio for all online algorithms is $\frac{5}{3}$.*

This is the lower bound proved in [82]. When $s = 1$, our online problem reduces to the one in [82].

5.2.3 Upper bounds of competitive ratio

In this subsection we show an upper bound of competitive ratio by giving an algorithms HSF. Let Σ_1 and Σ_2 denote the total processing time of jobs with $g_i = 1$ and with $g_i = 2$, respectively. We first prove that in the case where $s > 1$ and $\Sigma_1 \geq \frac{\Sigma_2}{s}$, HSF is optimal.

HSF algorithm

There is a straightforward algorithm called HSF algorithm (higher speed machine first). HSF algorithm schedules jobs as many as possible on higher speed machine. I.e., if $0 < s \leq 1$, HSF algorithm schedules all the jobs on M_1; else, HSF algorithm schedules all jobs with $g_i = 2$ on M_s, and the others on M_1.

Lemma 10 *In the case $0 < s \leq 1$, HSF is $(s+1)$-competitive.*

Proof. By the definition of HSF algorithm, $C_{ON} = \Sigma_1 + \Sigma_2$. On the other hand, the offline adversary can generate a job sequence which can be perfectly distributed to both machines in order to maintain a load balance between two machines; i.e., $\Sigma_1 = \frac{\Sigma_2}{s}$. Therefore,

$$\frac{C_{ON}}{C_{OPT}} \leq \frac{(\Sigma_1 + \Sigma_2)}{\frac{\Sigma_1+\Sigma_2}{s+1}} = s + 1.$$

■

Lemma 11 *In the case where $s > 1$, if $\Sigma_1 \geq \frac{\Sigma_2}{s}$, HSF is optimal.*

Proof. Since $C_{OPT} = max\{\Sigma_1, \frac{\Sigma_2}{s}\} = \Sigma_1$ and $C_{ON} = max\{\Sigma_1, \frac{\Sigma_2}{s}\} = \Sigma_1$, the lemma follows. ■

Lemma 12 *In the case where $s > 1$, if $\Sigma_1 < \frac{\Sigma_2}{s}$, HSF is $\frac{s+1}{s}$-competitive.*

Proof. Since $\Sigma_1 < \frac{\Sigma_2}{s}$, $C_{ON} = max\{\Sigma_1, \frac{\Sigma_2}{s}\} = \frac{\Sigma_2}{s}$. On the other hand, the offline adversary can generate a job sequence which can be perfectly distributed to both machines. Let $\frac{\Sigma_1}{\Sigma_2} = \epsilon$ $(0 < \epsilon < \frac{1}{s})$. Then

$$\frac{C_{ON}}{C_{OPT}} \leq \frac{\Sigma_2}{s} \cdot \frac{s+1}{\Sigma_1 + \Sigma_2} = \frac{s+1}{s(1+\epsilon)}.$$

Let $\epsilon \to 0$, we have the desired result. ■

5.3 Online scheduling with reassignment

In this section we consider the problem of online scheduling on two uniform machines with reassignment. Without loss of generality, we denote the machine with speed 1 by M_1 and the other one with speed s by M_s, where $s \geq 1$. We use C_{ON} and C_{OPT} to denote the makespan of an online algorithm and that of offline optimal algorithm (after reassignment), respectively.

5.3.1 Lower bound of competitive ratio for $\mathscr{P}_L$ (last k jobs can be reassigned)

In this subsection, we show a lower bound $1 + \frac{1}{1+s}$ of competitive ratio.

Theorem 8 *For $\mathscr{P}_L$, there is no online algorithm with a competitive ratio less than $1 + \frac{1}{1+s}$.*

Proof. Let ε be a sufficiently small positive number. For simplicity of expression, let $\varphi = 1 + \frac{1}{1+s}$. We generate job J_1 with $p_1 = s$, and discuss two cases according to the assignment of J_1 by the online algorithm in below.

Case 1: J_1 is assigned to M_1.

We generate J_2 with $p_2 = s^2$. If J_2 is assigned to M_1, we further generate last k jobs with processing time ε. It follows $C_{ON} \geq s + s^2$. The offline algorithm may assign jobs to the two machines with equivalent workload, i.e., $C_{OPT} = \frac{s+s^2+k\varepsilon}{1+s} = s + \frac{k\varepsilon}{1+s} < s + k\varepsilon$. So,

$$\frac{C_{ON}}{C_{OPT}} > \frac{s+s^2}{s+k\varepsilon} \to 1 + s \geq \varphi. \qquad (\varepsilon \to 0)$$

Otherwise if J_2 is assigned to M_s, we generate job J_3 with $p_3 = s^2(s+1)$ and further generate last k jobs with processing time ε. J_3 is assigned to either M_1 or M_s. In either case, $C_{ON} \geq \frac{1}{s}(s^2+s^2(s+1)) = s^2+2s$. The offline algorithm can make the two machines have equal workload, i.e., $C_{OPT} = \frac{s+s^2+s^2(s+1)+k\varepsilon}{1+s} < s + s^2 + k\varepsilon$. Hence,

$$\frac{C_{ON}}{C_{OPT}} > \frac{s^2+2s}{s+s^2+k\varepsilon} \to 1 + \frac{1}{1+s} = \varphi. \qquad (\varepsilon \to 0)$$

Case 2: J_1 is assigned to M_s.
We generate job J_2 with $p_2 = 1$. If J_2 is assigned to M_s, we further generate last k jobs with processing time ε. It follows $C_{ON} \geq 1 + \frac{1}{s}$. On the other hand, $C_{OPT} = \frac{s+1+k\varepsilon}{1+s} < 1 + k\varepsilon$.

$$\frac{C_{ON}}{C_{OPT}} > \frac{1+\frac{1}{s}}{1+k\varepsilon} \to 1 + \frac{1}{s} > \varphi. \qquad (\varepsilon \to 0)$$

If job J_2 is assigned to M_1, we generate job J_3 with $p_3 = s + s^2$ and then further generate last k jobs with processing time ε. Therefore, $C_{ON} \geq \frac{1}{s}(s + s + s^2) = 2 + s$. On the other hand, $C_{OPT} = \frac{s+1+(s+s^2)+k\varepsilon}{1+s} < 1 + s + k\varepsilon$.

$$\frac{C_{ON}}{C_{OPT}} > \frac{2+s}{1+s+k\varepsilon} \to 1 + \frac{1}{1+s} = \varphi. \qquad (\varepsilon \to 0)$$

The theorem follows. ■

Remark 6 *For $\mathscr{P}_L$, if the speeds of two machines are identical, a lower bound of competitive ratio is $\frac{3}{2}$.*

The result in [103] is a special case of Theorem 8. Note that when $s = 1$, List algorithm is optimal [24].

5.3.2 Lower bound of competitive ratio for $\mathscr{P}_A$ (we can reassign arbitrary k jobs)

In this subsection, we show a lower bound $\frac{(s+1)^2}{s^2+s+1}$ of competitive ratio for $\mathscr{P}_A$. For an arbitrary online algorithm A_{ON}, let $L_1 = \sum_{J_i \in S_1} p_i$ and $L_2 = \frac{1}{s}\sum_{J_i \in S_2} p_i$ denote the workloads of M_1 and M_s before reassignment, respectively.

Theorem 9 *For $\mathscr{P}_A$, there is no online algorithm with competitive ratio less than $\frac{(s+1)^2}{s^2+s+1}$.*

Proof. We begin with Mn jobs with processing time 1, where M and n are two positive integers.

Since the total processing time of the Mn jobs is Mn, $L_1 + sL_2 = Mn$. For simplicity, let $x = \frac{L_1}{Mn}$ and $y = \frac{L_2}{Mn}$. Thus,

$$x + sy = 1. \qquad (5.1)$$

Let $\varphi = \frac{(s+1)^2}{s^2+s+1}$. Consider the following two cases.

Case 1: $0 \leq L_2 \leq L_1$.
It follows

$$0 \leq y \leq x \leq 1. \tag{5.2}$$

By equation (5.1) and inequality (5.2), x is bounded such that $\frac{1}{s+1} \leq x \leq 1$. Since $s \geq 1$, $\frac{1}{s+1} < \frac{1+s}{s^2+s+1} < 1$. In below we further discuss two subcases according to the bound of x.

Case 1.1: $\frac{1+s}{s^2+s+1} < x \leq 1$.
No more jobs arrive. $C_{ON} > L_1 - k$ since A_{ON} can at most move k jobs from M_1 to M_s during reassignment, while $C_{OPT} = \frac{Mn}{1+s}$. So,

$$\frac{C_{ON}}{C_{OPT}} \geq \frac{L_1 - k}{\frac{Mn}{1+s}} \to (1+s)x > \frac{(s+1)^2}{s^2+s+1} = \varphi. \quad (n \to \infty)$$

Case 1.2: $\frac{1}{1+s} \leq x \leq \frac{1+s}{s^2+s+1}$.
We further generate the last job with processing time sMn. With similar reasoning as that in Case 1.1,

$$\frac{C_{ON}}{C_{OPT}} \geq \frac{L_2 + Mn - \frac{k}{s}}{Mn} \to y + 1 = \frac{1-x}{s} + 1 \geq \frac{(1+s)^2}{s^2+s+1} = \varphi. \quad (n \to \infty)$$

Case 2: $0 \leq L_1 \leq L_2$.
Together with equation (7),

$$0 \leq x \leq y \leq \frac{1}{s}. \tag{5.3}$$

By equation (5.1) and inequality (5.3), x is bounded such that $0 \leq x \leq \frac{1}{s+1}$. We divide the case into two subcases according to the bound of s in below.

Case 2.1: $s > \frac{\sqrt{5}+1}{2}$.
We further generate the last job with processing time sMn. Since $s > \frac{\sqrt{5}+1}{2}$, $1 + s - s^2 < 0$. Together with $0 \leq x \leq \frac{1}{s+1}$, we have $x + s > y + 1$. As $n \to \infty$,

$$\begin{aligned} \frac{C_{ON}}{C_{OPT}} &\geq \frac{\min\{L_1 + sMn, L_2 + Mn\} - k}{Mn} \to \min\{x + s, y + 1\} \\ &= y + 1 = \frac{1-x}{s} + 1 \geq \frac{s+2}{s+1} > \varphi. \end{aligned}$$

Case 2.2: $1 \leq s \leq \frac{\sqrt{5}+1}{2}$.
It follows $0 \leq \frac{1+s-s^2}{s+1} \leq \frac{1}{s+1}$.

Case 2.2.1: $0 \leq x \leq \frac{1+s-s^2}{2s+1}$.
No more jobs arrive. $C_{ON} > L_2 - \frac{k}{s}$ with similar reasoning as that in Case 1.1. So,

$$\frac{C_{ON}}{C_{OPT}} \geq \frac{L_2 - \frac{k}{s}}{\frac{Mn}{1+s}} \to (1+s)y = \frac{(1+s)(1-x)}{s} \geq \frac{(1+s)^2}{2s+1} \geq \varphi. \quad (n \to \infty)$$

Case 2.2.2: $\frac{1+s-s^2}{2s+1} < x \leq \frac{1+s-s^2}{s+1}$.
We further generate the last job with processing time sMn. $x + s \leq y + 1$ due

to $x \leq \frac{1+s-s^2}{s+1}$. It follows

$$\begin{aligned}\frac{C_{ON}}{C_{OPT}} &\geq \frac{\min\{L_1+sMn, L_2+Mn\}-\frac{k}{s}}{Mn} \rightarrow \min\{x+s, y+1\} \\ &= x+s > \frac{(1+s)^2}{2s+1} \geq \varphi. \quad (n \rightarrow \infty)\end{aligned}$$

Case 2.2.3: $\frac{1+s-s^2}{s+1} < x \leq \frac{1}{s+1}$.
We further generate the last job with processing time sMn. Since $\frac{1+s-s^2}{s+1} < x$, $x+s > y+1$. Therefore,

$$\begin{aligned}\frac{C_{ON}}{C_{OPT}} &\geq \frac{\min\{L_1+sMn, L_2+Mn\}-\frac{k}{s}}{Mn} \rightarrow \min\{x+s, y+1\} \\ &= y+1 \geq \frac{s+2}{s+1} > \varphi. \quad (n \rightarrow \infty)\end{aligned}$$

The theorem follows. ■

Remark 7 *For $\mathscr{P}_A$, there is no online algorithm with a competitive ratio less than $\frac{4}{3}$.*

The result analyzed in [103] is a special case of Theorem 9.

5.3.3 Upper bound of competitive ratio for $\mathscr{P}_L$ and $\mathscr{P}_A$

In this subsection, we propose a straightforward algorithm named HSF-1(higher speed machine first). HSF-1 schedules all the jobs on M_s ($s \geq 1$).

Given a job sequence $\mathcal{I}$, let $\Sigma = \sum_{J_i \in \mathcal{I}} p_i$ be the total processing time of jobs in $\mathcal{I}$.

Lemma 13 *HSF-1 is $\frac{s+1}{s}$-competitive for $\mathscr{P}_L$ and $\mathscr{P}_A$.*

Proof. By the definition of HSF algorithm, $C_{ON} = \frac{\Sigma}{s}$. While, the offline adversary can generate a job sequence which can be perfectly distributed to both machines to maintain a load balance between two machines. Therefore,

$$\frac{C_{ON}}{C_{OPT}} \leq \frac{\frac{\Sigma}{s}}{\frac{\Sigma}{s+1}} = \frac{s+1}{s}.$$

■

5.3.4 EX-RA algorithm for $\mathscr{P}_A$

In this subsection, we propose another online algorithm called EX-RA, which is superior to HSF in competitiveness when $1 \leq s \leq \sqrt{2}$. We define $\mathcal{L}_1$ and $\mathcal{L}_2$ to be the workloads of M_1 and M_s (after reassignment), respectively. In the following we first give a lemma.

Lemma 14 *Given that $r \geq 1$, for any online algorithm $\mathcal{A}_{ON}$, if $\mathcal{L}_2 \leq \mathcal{L}_1 \leq r\mathcal{L}_2$ or $\mathcal{L}_1 \leq \mathcal{L}_2 \leq r\mathcal{L}_1$, then the competitive ratio of $\mathcal{A}_{ON}$ is at most $\frac{r(s+1)}{r+s}$.*

Proof. We discuss the two cases in the lemma.

Case 1. $\mathcal{L}_2 \leq \mathcal{L}_1 \leq r\mathcal{L}_2$.

$$\frac{C_{ON}}{C_{OPT}} \leq \frac{\mathcal{L}_1}{\frac{\mathcal{L}_1+s\mathcal{L}_2}{s+1}} \leq \frac{(s+1)\mathcal{L}_1}{\mathcal{L}_1+\frac{s}{r}\mathcal{L}_1} = \frac{r(s+1)}{r+s}.$$

Case 2. $\mathcal{L}_1 \leq \mathcal{L}_2 \leq r\mathcal{L}_1$.

$$\frac{C_{ON}}{C_{OPT}} \leq \frac{\mathcal{L}_2}{\frac{\mathcal{L}_1+s\mathcal{L}_2}{s+1}} \leq \frac{(s+1)\mathcal{L}_2}{\frac{1}{r}\mathcal{L}_2+s\mathcal{L}_2} = \frac{r(s+1)}{r+s}.$$

The lemma follows. ■

EX-RA algorithm is an extension from the RA algorithm proposed in [103]. Before describing EX-RA, we introduce some notations. Let L_1^2 and L_2^2 denote the workloads of M_1 and M_s just after the first two jobs J_1 and J_2 (before the third job) have been scheduled, respectively. If there is only one job in the job sequence, we also use L_1^2 and L_2^2 to denote the workloads (of M_1 and M_s after scheduling the only job). Let $M_1 = \max\{L_1^2, L_2^2\}$ denote higher workload machine just after scheduling the first two jobs (before scheduling the third job) and $M_2 = \min\{L_1^2, L_2^2\}$ denote the other one. Note that M_1 and M_2 are determined just after scheduling two jobs in the job sequence and in the following loops the machine which is denoted by M_1 (or M_2) does not change. For example, just after the first two jobs (before the third job) have been scheduled, if M_s is the higher workload machine, M_1 denotes M_s. When the following jobs arrive, M_1 always denotes M_s. We define M_1^i and M_2^i to be M_1 and M_2 just before job J_i ($i \geq 3$) has been scheduled, respectively. We use $s(M_1)$ to denote the speed of M_1. I.e., if M_1 denotes M_s, then $s(M_1) = s$. Let M_1^θ and M_2^θ denote the workloads of M_1 and M_2 after all jobs have been scheduled (just before reassignment), respectively. Let M_1^π and M_2^π denote the workloads of M_1 and M_2 after reassignment, respectively. It follows $C_{ON} = \max\{M_1^\pi, M_2^\pi\}$.

EX-RA algorithm consists of two parts: assignment and reassignment. In assignment, the algorithm firstly assigns two jobs to different machines. Then it denote the higher workload machine with M_1 and the lower one with M_2. It maintains the workload of M_1 not greater than $(s+1)$ times of the workload of M_2. In reassignment, the EX-RA algorithm tries to maintain the workload of M_2 not greater than $(s+1)$ times of that of M_1.

EX-RA algorithm works as follows:

Step 1: Assign the first two jobs to two different machines. Let $M_1 = \max\{L_1^2, L_2^2\}$ and $M_2 = \min\{L_1^2, L_2^2\}$. If no job arrives in future, go to **Step 5**.

Step 2: Receive job J_i.

Step 3: If $M_1^i + \frac{p_i}{s(M_1^i)} \leq (s+1)M_2^i$, assign J_i to M_1^i;
otherwise, assign J_i to M_2^i.

Step 4: If no job arrives in future, go to **Step 5**;
otherwise, $i := i+1$ and go to **Step 2**.

Step 5: If $M_2^\theta \leq (s+1)M_1^\theta$, let $M_1^\pi := M_1^\theta$ and $M_2^\pi := M_2^\theta$;
otherwise, reassign the last second job of M_2^θ to M_1^θ and update M_1^θ and M_2^θ to be M_1^π and M_2^π.

Step 6: Output M_1^π and M_2^π.

Theorem 10 *For problem $\mathscr{P}_A$ of two uniform machines, EX-RA algorithm is $\frac{(s+1)^2}{s+2}$-competitive, where $1 \leq s \leq \frac{\sqrt{5}+1}{2}$.*

Proof. Let $\varphi = \frac{(s+1)^2}{s+2}$. If there is only one job, it is trivial that $\frac{C_{ON}}{C_{OPT}} \leq s < \varphi$. If there are only two jobs J_1 and J_2, without loss of generality, let $p_1 \leq p_2$. Thus, $C_{ON} \leq p_2$ and $C_{OPT} \geq \max\{p_1, \frac{p_2}{s}\}$. If $p_2 \leq sp_1$,

$$\frac{C_{ON}}{C_{OPT}} \leq \frac{p_2}{\max\{p_1, \frac{p_2}{s}\}} = \frac{p_2}{p_1} < s < \varphi.$$

Otherwise if $p_2 > sp_1$,

$$\frac{C_{ON}}{C_{OPT}} \leq \frac{p_2}{\max\{p_1, \frac{p_2}{s}\}} = \frac{p_2}{\frac{p_2}{s}} = s < \varphi.$$

In the rest proof, we assume that there are more than two jobs in the job input sequence.

Case 1: $M_1^\theta \geq M_2^\theta$.
It follows $M_2^\theta \leq (s+1)M_1^\theta$. By **Step 5**,

$$M_1^\pi = M_1^\theta, \quad M_2^\pi = M_2^\theta. \tag{5.4}$$

If no job is assigned to M_1 in **Step 3**, let J_k with processing time p_k denote the job assigned to M_1. Therefore, $C_{ON} = M_1^\pi = p_k$. Since $C_{OPT} \geq \frac{p_k}{s}$,

$$\frac{C_{ON}}{C_{OPT}} \leq \frac{p_k}{\frac{p_k}{s}} = s < \varphi.$$

Otherwise if some jobs are assigned to M_1 in **Step 3**, $M_1^\theta \leq (s+1)M_2^\theta$. By equation (5.4), it follows $M_2^\pi \leq M_1^\pi \leq (s+1)M_2^\pi$. By Lemma 14, EX-RA algorithm is φ-competitive.

Case 2: $M_1^\theta < M_2^\theta$.
If $M_2^\theta \leq (s+1)M_1^\theta$, EX-RA algorithm is φ-competitive due to **Step 5** and Lemma 14. Therefore, in the following we focus on the case where $M_2^\theta > (s+1)M_1^\theta$, which implies that at least two jobs are assigned to M_2. Let J_y and J_x be the last two jobs on M_2 before reassignment such that J_y arrives before J_x. Let B be the total processing time of jobs arriving between J_y and J_x (except J_y and J_x). Let A be the total processing time of jobs coming after J_x. These jobs are assigned to M_1.

Case 2.1: M_1 denotes M_1.
Considering jobs J_x and J_y, we have

$$M_1^y + p_y > (s+1)M_2^y, \tag{5.5}$$

$$M_1^y + B + p_x > (s+1)\left(M_2^y + \frac{p_y}{s}\right). \tag{5.6}$$

By $M_2^\theta > (s+1)M_1^\theta$ and **Step 5**, the reassignment consists of moving J_y from M_s to M_1. Before reassignment,

$$M_1^\theta = M_1^y + B + A, \qquad M_2^\theta = M_2^y + \frac{p_y}{s} + \frac{p_x}{s}. \tag{5.7}$$

Since $M_2^\theta > (s+1)M_1^\theta$,

$$M_2^y + \frac{p_y}{s} + \frac{p_x}{s} > (s+1)(M_1^y + B + A). \tag{5.8}$$

After reassignment,

$$M_1^\pi = M_1^y + B + A + p_y, \qquad M_2^\pi = M_2^y + \frac{p_x}{s}. \tag{5.9}$$

Case 2.1.1: $M_2^\pi > (s+1)M_1^\pi$.
It follows $C_{OPT} = M_2^\pi$. By inequality (5.5) and equation (5.9),

$$\begin{aligned} p_x &> s[(s+1)(M_1^y + B + A + p_y) - M_2^y] \geq s[(s+1)(M_1^y + p_y) - M_2^y] \\ &\geq s^2(s+2)M_2^y. \end{aligned}$$

Since $C_{OPT} \geq p_x/s$,

$$\frac{C_{ON}}{C_{OPT}} \leq \frac{M_2^\pi}{\frac{p_x}{s}} = \frac{M_2^y + \frac{p_x}{s}}{\frac{p_x}{s}} < \frac{1}{s(s+2)} + 1 \leq \varphi.$$

Case 2.1.2: $M_1^\pi \leq M_2^\pi \leq (s+1)M_1^\pi$.
It follows $C_{ON}/C_{OPT} \leq \varphi$ by Lemma 14.
Case 2.1.3: $M_2^\pi < M_1^\pi$.
It follows $C_{ON} = M_1^\pi$.

Let both sides of inequality (5.6) multiply a factor $\frac{s^2+s+1}{2s+1}$ and add to both sides of inequality (5.8), considering equation (5.9), it follows that

$$\begin{aligned} M_2^\pi &< M_1^\pi = M_1^y + B + A + p_y < \frac{s^3+s^2+3s+1}{s^2(s+2)} p_x \\ &\leq \frac{s^3+s^2+3s+1}{s(s+2)} M_2^\pi \leq (s+1)M_2^\pi. \end{aligned}$$

It follows $C_{ON}/C_{OPT} \leq \varphi$ by Lemma 14.
Case 2.2: M_1 denotes M_s.
Considering jobs J_x and J_y, we have

$$M_1^y + \frac{p_y}{s} > (s+1)M_2^y, \tag{5.10}$$

$$M_1^y + \frac{B}{s} + \frac{p_x}{s} > (s+1)\left(M_2^y + P_y\right). \tag{5.11}$$

Combining $M_2^\theta > (s+1)M_1^\theta$ with **Step 5**, the reassignment consists of shifting J_y from M_1 to M_s. Before reassignment,

$$M_1^\theta = M_1^y + \frac{B}{s} + \frac{A}{s}, \qquad M_2^\theta = M_2^y + p_y + p_x. \tag{5.12}$$

Since $M_2^\theta > (s+1)M_1^\theta$,

$$M_2^y + p_y + p_x > (s+1)\left(M_1^y + \frac{B}{s} + \frac{A}{s}\right). \tag{5.13}$$

After reassignment,

$$M_1^\pi = M_1^y + \frac{B}{s} + \frac{A}{s} + \frac{p_y}{s}, \qquad M_2^\pi = M_2^y + p_x. \tag{5.14}$$

Case 2.2.1: $M_2^\pi > (s+1)M_1^\pi$.
It follows $C_{ON} = M_2^\pi$. Considering inequality (5.10) and equation (5.14), it follows that

$$P_x > [(s+1)(M_1^y + \frac{B}{s} + \frac{A}{s} + \frac{p_y}{s}) - M_2^y] \geq [(s+1)(M_1^y + \frac{p_y}{s}) - M_2^y] \geq s(s+2)M_2^y.$$

Since $C_{OPT} \geq \frac{p_x}{s}$,

$$\frac{C_{ON}}{C_{OPT}} \leq \frac{M_2^\pi}{\frac{P_x}{s}} = \frac{M_2^y + P_x}{\frac{P_x}{s}} < s\left(\frac{1}{s(s+2)} + 1\right) = \frac{(s+1)^2}{s+2} = \varphi.$$

Case 2.2.2: $M_1^\pi \leq M_2^\pi \leq (s+1)M_1^\pi$.
It follows $C_{ON}/C_{OPT} \leq \varphi$ by Lemma 14.
Case 2.2.3: $M_2^\pi < M_1^\pi$.
Let both sides of inequality (5.13) multiply a factor $\frac{s^2+s+1}{2s+1}$ and add to both sides of inequality (5.11), considering equation (5.14),

$$\begin{aligned} M_2^\pi &< M_1^\pi = M_1^y + \frac{B}{s} + \frac{A}{s} + \frac{p_y}{s} < \frac{s^3+s^2+3s+1}{s^3(s+2)} p_x \\ &\leq \frac{s^3+s^2+3s+1}{s^3(s+2)} M_2^\pi \leq (s+1)M_2^\pi. \end{aligned}$$

It follows $C_{ON}/C_{OPT} \leq \varphi$ by Lemma 14.
According to the above discussion, the theorem follows. ■

Remark 8 *EX-RA algorithm is superior to HSF algorithm in the case where* $1 \leq s \leq \sqrt{2}$.

Remark 9 *EX-RA algorithm is optimal in the case where* $s = 1$.

As a consequence, Theorem 10 generalizes the result in [103].

Chapter 6

M identical parallel machine model

We study a maximization problem: online scheduling on m identical machines to maximize the number of early jobs. The problem is online in the sense that all jobs arrive over time. Each job's characteristics, such as processing time and due date, become known at its arrival time. We consider the *preemption-restart model*, which means that preemption is allowed and once a job is restarted it loses all the progress that has been made on this job so far. If in some schedule a job is completed before or at its due date, then it is called *early* (or *on time*). The objective is to maximize the number of early jobs. For m identical machines, we show that an upper bound of competitive ratio is $1-\frac{1}{2m}$ and prove that ECT (earliest completion time) algorithm is $\frac{1}{2}$-competitive.

6.1 Introduction

In this chapter, we consider the third model where jobs arrive over time. There are also three models for online scheduling where jobs arrive over time. The first one, called *non-preemptive model*, assumes that once a job is started on the machine, it must run to completion. The second one, named *preemption-resume model*, assume the currently processed job may be preempted at any moment in time, and it may be resumed at any later moment in time. The third one is *preemption-restart model*. It assumes that the currently processed job may be preempted at any moment in time. However, by preempting a job, all the progress that has been made on this job so far is lost. In this model, the finally constructed schedule is non-preemptive. In this paper, we focus on the preemption-restart model in the online setting where jobs arrive over time. Note that we investigate a maximization problem in this chapter. Correspondingly, the competitive ratio form changes.

Sgall et al. [93] gave a survey on online scheduling. There are many results in the non-preemptive and in the preemption-resume model. Shmoys et al. [95] introduced the preemption-restart model and presented several results for scheduling m parallel machines to minimize the makespan. They studied the minimization problems. Maximization problems are rarely researched in the online settings. Hoogeveen et al. [50] studied the online scheduling on a single

machine to maximize the number of early jobs in the preemption-restart model. They proved that the shortest remaining processing time(SRPT) rule yields an optimal online algorithm with competitive ratio $\frac{1}{2}$. Note that SRPT rule implies earliest completion time (ECT) rule. Our results in the paper are generalizations of his results.

The rest of this chapter is organized as follows. In section 2, we introduce some definitions and notations. In section 3, we show that $1 - \frac{1}{2m}$ is an upper bound of competitive ratio for all online algorithms. In section 4, we present an online algorithm *ECT* based on the earliest completion time (ECT) rule, and prove that it is $\frac{1}{2}$-competitive.

6.2 Problem definition and notations

We are given m identical machines. Without loss of generality, we denote them by M_1, ..., M_m, respectively. A sequence of jobs $\mathcal{I} = \{J_1, ..., J_n\}$ arrive over time. Each machine can process at most one job at a time. The model considered is a preemption-restart model. Preemption is allowed, and by preempting a job, all the progress that has been made on this job so far is lost. Each job has a due date. If job is completed before or at its due date, then it is called *early* (or *on time*). The objective is to maximize the number of early job.

We introduce some definitions and notations as follows.

early job: a job is completed before or at its due date in some schedule. (The opposite is tardy job.)
p_j*:* the processing time of job J_j.
r_j*:* the release time of job J_j.
d_j*:* the due date of job J_j.
feasible schedule: a schedule of early jobs.
current workload (of a machine): the total processing time of all jobs currently assigned to that machine.

Since the tardy jobs in a schedule cannot influence the objective value for this problem, we only consider the feasible schedule (omitting the tardy jobs). Thus, we use a feasible schedule to replace a schedule in the remainder.

At any time t, let $\overline{p_j}$ denote the remaining processing time of job J_j with $r_j \leq t$. From this definition, $\overline{p_j} = p_j$ it no machine processes J_j immediately before time t. On the other hand, if one machine processes J_j throughout some time interval $[s, t]$, but J_j is not processed immediately before time s, then $\overline{p_j} = p_j - (t - s)$.

6.3 Upper bound of competitive ratio

In order to show the upper bound of competitive ratio for all online algorithms, we use the following lemma.

Lemma 15 *In m identical machine scheduling problem, for every integer $k \geq 0$ and for all real numbers r and d with $r < d$, there exists an adversary strategy $\mathscr{S}_m = \mathscr{S}_m(k, r, d)$ with the following properties.*
(1) $\mathscr{S}_m$ creates $(2k+1)m$ jobs. The earliest release time of these jobs is r, and the latest due date of these jobs is d.
(2) There exists a feasible schedule in which all $(2k+1)m$ jobs are early. In such

a schedule, the machines are continuously busy throughout the interval $[r, d]$.
(3) If $u \in \{0, 1, ..., m\}$ machines are unavailable throughout the interval $[r, d]$, the adversary strategy $\mathscr{S}_m$ can prevent any online algorithm from scheduling more than $(2k + 1)(m - u)$ jobs to be early.
(4) The adversary strategy $\mathscr{S}_m$ can prevent any online algorithm from scheduling more than $(2m-1)k+m$ jobs to be early. (i.e., the adversary strategy can force any online algorithm to have at least k tardy jobs.)

Proof. The proof is by induction on k. For $k = 0$, the adversary $\mathscr{S}_m$ releases m jobs with processing time $d - r$ at time r. For $k \geq 1$, the adversary $\mathscr{S}_m$ proceeds as follows. Let $L = \frac{d-r}{8}$, and note that $d = r + 8L$. The adversary releases $2m$ jobs $J_1, J_2, ..., J_{2m}$ with processing times $p_i = 3L$ and $p_{m+i} = 4L$ for $i = 1, 2, ..., m$ at time r. All these $2m$ jobs have due date d. Then the adversary waits until $r + 2L$.

Case 1. If at time $r + 2L$ the online algorithm is processing at least one job J_i for $i \in \{1, 2, ..., m\}$ (i.e.,$p_i = 3L$), then $\mathscr{S}_m$ calls the sub-adversary $\mathscr{S}_m(k - 1, r + 4L, r + 5L)$.
Case 2. Otherwise, $\mathscr{S}_m$ calls the sub-adversary $\mathscr{S}_m(k - 1, r + 3L, r + 4L)$.

(1) The proof of Property (1).

When $k = 0$, Property (1) holds. Considering $k \geq 1$. We assume that Property (1) follows for $k - 1$. Our aim is to prove that Property (1) holds for k. Since $\mathscr{S}_m$ creates $2m$ new jobs $J_1, J_2, ..., J_{2m}$ together with the $(2k - 1)m$ jobs generated by the sub-adversary, Property (1) holds for k.

(2) The proof of Property (2).

To prove Property (2), we consider the following schedules with all jobs early. In case 1, we schedule jobs $J_1, J_2, ..., J_m$ at each of the m machines during the interval $[r + 5L, d]$ respectively; we assign jobs $J_{m+1}, J_{m+2}, ..., J_{2m}$ to all m machines during the interval $[r, r + 4L]$, respectively. Then we process all jobs of the sub-adversary by induction. Similarly, in case 2, we schedule $J_1, J_2, ..., J_m$ at all m machines during the interval $[r, r + 3L]$, respectively; we assign $J_{m+1}, J_{m+2}, ..., J_{2m}$ to all m machines during the interval $[r + 4L, d]$, respectively. Then we process all jobs of the sub-adversary by induction.

(3) The proof of Property (3).

Assume that $u \in \{0, 1, ..., m\}$ machines are unavailable throughout the interval $[r, d]$. We prove Property (3) by induction on k. For $k = 0$ Property (3) holds, i.e., $\mathscr{S}_m$ can prevent any online algorithm from scheduling more than $(m - u)$ jobs to be early with the precondition that u machines are unavailable throughout the interval $[r, d]$. We assume that for $k - 1$ (in $k \geq 1$ case), Property (3) holds, i.e., considering the assumption that u machines are unavailable throughout the interval $[r, d]$, $\mathscr{S}_m(k - 1, r, d)$ can prevent any online algorithm from scheduling more than $(2k - 1)(m - u)$ jobs to be early. Our aim is to prove that Property (3) holds for k. Because the processing times of all $2m$ jobs released by $\mathscr{S}_m$, we know that $m - u$ machines can not schedule more than $2(m-u)$ jobs. No matter how these $2m$ jobs are scheduled, at most m machines have a common available interval $[r + 3L, r + 4L]$ or $[r + 4L, r + 5L]$.

In case 1, by assumption that $\mathscr{S}_m$ can prevent any online algorithm from scheduling more than $(2k-1)(m-u)$ jobs to be early with the precondition that u machines are unavailable, Property (3) holds, since the online algorithm can schedule at most $2(m - u)$ jobs (in $2m$ jobs) together with the $(2k - 1)(m - u)$ jobs generated by the sub-adversary. In case 2, similarly, Property (3) holds.

Considering $(2k+1)(m-u)$, the more machines are available, the more jobs the online algorithm can schedule to be early.

(4) The proof of Property (4).

Now we show that Property (4) holds by induction on k. For $k=0$, Property (4) holds because $\mathscr{S}_m$ creates m jobs. We assume that for $k-1$ (in $k \geq 1$ case), Property (4) holds, i.e., $\mathscr{S}_m(k-1,r,d)$ can prevent any online algorithm from scheduling more than $(2m-1)(k-1)+m$ jobs to be early. Our aim is to prove that Property (4) holds for k.

Firstly, we assume that the online algorithm schedules all $2m$ jobs $J_1, J_2, ..., J_{2m}$ to be early. Then in case 1, there must be one job J_j for $j \in \{m+1, m+2, ..., 2m\}$ (i.e.,$p_j = 4L$) which covers the interval $[r+4L, r+5L]$ on one machine. That is to say, throughout the interval $[r+4L, r+5L]$, at least one machine is unavailable. Then we calls sub-adversary $\mathscr{S}_m = \mathscr{S}_m(k-1, r+4L, r+5L)$. By Property (3), throughout the interval $[r+4L, r+5L]$ the online algorithm can schedule at most $(2k-1)(m-1)$ jobs to be early. Therefore,

$$2m + (2k-1)(m-1) = (2m-1)k + m + (1-k) \leq (2m-1)k + m.$$

Property (4) holds.

Secondly, we assume that the online algorithm does not schedule all $2m$ jobs $J_1, J_2, ...J_{2m}$ to be early, i.e., the online algorithm schedules at most $2m-1$ jobs of these $2m$ jobs to be early. In case 1, if all m machines are available at interval $[r+4L, r+5L]$, by assumption for $k-1$, the sub-adversary $\mathscr{S}_m(k-1, r+4L, r+5L)$ can prevent any online algorithm from scheduling more than $(2m-1)(k-1)+m$ jobs to be early in the interval $[r+4L, r+5L]$. That is to say, the online algorithm can schedule at most $(2m-1)(k-1)+m$ jobs to be early in the interval $[r+4L, r+5L]$. Together with at most $2m-1$ early jobs scheduled by the online algorithm, we know that $(2m-1)+(2m-1)(k-1)+m = (2m-1)k+m$. Property (4) follows. Otherwise, in Case 1, not all m machines are available at interval $[r+4L, r+5L]$, i.e., $u \geq 1$. By Property (3), we know that the sub-adversary $\mathscr{S}_m(k-1, r+4L, r+5L)$ can prevent any online algorithm from scheduling more that $(2k-1)(m-u)$ jobs to be early in the interval $[r+4L, r+5L]$. Since $u \geq 1$, we have $(2k-1)(m-u) \leq (2k-1)(m-1)$. Together with at most $2m-1$ early jobs scheduled by the online algorithm, we have

$$2m - 1 + (2k-1)(m-1) = (2m-2)k + m < (2m-1)k + m.$$

Property (4) follows. In case 2, similar to the proof of case 1, by changing the interval from $[r+4L, r+5L]$ to $[r+3L, r+4L]$, we show that Property (4) also holds. ■

Remark 10 *The Lemma 2 in [3] is a special case of Lemma 1 such that* $m = 1$.

Theorem 11 *For the online problem of scheduling* m *machines to maximize the number of early jobs, every online algorithm* $\mathcal{A}$ *has a competitive ratio* $\rho_{\mathcal{A}} \leq 1 - \frac{1}{2m}$.

Proof. Let C_{OPT} and C_{ON} denote the number of early jobs of an offline optimal algorithm (offline algorithm for short) and that of online algorithm, respectively. By Lemma 15, we have $C_{OPT} = (2k+1)m$ and $C_{ON} \leq (2m-1)k+m$. Therefore,

$$\frac{C_{ON}}{C_{OPT}} \leq \frac{(2m-1)k+m}{(2k+1)m} \to 1 - \frac{1}{2m}, \quad k \to \infty.$$

The theorem follows. ■

Remark 11 *The Theorem 3 in [3] is a special case of Theorem 1 such that* $m = 1$.

6.4 ECT algorithm

In this section, we describe and analyze online algorithm *ECT*, which is based on the *shortest remaining processing time* (SRPT) or ECT rule. Algorithm ECT constructs a schedule of early jobs only, since any tardy jobs can be appended to this schedule in an arbitrary order.

Given a job instance $\mathcal{I}$, **algorithm ECT** runs as follows.

Step 1: Wait until a decision point t, at which a new job is released or at least one of m identical machines becomes available (idle).

Step 2: At any decision point, if there is no idle machine, schedule job J_i such that $\overline{p_i} = \min\{\overline{p_k} | J_k \in \mathcal{I}, r_k \leq t, \overline{p_k} > 0, t + \overline{p_k} \leq d_k\}$ (shortest remaining processing time) to process on the machine with the least current workload; otherwise, schedule job J_i on one of idle machines. Ties are broken by alphabetic order.

Step 3: If at some decision point t, no jobs are subsequently released and $t + \overline{p_k} > d_k$ for all jobs J_k with $r_k \leq t$ and $\overline{p_k} > 0$, stop; else, go to **Step 1**.

Note that ECT algorithm is based on the earliest completion time (ECT) policy, i.e., when ECT algorithm makes a decision to schedule a job J_j with the shortest remaining processing time among available jobs, then if it is not preempted, J_j completes no later than if any other available job were to be processed next. Therefore, a job is preempted only if a newly released job can complete earlier.

Without loss of generality, we assume that the machines are reindexed in increasing order of the job scheduled at the first position. Let $N(S)$ be the number of jobs scheduled in S. Let $\overrightarrow{S}$ and S^* be the (feasible) schedule obtained by ECT algorithm and an optimal (feasible) schedule, respectively. Let $J_k(S)$ be the job with the kth smallest completion time in schedule S. If two jobs are completed at the same time, the job completed on the smallest indexed machine is considered to be completed earlier.

Note that $\overrightarrow{S}$ only contains early jobs and algorithm ECT terminates at time t when all unscheduled jobs in $\mathcal{I}$ cannot meet their respective due dates (i.e., any job $J_j \in \mathcal{I} \backslash J(\overrightarrow{S})$ satisfies the conditions that $t + \overline{p_j} > d_j$ and $\overline{p_j} > 0$). That is to say, once algorithm ECT stops, no job in $\mathcal{I} \backslash J(\overrightarrow{S})$ can be scheduled to be early (by any algorithm).

Theorem 12 *Algorithm ECT is* $\frac{1}{2}$*-competitive.*

Proof. If $S^* = \overrightarrow{S}$, the theorem follows. Therefore, in the following proof, we assume that $S^* \neq \overrightarrow{S}$. In order to prove the theorem, it is sufficient to prove that $N(\overrightarrow{S}) \geq \frac{N(S^*)}{2}$.

For this purpose, we construct a series of feasible schedules $S^0, S^1, ..., S^h = \overrightarrow{S}$ such that $S^0 = S^*$ and $S^1, ..., S^{h-1}$ are different from $\overrightarrow{S}$. Schedule S^q is obtained from S^{q-1} for $q = 1, ..., h$ as follows.

Let $k'(q)$ be the smallest k such that $J_k(S^{q-1}) \neq J_k(\overrightarrow{S})$ for $1 \leq k \leq N(\overrightarrow{S})$. Such a k necessarily exists, since S^{q-1} is different from $\overrightarrow{S}$. From the notation, we have either $k'(q) = 1$ or

$$J_k(S^{q-1}) = J_k(\overrightarrow{S}), \quad k \in \{1, ..., k'(q) - 1\}. \tag{6.1}$$

S^q is obtained by
(i) Deleting job $J_{k'(q)}(S^{q-1})$ from S^{q-1};
(ii) Moving or adding job $J_{k'(q)}(\overrightarrow{S})$ to the place of $J_{k'(q)}(S^{q-1})$ depending on whether the job appears in S^{q-1}.

By construction, S^q is necessarily feasible since $J_{k'(q)}(\overrightarrow{S})$ is the job that can be completed the earliest among all remaining jobs at that time (ECT policy). Furthermore, for $q = 1, ..., h$, we have either $S^q = \overrightarrow{S}$ or

$$k'(q) > k'(q-1), \tag{6.2}$$

$$N(S^q) \geq N(S^{q-1}) - 1. \tag{6.3}$$

From inequality (6.2), we obtain

$$h \leq N(\overrightarrow{S}). \tag{6.4}$$

From inequalities (6.3) and (6.4), we obtain

$$\begin{aligned} N(\overrightarrow{S}) &= N(S^h) \geq N(S^{h-1}) - 1 \geq N(S^{h-2}) - 2 \geq ... \geq N(S^0) - h \\ &= N(S^*) - h \geq N(S^*) - N(\overrightarrow{S}). \end{aligned}$$

As a consequence, $N(\overrightarrow{S}) \geq \frac{N(S^*)}{2}$ and the theorem follows. ■

Chapter 7

Single batch processing machine model

We consider two semi-online scheduling problems on a single batch (processing) machine with jobs' nondecreasing processing times and jobs' nonincreasing processing times, respectively. Our objective is to minimize the makespan. A batch processing machine can handle up to B jobs simultaneously. We study an unbounded model where $B = \infty$. The jobs that are processed together construct a batch, and all jobs in a batch start and complete at the same time. The processing time of a batch is given by the longest processing time of any job in the batch. Jobs arrive over time. Let p_j denote the processing time of job J_j. Given job J_j and its following job J_{j+1}, we assume that $p_{j+1} \geq \alpha p_j$, where $\alpha \geq 1$ is a constant number, for the first problem with jobs' nondecreasing processing times. For the second problem, we assume that $p_{j+1} \leq \alpha p_j$, where $0 < \alpha < 1$ is a constant number. We propose an optimal algorithm for both problems with a competitive ratio $\frac{\sqrt{\alpha^2+4}-\alpha}{2}+1$ for the first problem and $\frac{\sqrt{4\alpha+1}+1}{2}$ for the second problem.

7.1 Introduction

Batch processing machine scheduling has been motivated by burn-in operations in the final testing stage of semiconductor manufacturing in [66] and [65]. Batch scheduling means that a machine can process up to B jobs simultaneously as a batch, and the processing time of a batch is equal to the longest processing time of the jobs assigned to it. Unbounded model means that B is sufficiently large and the bounded model means that B is a constant positive integer.

In this chapter, we consider the third model where jobs arrive over time. Zhang et al. [113] considered the problem of online scheduling on a batch processing machine. They provided an optimal online algorithm with a competitive ratio $\frac{\sqrt{5}+1}{2}$. Based on their results, we study the semi-online problems by using additional jobs' information. We present an optimal semi-online algorithm with a competitive ratio no more than $\frac{\sqrt{5}+1}{2}$. Let p_j denote the processing time of job J_j. Given job J_j and its following job J_{j+1}, we assume that $p_{j+1} \geq \alpha p_j$, $\alpha \geq 1$ is a constant number, in the first problem with jobs' nondecreasing processing

times. For the second problem, we assume that $p_{j+1} \leq \alpha p_j$, $0 < \alpha < 1$ is a constant number.

Online scheduling on a (parallel) batch processing machine has been studied in the last decade. In the unbounded model, for the problem of online scheduling on a single batch machine to minimize the makespan, Zhang et al. [113] and Deng et al. [26] independently provided an online algorithm with a competitive ratio $\frac{\sqrt{5}+1}{2}$. Poon et al. [85] showed that for the same problem in the bounded model, any FBLPT-based (Full Batch Longest Processing Time) algorithm is 2-competitive. Moreover, they presented an algorithm with a competitive ratio $\frac{7}{4}$ for batch size $B = 2$. If a batch is allowed to restart, Fu et al. [35] showed that for minimizing makespan on an unbounded batch machine there is no algorithm with a competitive ratio less than $\frac{5-\sqrt{5}}{2}$. Further, they provided an online algorithm with a competitive ratio $\frac{3}{2}$. For this problem, Fu et al. [34] further considered limited restarts which means that any batch containing a job has already been restarted once cannot be restarted any more. They present an optimal algorithm with a competitive ratio $\frac{3}{2}$. Recently, Nong et al. [80] considered family jobs constraint in the single batch machine scheduling problem. For the unbounded case, they provided an optimal online algorithm with a competitive ratio 2. For the bounded case, they gave a 2-competitive algorithm.

For convenience, we use online algorithm to denote semi-online algorithm in the remainder. The rest of this chapter is organized as follows. In section 2, we give the problem definitions and some notations. In section 3, we present a lower bound $\varphi = \frac{\sqrt{\alpha^2+4}-\alpha}{2} + 1$ ($\alpha \geq 1$) for the first problem and a lower bound $\varphi = \frac{\sqrt{4\alpha+1}+1}{2}$ ($0 < \alpha < 1$) for the second problem, respectively. In section 4, we use the lower bounds obtained in the above section to design algorithm H_α^∞. After that, we prove that this algorithm is optimal for both problems.

7.2 Problem definitions and notations

We are given a job instance $\mathcal{I} = \{J_1, ..., J_n\}$ ($n \geq 2$) where each job J_j is associated with a release time r_j and a processing time p_j. The jobs are to be processed by a batch processing machine of capacity $B = \infty$, i.e., we study unbounded model. The processing time of a batch is the longest processing time of any job in the batch. Jobs arrive over time, i.e., each job's character, such as processing time, becomes known at its arrival. Let p_j denote the processing time of job J_j. Let J_{i+1} denote the following job of J_i. In the first problem with jobs' nondecreasing processing times, we assume that $p_{j+1} \geq \alpha p_j$, where $\alpha \geq 1$ is a constant number. For the second problem, we assume that $p_{j+1} \leq \alpha p_j$, where $0 < \alpha < 1$ is a constant number. Our objective is to minimize the makespan. Using three field notations, our problems can be denoted by $1|online, r_j, B = \infty, nondecreasing|C_{max}$ and $1|online, r_j, B = \infty, nonincreasing|C_{max}$, respectively.

We use $U(t)$ to denote the set of unscheduled jobs available at time t.

7.3 Lower bounds for both problems

In this section, we deal with the problems of online scheduling on a batch machine with jobs' nondecreasing processing times and jobs' nonincreasing processing times, respectively. We study unbounded model, i.e., the batch's size is sufficiently large. We first respectively give a lower bound of competitive ratio for each problem, then we provide an optimal algorithm for both problems. Let $\varphi = \frac{\sqrt{\alpha^2+4}-\alpha}{2} + 1$ when $\alpha \geq 1$ and $\varphi = \frac{\sqrt{4\alpha+1}+1}{2}$ when $0 < \alpha < 1$. Note that $1 < \varphi \leq \frac{\sqrt{5}+1}{2}$ and we will use it in the remainder for simplicity.

Given an job instance $\mathcal{I}$, let C_{ON} and C_{OPT} denote the makespan obtained by an online algorithm $\mathcal{A}_{ON}$ and the value of an optimal (offline) schedule, respectively.

Lemma 16 *For $1|online, r_j, B = \infty, nondecreasing|C_{max}$, there is no algorithm with a competitive ratio less than $\frac{\sqrt{\alpha^2+4}-\alpha}{2} + 1$, where $\alpha \geq 1$.*

Proof. For any online algorithm A_{ON}, we construct a special job instance $\mathcal{I}$ such that C_{ON}/C_{OPT} is as large as possible. Let ϵ be a sufficiently small positive number. At time 0, we give the first job J_1 with $p_1 = 1$. We assume that A_{ON} schedules this job at time $T_{\mathcal{A}}$. Depending on $T_{\mathcal{A}}$, we discuss the following two cases.

Case 1. $T_{\mathcal{A}} \geq \frac{\sqrt{\alpha^2+4}-\alpha}{2}$.

No job arrives in future. We know that $C_{ON} \geq \frac{\sqrt{\alpha^2+4}-\alpha}{2} + 1$. An optimal scheme consists of scheduling J_1 at time 0, i.e., $C_{OPT} = 1$. Therefore,

$$\frac{C_{ON}}{C_{OPT}} \geq \frac{\sqrt{\alpha^2+4}-\alpha}{2} + 1 = \varphi.$$

Case 2. $0 \leq T_{\mathcal{A}} < \frac{\sqrt{\alpha^2+4}-\alpha}{2}$.

We further generate job J_2 with $p_2 = \alpha$ at time $T_{\mathcal{A}} + \epsilon$. We obtain $C_{ON} \geq T_{\mathcal{A}} + 1 + \alpha$. An optimal schedule consists of scheduling J_1 and J_2 in a batch at time $T_{\mathcal{A}} + \epsilon$. Therefore, $C_{OPT} = T_{\mathcal{A}} + \epsilon + \alpha$ due to $\alpha \geq 1$. It follows that

$$\frac{C_{ON}}{C_{OPT}} \geq \frac{T_{\mathcal{A}} + 1 + \alpha}{T_{\mathcal{A}} + \epsilon + \alpha} = 1 + \frac{1}{T_{\mathcal{A}} + \alpha} > 1 + \frac{1}{\frac{\sqrt{\alpha^2+4}-\alpha}{2} + \alpha} = \varphi, \quad \epsilon \to 0.$$

According to the above two cases, the lemma holds. ■

Lemma 17 *For $1|online, r_j, B = \infty, nonincreasing|C_{max}$, there is no algorithm with a competitive ratio less than $\frac{\sqrt{4\alpha+1}+1}{2}$, where $0 < \alpha < 1$.*

Proof. For any online algorithm A_{ON}, we construct a special job instance $\mathcal{I}$ such that C_{ON}/C_{OPT} is as large as possible. Let ϵ be a sufficiently small positive number. At time 0, we give the first job J_1 with $p_1 = 1$. We assume that A_{ON} scheduling this job at time $T_{\mathcal{A}}$. Depending on $T_{\mathcal{A}}$, we discuss the following two cases.

Case 1. $T_{\mathcal{A}} \geq \frac{\sqrt{4\alpha+1}-1}{2}$.

No job arrives in future. We know that $C_{ON} \geq \frac{\sqrt{4\alpha+1}+1}{2}$. An optimal scheme consists of scheduling J_1 at time 0, i.e., $C_{OPT} = 1$. Therefore,

$$\frac{C_{ON}}{C_{OPT}} \geq \frac{\sqrt{4\alpha+1}+1}{2} = \varphi.$$

Case 2. $0 \leq T_{\mathcal{A}} < \frac{\sqrt{4\alpha+1}-1}{2}$.

We further give job J_2 with $p_2 = \alpha$ at time $T_{\mathcal{A}} + \epsilon$. We obtain $C_{ON} \geq T_{\mathcal{A}} + 1 + \alpha$. An optimal schedule consists of scheduling J_1 and J_2 in a batch at time $T_{\mathcal{A}} + \epsilon$. Therefore, $C_{OPT} = T_{\mathcal{A}} + \epsilon + 1$ due to $\alpha < 1$. It follows that

$$\frac{C_{ON}}{C_{OPT}} \geq \frac{T_{\mathcal{A}} + 1 + \alpha}{T_{\mathcal{A}} + \epsilon + 1} = 1 + \frac{\alpha}{T_{\mathcal{A}} + 1} > 1 + \frac{\alpha}{\frac{\sqrt{4\alpha+1}-1}{2} + 1} = \varphi, \quad \epsilon \to 0.$$

According to the above cases, the lemma holds. ■

7.4 An optimal algorithm for both problems

Considering the proofs of two lower bounds of competitive ratios, we realize that the algorithm should wait for a while rather than scheduling the jobs immediately after their arrivals. We shift the release time of job J_j to $\overline{r}_j = \varphi r_j + (\varphi - 1)p_j$ in order to contain the waiting time. This equation can guarantee that if J_j is scheduled at time $\varphi r_j + (\varphi - 1)p_j$, its completion time cannot exceed φ times the optimal value, i.e., $\varphi r_j + (\varphi - 1)p_j + p_j = \varphi(r_j + p_j)$. This idea is used in [113]. We design the following algorithm using the same idea.

Algorithm H_α^∞ works as follows.

Step 1: Wait until a decision point, where the batch machine is idle and at least one job is available (If all jobs have been scheduled, output the schedule). Suppose this happens at time t. Choose a job J_j with the longest processing time in $U(t)$.

Step 2: If $\overline{r}_j \leq t$, schedule all jobs in $U(t)$ as a single batch;
otherwise, wait until a new job arrive or until time $\overline{r}_j$, whichever happens first.

Step 3: Go to **Step 1**.

We adopt some notations and definitions from [113]. Given an job instance, we assume that H_α^∞ generate m batches in total. We index these batches in nondecreasing order of their completion times. For convenience, in batch i, denote by $J_{(i)}$ the job with the longest processing time in that batch. Let $p_{(i)}$ and $r_{(i)}$ be the processing time and the release time (or arrival time) of job $J_{(i)}$, respectively. Let $s_{(i)}$ be the starting time of batch i. Note that batch i is processed either at time $\varphi r_{(i)} + (\varphi - 1)p_{(i)}$ or after batch $i-1$ is finished at time $s_{(i-1)} + p_{(i-1)}$.

If batch i starts at time $\varphi r_{(i)} + (\varphi - 1)p_{(i)}$, we call it a *regular* batch; otherwise, it is called a *delayed* batch.

Similar to Lemma 2 in [113], we have the following lemma.

Lemma 18 *If batch i is a regular batch, then batch $i+1$ or batch $i+2$ is also a regular batch.*

Proof. We prove this lemma for two cases: $\alpha \geq 1$ and $0 < \alpha < 1$.

(1) $\alpha \geq 1$.

By contradiction. Suppose both batches $i+1$ and $i+2$ are delayed batches. We know that each job in batch $i+1$ arrives later than the starting time of batch i. Therefore, considering job $J_{(i+1)}$, we have $r_{(i+1)} > \varphi r_{(i)} + (\varphi - 1)p_{(i)}$. Since batch $i+1$ is a delayed batch, we obtain

$$\varphi(r_{(i)} + p_{(i)}) > \varphi r_{(i+1)} + (\varphi - 1)p_{(i+1)} > \varphi^2 r_{(i)} + \varphi(\varphi - 1)p_{(i)} + (\varphi - 1)p_{(i+1)}.$$

It follows that

$$(2\varphi - \varphi^2)p_{(i)} > (\varphi^2 - \varphi)r_{(i)} + (\varphi - 1)p_{(i+1)} \geq (\varphi - 1)p_{(i+1)}. \tag{7.1}$$

Considering $\varphi = \frac{\sqrt{\alpha^2+4}-\alpha}{2} + 1$ and $\alpha \geq 1$, we have

$$p_{(i+1)} < \frac{2\varphi - \varphi^2}{\varphi - 1} p_{(i)} = \left[\frac{1}{\varphi - 1} - (\varphi - 1)\right] p_{(i)} = \alpha p_{(i)}.$$

This contradicts to the assumption that $p_{(i+1)} \geq \alpha p_{(i)}$ for $1|online, r_j, B = \infty, nondecreasing|C_{max}$.

(2) $0 < \alpha < 1$.

By contradiction. Suppose both batches $i+1$ and $i+2$ are delayed batches. We know that each job in batch $i+2$ arrives later than the starting time of batch $i+1$. Therefore, considering job $J_{(i+2)}$, we have $r_{(i+2)} > \varphi(r_{(i)} + p_{(i)})$. Since batch $i+2$ is a delayed batch, we obtain

$$\varphi(r_{(i)} + p_{(i)}) + p_{(i+1)} > \varphi r_{(i+2)} + (\varphi - 1)p_{(i+2)} > \varphi^2(r_{(i)} + p_{(i)}) + (\varphi - 1)p_{(i+2)}.$$

It follows that

$$p_{(i+1)} > (\varphi^2 - \varphi)(r_{(i)} + p_{(i)}) + (\varphi - 1)p_{(i+2)} \geq (\varphi^2 - \varphi)p_{(i)} = \varphi(\varphi - 1)p_{(i)}. \tag{7.2}$$

Considering $\varphi = \frac{\sqrt{4\alpha+1}+1}{2}$ and $\alpha < 1$, we have $\varphi(\varphi - 1) = \alpha$, which implies

$$p_{(i+1)} > \alpha p_{(i)}.$$

This contradicts to the assumption that $p_{(i+1)} \leq \alpha p_{(i)}$ for $1|online, r_j, B = \infty, nonincreasing|C_{max}$.

The lemma follows. ■

Corollary 1 *In the schedule obtained by algorithm H_α^∞, there do not exist two successive delayed batches.*

Proof. By the algorithm, the first batch must be a regular batch. Then we have the desired result. ■

Let C_H and C^* denote the value obtained by algorithm H_α^∞ and the optimal objective value (for an instance), respectively.

Lemma 19 *If the last batch is a regular batch, i.e., $s_{(m)} = \varphi r_{(m)} + (\varphi - 1)p_{(m)}$, then $C_H/C^* \leq \varphi$.*

Proof. Since $C_H = s_{(m)} + p_{(m)} = \varphi(r_{(m)} + p_{(m)})$ and $C^* \geq r_{(m)} + p_{(m)}$, the lemma follows. ■

Theorem 13 *Algorithm H_α^∞ is optimal for both problems: $1|online, r_j, B = \infty, nondecreasing|C_{max}$ and $1|online, r_j, B = \infty, nonincreasing|C_{max}$. ($\alpha \geq 1$ in the first problem and $0 < \alpha < 1$ in the second problem.)*

Proof. By contradiction. By Lemma 19, we only need to consider the case where the last batch m is a delayed batch. By algorithm, we know that the first batches must be a regular batch. Thus, in this case, there must be some batch processed before batch m. By Corollary 1, there do not exist two successive

delayed batches in the schedule obtained by algorithm H_α^∞. Therefore, batch $m-1$ must be a regular batch. It follows that

$$s_{(m-1)} = \varphi r_{(m-1)} + (\varphi - 1)p_{(m-1)} \geq (\varphi - 1)p_{(m-1)}. \tag{7.3}$$

We know

$$C_H = s_{(m-1)} + p_{(m-1)} + p_{(m)}. \tag{7.4}$$

Note that

$$r_{(m)} > s_{(m-1)}. \tag{7.5}$$

In the following, we respectively discuss two cases: $\alpha \geq 1$ and $0 < \alpha < 1$.

(1) $\alpha \geq 1$.

We further discuss two cases depending on whether $J_{(m)}$ and $J_{(m-1)}$ are in the same batch in an optimal schedule.

Case 1. In an optimal schedule, $J_{(m)}$ and $J_{(m-1)}$ are in the same batch.

By the assumption that $p_{j+1} \geq \alpha p_j$, we have $C^* \geq r_{(m)} + p_{(m)}$ and $p_{(m)} \geq \alpha p_{(m-1)}$. Note that $\varphi = \frac{\sqrt{\alpha^2+4}-\alpha}{2} + 1$.

Considering equation (7.4) and inequalities (7.3) and (7.5), it follows

$$\frac{C_H}{C^*} \leq \frac{s_{(m-1)} + p_{(m-1)} + p_{(m)}}{r_{(m)} + p_{(m)}} < 1 + \frac{p_{(m-1)}}{s_{(m-1)} + p_{(m)}} \leq 1 + \frac{1}{\varphi - 1 + \alpha} = \varphi.$$

Case 2. In an optimal schedule, $J_{(m)}$ and $J_{(m-1)}$ are in different batches.

It follows that $C^* \geq r_{(m-1)} + p_{(m-1)} + p_{(m)}$. Considering equations (7.4) and (7.3), we have

$$\frac{C_H}{C^*} \leq \frac{s_{(m-1)} + p_{(m-1)} + p_{(m)}}{r_{(m-1)} + p_{(m-1)} + p_{(m)}} = \frac{\varphi(r_{(m-1)} + p_{(m-1)}) + p_{(m)}}{r_{(m-1)} + p_{(m-1)} + p_{(m)}} < \varphi.$$

(2) $0 < \alpha < 1$.

We discuss two cases depending on whether $J_{(m)}$ and $J_{(m-1)}$ are in the same batch in an optimal schedule.

Case 1. In an optimal schedule, $J_{(m)}$ and $J_{(m-1)}$ are in the same batch.

By the assumption that $p_{j+1} \leq \alpha p_j$, we have $C^* \geq r_{(m)} + p_{(m-1)}$ and $p_{(m)} \leq \alpha p_{(m-1)}$. Note that $\varphi = \frac{\sqrt{4\alpha+1}+1}{2}$.

Considering equation (7.4) and inequalities (7.3) and (7.5), it follows

$$\frac{C_H}{C^*} \leq \frac{s_{(m-1)} + p_{(m-1)} + p_{(m)}}{r_{(m)} + p_{(m-1)}} < 1 + \frac{p_{(m)}}{s_{(m-1)} + p_{(m-1)}} \leq 1 + \frac{\alpha}{\varphi} = \varphi.$$

Case 2. In an optimal schedule, $J_{(m)}$ and $J_{(m-1)}$ are in different batches.

It follows that $C^* \geq r_{(m-1)} + p_{(m-1)} + p_{(m)}$. Considering equations (7.4) and (7.3), we have

$$\frac{C_H}{C^*} \leq \frac{s_{(m-1)} + p_{(m-1)} + p_{(m)}}{r_{(m-1)} + p_{(m-1)} + p_{(m)}} = \frac{\varphi(r_{(m-1)} + p_{(m-1)}) + p_{(m)}}{r_{(m-1)} + p_{(m-1)} + p_{(m)}} < \varphi.$$

The theorem follows. ■

Chapter 8

Open shop model

This chapter deals with a two-machine open shop problem. The objective is to minimize the makespan. Jobs arrive over time. We study preemption-resume model, i.e., the currently processed job may be preempted at any moment in time, and it may be resumed at any later moment in time. Let $p_{1,j}$ and $p_{2,j}$ denote the processing time of job J_j on machines M_1 and M_2, respectively. Bounded processing times mean that $1 \leq p_{i,j} \leq \alpha$ $(i = 1, 2)$ for each job J_j, where $\alpha \geq 1$ is a constant number. We propose an optimal online algorithm with a competitive ratio $\frac{5\alpha-1}{4\alpha}$.

8.1 Introduction

In this chapter, we consider the online model where jobs arrive over time. In reality, the processing time of a job could not be enormously large. So we focus on the problem with jobs' bounded processing times. In the literature, there are some results about (offline) shop scheduling problems [84, 13]. This chapter concerns about online shop scheduling problem. Chen et al. [18] investigated the problem of on-line scheduling open shops of two and three machines to minimize the makespan. They studied the first online model where jobs arrive one by one. They proposed a 1.848-competitive algorithm for the non-preemptive scheduling problem of two machines and showed a lower bound of 1.754. Then they developed a (27/19)-competitive algorithm for the preemptive scheduling problem of three machines. A.P.A. Vestjens [108] addressed the online two-machine open shop scheduling problem. They considered the third model where jobs arrive over time. They proposed an optimal algorithm with a competitive ratio 3/2 for non-preemptive situation. Moreover, they studied the preemptive-resume scenario, and presented a 5/4-competitive algorithm which was shown to be optimal later. Based on the results of Vestjens, we further consider the situation with jobs' bounded processing times, which is more practical. For instance, jobs' processing times are bounded in a interval $[1, \alpha]$ where $\alpha \geq 1$.

The remainder of this chapter is organized as follows. Section 2 contains the problem definition, some notations and preliminaries. In section 3, we show a lower bound of competitive ratio $\frac{5\alpha-1}{4\alpha}$. In section 4, we propose an algorithm SLICE-α and show that it is optimal with a competitive ratio $\frac{5\alpha-1}{4\alpha}$.

8.2 Problem definition, notations and preliminaries

We are given two machines, denoted by M_1 and M_2, and an job instance $\mathcal{I} = \{J_1, ..., J_n\}$. Each job J_j is released at time r_j. We denote each job J_j by an ordered pair $(p_{1,j}, p_{2,j})$, where $p_{1,j}$ and $p_{2,j}$ are the processing times of two operations $O_{1,j}$ and $O_{2,j}$ on machines M_1 and M_2. We study the open shop problem, i.e., job J_j may be processed first on M_1 then on M_2 or vice versa; the decision maker may determine the routes. We study the preemption-resume model, which means that the currently processed job may be preempted at any moment in time, and it may be resumed at any later moment in time. Jobs arrive over time, i.e., each job's character, such as processing time, becomes known at its arrival. We assume that $1 \leq p_{1,j}, p_{2,j} \leq \alpha$, where $\alpha \geq 1$ is a constant number. Our objective is to minimize the makespan. Using three-field notation, the problem can be denoted by $O2|online, r_j, 1 \leq p_{\cdot,j} \leq \alpha|C_{max}$.

Let $C_{max}(\sigma)$ denote the makespan of the schedule σ obtained by an algorithm. For a set of jobs, say $S \in \mathcal{I}$, let $P_1(S) = \sum_{J_j \in S} p_{1,j}$, $P_2(S) = \sum_{J_j \in S} p_{2,j}$ and $L(S) = \max_{J_j \in S}\{r_j + p_{1,j} + p_{2,j}\}$. Let $r(S) = min_{J_j \in S} r_j$. We define $Z^* = \max_{S \subseteq \mathcal{I}}\{r(S) + P_1(S), r(S) + P_2(S), L(S)\}$. Clearly, the minimum makespans of an optimal preemptive schedule ϖ and an optimal non-preemptive schedule π satisfy $C_{max}(\pi) \geq C_{max}(\varpi) \geq Z^*$.

We introduce a lemma which states some structural information of an optimal schedule. We will use these structural information to design algorithm SLICE-α and to prove the optimality of the algorithm.

Lemma 20 *[39] For any instance $\mathcal{I}$ of the two-machine open shop with all jobs being released at time 0, $C_{max}(\pi) \geq C_{max}(\varpi) \geq Z^* = \max\{P_1(\mathcal{I}), P_2(\mathcal{I}), L(\mathcal{I})\}$. Furthermore, the structure of any optimal schedule is as follows:*
(1) If $P_1(\mathcal{I}) \geq \max\{P_2(\mathcal{I}), L(\mathcal{I})\}$, then M_1 is busy all the time.
(2) If $P_2(\mathcal{I}) \geq \max\{P_1(\mathcal{I}), L(\mathcal{I})\}$, then M_1 is busy all the time.
(3) If $L(\mathcal{I}) > \max\{P_1(\mathcal{I}), P_2(\mathcal{I})\}$, then the job J_j such that $p_{1,j} + p_{2,j} = L(\mathcal{I})$ is processed all the time either on M_1 or on M_2, and all other jobs are processed in the remaining available machine time.

8.3 A lower bound

In this section, we deal with the two-machine open shop problem with bounded processing times. We give the following lemma to show a lower bound of the competitive ratio.

Given a job instance $\mathcal{I}$, let C_{ON} and C_{OPT} denote the makespan obtained by an online algorithm $\mathcal{A}_{ON}$ and the makespan of an optimal (offline) schedule, respectively. Let $\phi = \frac{5\alpha - 1}{4\alpha}$ and we use this notation in the remainder for simplicity.

Theorem 14 *For $O2|online, r_j, 1 \leq p_{\cdot j} \leq \alpha|C_{max}$, there is no algorithm with a competitive ratio less than ϕ.*

Proof. For any online algorithm $\mathcal{A}_{ON}$, we construct a special job instance $\mathcal{I}$ such that C_{ON}/C_{OPT} is as large as possible. At time 0, we give the first job

$J_1 = (\alpha, \alpha)$. We assume that $\mathcal{A}_{ON}$ schedules this job at time T. Depending on T, we discuss the following two cases. Note that $\alpha - 1 \geq 0$.

Case 1. $T \geq \alpha - 1$.

No job arrives in future. We know that $C_{ON} \geq T + 2\alpha \geq 3\alpha - 1$. An optimal scheme consists of scheduling J_1 at time 0, i.e., $C_{OPT} = 2\alpha$. Therefore,

$$\frac{C_{ON}}{C_{OPT}} \geq \frac{3\alpha - 1}{2\alpha} = 1 + \frac{\alpha - 1}{2\alpha} > 1 + \frac{\alpha - 1}{4\alpha} = \phi.$$

Case 2. $0 \leq T < \alpha - 1$.

We know $\alpha - T > 1$. We denote by $p^l_{1,1}$ and $p^l_{2,1}$ the respective parts of $O_{1,1}$ and $O_{2,1}$ that have been processed by time $\alpha - 1$, and by $p^r_{1,1}$ and $p^r_{2,l}$ the respective parts of $O_{1,1}$ and $O_{2,1}$ that are processed after $\alpha - 1$. It follows that $p^l_{1,1} + p^l_{2,1} \leq \alpha - 1 - T$, which also implies that none of the operations is completed at time $\alpha - 1$. By symmetry, without loss of generality, we assume that $p^l_{1,1} \geq p^l_{2,1}$. Therefore, $p^l_{2,1} \leq \frac{\alpha - 1 - T}{2}$. Considering $p^l_{2,1} + p^r_{2,1} = \alpha$, we have $p^r_{2,1} \geq \frac{\alpha + 1 + T}{2}$.

We further generate job $J_2 = (1, \alpha)$ at time $\alpha - 1$. Since $p^r_{2,1} \geq \frac{\alpha + 1 + T}{2}$ and $T \geq 0$, we have $C_{ON} \geq (\alpha - 1) + \alpha + p^r_{2,1} \geq \frac{5\alpha - 1}{2}$. An optimal schedule consists of processing $O_{2,1}$ at time 0 on M_2 followed by $O_{2,2}$ and processing $O_{1,2}$ at time $\alpha - 1$ on M_1 followed by $O_{1,1}$. Thus, $C_{OPT} = 2\alpha$. It follows

$$\frac{C_{ON}}{C_{OPT}} \geq \frac{5\alpha - 1}{4\alpha} = \phi.$$

The theorem follows. ■

8.4 An optimal algorithm

In this section, we show an optimal online algorithm **SLICE-α** with a competitive ratio which matches the lower bound ϕ. Based on *SLICE* Algorithm proposed in [108], we develop a new algorithm for the concerning problem. Since $\phi \leq \frac{5}{4}$, we will show that SLICE-α algorithm outperforms SLICE algorithm under the considered circumstances.

For a job instance, let σ denote the schedule obtained by algorithm SLICE-α. We first recall some notations used in [3]. For any time t, define the *surviving job* of a job J_j to be the part of J_j that still needs to be processed in σ after time t. Let $p_{1,j}(t)$ and $p_{2,j}(t)$ denote the (remaining) processing times of this surviving job J_j at time t. We denote by $S(t)$ the set of surviving jobs at time t. We denote by $P_1(t)$ the total processing times of M_1 operations in $S(t)$, i.e., $P_1(t) = \sum_{J_j \in S(t)} p_{1,j}(t)$, by $P_2(t)$ the total processing times of M_2 operations in $S(t)$, i.e., $P_2(t) = \sum_{J_j \in S(t)} p_{2,j}(t)$. We use $L(t)$ to denote the total processing time of the largest job J_k in $S(t)$, i.e., $L(t) = p_{1,k}(t) + p_{2,k}(t) = \max_{J_j \in S(t)}\{p_{1,j}(t) + p_{2,j}(t)\}$. We also say that J_k determines $L(t)$. Note that, given $S(t)$, if no more jobs are released after time t, then the optimal makespan is $t + \max\{P_1(t), P_2(t), L(t)\}$ according to Lemma 20.

Algorithm SLICE-α works as follows.

Step 1: Wait until a release time r_j of job J_j, compute $P_1(r_j)$, $P_2(r_j)$ and $L(r_j)$ (If all jobs have been scheduled, output the schedule).

Step 2: If $L(r_j) > \max\{P_1(r_j), P_2(r_j)\}$, do the following until a new job is

released (and go to Step 1).
Step 2.1: If J_j determines $L(r_j)$, do the following.
Step 2.1.1: At time r_j process $O_{1,j}$ until time $r_j+\min\{p_{1,j}, (\phi-1)(p_{1,j}+p_{2,j})\}$ on M_1.
Step 2.1.2: if $p_{2,j} > \min\{p_{1,j}, (\phi-1)(p_{1,j}+p_{2,j})\}$, process $O_{2,j}$ until time $r_j + p_{2,j}$ on M_2.
Step 2.1.3: Processing the remaining portion of $O_{1,j}$ on M_1, which is followed by the remaining portion of $O_{2,j}$ on M_2.
Step 2.2: If another job J_l determines $L(r_j)$, then maintain the schedule for J_l determined at the previous release time.
Step 2.3: Process the other jobs in $S(r_j)$ on both machines as much as possible, and preempt if necessary.
Step 3: If $L(r_j) \leq \max\{P_1(r_j), P_2(r_j)\}$, do the following schedule $\mathscr{S}$ until a new job is released (and go to Step 1). Construct an optimal schedule $\mathscr{S}$ for the jobs in $S(r_j)$, satisfying the extra condition that at the beginning both machines are busy and in the remaining part only one machine is busy. This can easily be done according to Lemma 20, and by preempting a few jobs if necessary.

The intuitive idea behind algorithm SLICE-α is similar to that of algorithm SLICE. At time r_j, a busiest slice of an optimal schedule for all jobs in $S(r_j)$ is assigned into the interval $[r_j, r_{j+1}]$. At time r_j, if the largest surviving job determines the makespan of the optimal schedule for all jobs in $S(r_j)$, this slice is scheduled in such a way that the two operations of the largest surviving job are reduced in a proper amount related to their processing times. The algorithm tries to remain the currently largest job to be largest one at the next release time r_{j+1}, except for possibly J_{j+1}. If the total processing time on one machine determines the makespan, the algorithm keeps both machines busy during $[r_j, r_{j+1}]$ as long as possible.

In order to prove that algorithm SLICE-α is ϕ-competitive by contradiction, we assume that there exists a smallest counterexample, denoted by $\mathcal{I}$, which consists of a minimum number of jobs. For this smallest counterexample $\mathcal{I}$, the competitive ratio of algorithm SLICE-α is greater than ϕ. This idea was used in [108] to prove that SLICE algorithm is 5/4-competitive. We first show several properties of this smallest counterexample. Then we will derive a contradiction to the existence of such a counterexample. We use σ and π to denote the schedule obtained by algorithm SLICE-α and an optimal schedule for this smallest counterexample $\mathcal{I}$, respectively. Thus, $C_{max}(\sigma)$ and $C_{max}(\pi)$ denote the makespan obtained by SLICE-α and that of an optimal schedule for $\mathcal{I}$, respectively.

Lemma 21 *In σ, there is no simultaneous idle time on both machines before time $C_{max}(\sigma)$.*

Proof. By contradiction. Suppose that there exists simultaneous idle time on both machines before $C_{max}(\sigma)$. Deleting all jobs scheduled before this idle time neither decreases $C_{max}(\sigma)$ nor increases $C_{OPT}(\pi)$. This contradicts the minimality of $\mathcal{I}$. Therefore, the lemma follows. ■

Furthermore, if we modify the smallest counterexample $\mathcal{I}$ by decreasing all release times such that the first job arrives (or is released) at time 0, the modified instance is a smallest counterexample too. Because this modification does not

decrease the value of $C_{max}(\sigma)/C_{max}(\pi)$. Therefore, without loss of generality, we assume that the first job in $\mathcal{I}$ is released at time 0. By symmetry of two machines, without loss of generality, we assume that M_2 finishes last in σ.

Lemma 22 *In σ, there exists idle time on M_2.*

Proof. By contradiction. Suppose that there is no idle time on M_2. Since the first job arrives at time 0 and M_2 finishes last in σ, we know that σ is an optimal schedule. There is a contradiction and the lemma follows. ■

Let $r_k \geq 0$ be the last release time before the last idle time on M_2 finishes. We know that idle time exists at sometime during $[r_k, C_{max}(\sigma)]$. Let J_{k+1} be the following job of J_k such that $r_{k+1} > r_k$ in $\mathcal{I}$. We have the following lemma.

Lemma 23 *In σ, there is no idle time on M_2 throughout $[r_{k+1}, C_{max}(\sigma)]$.*

Proof. By contradiction. Suppose that there is idle time on M_2 during $[r_{k+1}, C_{max}(\sigma)]$. This contradicts to the definition of r_k. ■

Corollary 2 *In σ, there is idle time on M_2 during $[r_k, r_{k+1}]$.*

Lemma 24 *Algorithm SLICE-α executes Step 2 (not Step 3) at release time r_k.*

Proof. By contradiction. Suppose that SLICE-α executes Step 3 (not Step 2) at time r_k. Then the algorithm must create a schedule in which M_2 is contiguously busy until all jobs in $S(r_k)$ are finished on this machine. By Corollary 2, all jobs in $S(r_k)$ must finish earlier than r_{k+1} on M_2. This implies that all jobs processed on M_2 after (or at) time r_{k+1} are released at time r_{k+1} or later. By Lemma 23, we have that σ is an optimal schedule. There is a contradiction. The lemma holds. ■

Let J_l be the largest job in $S(r_k)$ and hence determine $L(r_k)$. We can observe that J_l is the only job in $S(r_k)$ that may still need processing on M_2 after time r_k. The following lemma indicates that the surviving job of J_l has been the largest job from its release time r_l.

Lemma 25 *At any release time t such that $r_l \leq t \leq r_k$, SLICE-α executes Step 2 (not Step 3) and the surviving job of J_l always determines $L(t)$.*

Proof. By induction. By Lemma 24, we know that at time r_k the algorithm executes Step 2. By the definition of J_l, we know that the surviving job of J_l determines $L(r_k)$. Assume that at release time r_j such that $r_l < r_j \leq r_k$, the algorithm executes Step 2 and the surviving job of J_l determines $L(r_j)$. Therefore, at r_j the algorithm must execute Step 2.2 since J_l is released before r_j. The execution of Step 2.2 implies that at the previous release time t_{j-1}, J_l determines $L(r_{j-1})$, i.e., $L(r_{j-1}) = p_{1,l}(r_{j-1}) + p_{2,l}(r_{j-1})$. Therefore, we only need to show that at release time r_{j-1}, the algorithm also executes Step 2.

By contradiction. Suppose that at time r_{j-1}, the algorithm executes Step 3 (not Step 2). By Lemma 20, all jobs in $S(r_{j-1})$, including the surviving job of J_l at r_{j-1} must be scheduled in such a way that they are processed in a period of length $\max\{P_1(r_{j-1}), P_2(r_{j-1})\}$. This implies that

$$r_{j-1} + \max\{P_1(r_{j-1}), P_2(r_{j-1})\} \geq r_j + p_{1,l}(r_j) + p_{2,l}(r_j). \tag{8.1}$$

Since at time r_j, the algorithm executes Step 2 and the surviving job $J_l = (p_{1,l}(r_j), p_{2,l}(r_j))$ is the largest surviving job. Then we have $p_{1,l}(r_j) + p_{2,l}(r_j) > \max\{P_1(r_j), P_2(r_j)\}$. Together with inequality (8.1),

$$r_{j-1} + \max\{P_1(r_{j-1}), P_2(r_{j-1})\} > r_j + \max\{P_1(r_j), P_2(r_j)\}.$$

It follows that

$$\max\{P_1(r_{j-1}), P_2(r_{j-1})\} - (r_j - r_{j-1}) > \max\{P_1(r_j), P_2(r_j)\}. \tag{8.2}$$

We further consider two possible cases.

Case 1. $P_1(r_{j-1}) \geq P_2(r_{j-1})$.

By Lemma 20 and the fact that there is no idle time on M_2 throughout $[0, r_k]$, we know M_1 is busy throughout $[r_{j-1}, r_j]$. This implies that

$$P_1(r_{j-1}) - P_1(r_j) = r_j - r_{j-1}. \tag{8.3}$$

Consider inequality (8.2), we have that

$$P_1(r_{j-1}) > r_j - r_{j-1} + \max\{P_1(r_j), P_2(r_j)\} \geq r_j - r_{j-1} + P_1(r_j).$$

This contradicts to inequality (8.3).

Case 2. $P_1(r_{j-1}) < P_2(r_{j-1})$.

Since there is no idle time on M_2 throughout $[0, r_k]$, we know M_2 is busy throughout $[r_{j-1}, r_j]$. This implies that

$$P_2(r_{j-1}) - P_2(r_j) = r_j - r_{j-1}. \tag{8.4}$$

Consider inequality (8.2), we have that

$$P_2(r_{j-1}) > r_j - r_{j-1} + \max\{P_1(r_j), P_2(r_j)\} \geq r_j - r_{j-1} + P_2(r_j).$$

This contradicts to inequality (8.4).

Therefore, the lemma follows. ■

Lemma 26 *In σ, J_l cannot be completed before time r_{k+1}.*

Proof. By contradiction. Suppose that J_l is completed at time $C_l(\sigma)$ such that $r_k \leq C_l(\sigma) < r_{k+1}$. By Step 2 of the algorithm, all jobs in $S(r_k)$ are completed not later than $C_l(\sigma)$. There exists simultaneous idle time throughout $[C_l(\sigma), r_{k+1}]$. This contradiction to Lemma 21. The lemma holds. ■

Corollary 3 *J_l is processed thoughout $[r_l, r_{k+1}]$, either on M_1 or on M_2.*

We denote by $p^l_{1,l}$ and $p^l_{2,l}$ the respective parts of $O_{1,l}$ and $O_{2,l}$ that have been processed by time r_{k+1}, and by $p^r_{1,l}$ and $p^r_{2,l}$ the respective parts that are processed after r_{k+1}. We know that

$$p^l_{1,l} + p^r_{1,l} = p_{1,l}. \tag{8.5}$$

$$p^l_{2,l} + p^r_{2,l} = p_{2,l}. \tag{8.6}$$

By Corollary 3, we further have

$$p^l_{1,l} + p^l_{2,l} = r_{k+1} - r_l. \tag{8.7}$$

Recall that $\phi = \frac{5\alpha - 1}{4\alpha}$ and $\alpha \geq 1$.

Lemma 27 *$p^l_{1,l} > (\phi - 1)C_{max}(\pi)$ and $p^r_{2,l} > (\phi - 1)C_{max}(\pi)$.*

Proof. Let $\overline{P}_2$ be the total time spent by M_2 on processing jobs other than J_l from time r_{k+1} onwards, i.e., $\overline{P}_2 = C_{max}(\sigma) - r_{k+1} - p^r_{2,l},$. Because $\mathcal{I}$ is a counterexample, we have

$$\phi C_{max}(\pi) < C_{max}(\sigma) = r_{k+1} + p^r_{2,l} + \overline{P}_2. \tag{8.8}$$

By Corollary 2, we know that all jobs that contribute to $\overline{P}_2$ are released at time r_{k+1} or later. Otherwise, M_2 could not be idle sometime during $[r_k, r_{k+1}]$. Therefore, $r_{k+1} + \overline{P}_2 \leq C_{max}(\pi)$. Together with inequality (8.8), we have

$$\phi C_{max}(\pi) \leq C_{max}(\pi) + p^r_{2,l}.$$

It follows that $p^r_{2,l} > (\phi - 1)C_{max}(\pi)$.

Due to inequality (8.8) and equations (8.7) and (8.6), we obtain

$$\phi C_{max}(\pi) < p^l_{1,l} + p^l_{2,l} + r_l + p^r_{2,l} + \overline{P}_2 = r_l + p^l_{1,l} + p_{2,l} + \overline{P}_2. \tag{8.9}$$

Since J_l and all jobs that contribute to $\overline{P}_2$ are released at time r_l or later, we have $r_l + p_{2,l} + \overline{P}_2 \leq C_{max}(\pi)$. Together with inequality (8.9), we have

$$\phi C_{max}(\pi) < C_{max}(\pi) + p^l_{1,l}.$$

This implies $p^l_{1,l} > (\phi - 1)C_{max}(\pi)$. This completes the proof. ■

Theorem 15 *For $O2|online, r_j, 1 \leq p_{\cdot,j} \leq \alpha|C_{max}$, algorithm SLICE-$\alpha$ is optimal with a competitive ratio ϕ.*

Proof. By contradiction. We will show that there is a contradiction to Lemma 27 in order to prove that the smallest counterexample $\mathcal{I}$ does not exist. We only need to prove $p^l_{1,l} \leq (\phi - 1)C_{max}(\pi)$ or $p^r_{2,l} \leq (\phi - 1)C_{max}(\pi)$. Note that if $\min\{p_{1,l}, p_{2,l}\} \leq (\phi - 1)(p_{1,l} + p_{2,l})$, then $\min\{p^l_{1,l}, p^r_{2,l}\} \leq (\phi - 1)C_{max}(\pi)$ and the theorem follows. Therefore, we assume that the processing times of both operations are at least $\phi - 1$ of the total processing time of the job, i.e.,

$$\min\{p_{1,l}, p_{2,l}\} > (\phi - 1)(p_{1,l} + p_{2,l}). \tag{8.10}$$

This implies

$$\min\{p_{1,l}, (\phi - 1)(p_{1,l} + p_{2,l})\} = (\phi - 1)(p_{1,l} + p_{2,l}). \tag{8.11}$$

We know that if $p^l_{1,l} \leq (\phi - 1)C_{max}(\pi)$, the theorem follows. So we suppose that $p^l_{1,l} > (\phi - 1)C_{max}(\pi)$. It follows that

$$p^l_{1,l} > (\phi - 1)(p_{1,l} + p_{2,l}). \tag{8.12}$$

By Step 2 and Lemma 25, we know that J_l was first processed on M_1 from time r_l. Due to inequality (8.10) and equation (8.11), we know that $p_{2,l} > (\phi - 1)(p_{1,l}+p_{2,l}) = \min\{p_{1,l}, (\phi-1)(p_{1,l}+p_{2,l})\}$. Considering Step 2 of the algorithm, this inequality means that $O_{2,l}$ is processed sometime during $[r_l, r_{k+1}]$. By Step 2 of the algorithm and equalition (8.11), we know that J_l must be processed on

M_1 at time r_{k+1}. Otherwise, $p_{1,l}^l = (\phi - 1)(p_{1,l} + p_{2,l})$ and this contradicts to inequality (8.12).

By inequality (8.12) and Step 2 of the algorithm, we know that at least a portion $p_{2,l} - (\phi - 1)(p_{1,l} + p_{2,l})$ of $O_{2,l}$ has been processed, i.e., $p_{2,l}^l \geq p_{2,l} - (\phi - 1)(p_{1,l} + p_{2,l})$. Together with equation (8.6), we obtain

$$p_{2,l}^r = p_{2,l} - p_{2,l}^l \leq (\phi - 1)(p_{1,l} + p_{2,l}) \leq (\phi - 1)C_{max}(\pi).$$

This completes the proof. ■

Remark 12 *When $\alpha \to \infty$, we have $\phi = 5/4$ which is the competitive ratio of algorithm SLICE proposed in [108].*

Remark 13 *$\phi = 1$ when $\alpha = 1$.*

Chapter 9

Conclusion

The main topic of this thesis has been the evaluation of online algorithms for machine scheduling problems. We used the concept of competitive ratio to measure the performance of online algorithms. For each minimization problem, we gave a lower bound of competitive ratio to establish a bound below which there exists no online algorithm. In contrast, for the maximization problem, we showed an upper bound of competitive ratio to construct a bound over which there exists no online algorithm. By proving that the competitive ratio of an online algorithm matches the lower bound, we showed that the algorithm is optimal. This method has been commonly used in the research of online algorithm.

In this chapter, we generally summarize what we learned from studying online scheduling problems.

9.1 Approximation ratio and competitive ratio

For offline scheduling problems, the number of jobs are finite. By enumerating all possible job sequences, we can obtain an optimal solution. Since enumeration method requires a lot of time, it is impractical. Polynomial approximation algorithms are developed due to this reason.

Approximation algorithms are algorithms used to find approximate solutions to optimization problems. Approximation algorithms are often associated with NP-hard problems; since it is unlikely that there can ever be efficient polynomial time exact algorithms solving NP-hard problems, one settles for polynomial time suboptimal solutions. Unlike heuristics, which usually only find reasonably good solutions reasonably fast, one wants provable solution quality and provable run time bounds. Ideally, the approximation is optimal up to a small constant factor. Approximation algorithms are increasingly being used for problems where exact polynomial-time algorithms are known but are too expensive due to the input size.

Similarity. The performance of an approximation algorithm is measured by its approximative ratio. Similarly, the performance of an online algorithm is measured by its competitive ratio. The definitions of two ratios are similar. They are all worst-case analysis. Usually, online algorithms are polynomial as approximation algorithms.

Difference. The environment of the problem are different. An online algorithm makes decision without future information, while an approximation algorithm does with all information. The input size of job sequences are different; for an online problem, an job sequence can be infinite.

9.2 Getting a better online algorithm

First, develop some feeling for online algorithms by carefully considering the proof of the lower bound of competitive ratio. An online algorithm tries to avoid the worst-case. Note the threshold values which cause case-by-case discussion in the proof. Some times we can use the lower bound of competitive ratio to design an online algorithm.

Second, design an simple algorithm with an assumption that no new job arrive any more. Such an algorithm makes its decision purely on the basis of all available information, disregarding any possible event in the future.

Third, consider waiting strategy. This means that postponing the decision to avoid the decision too greedy. An greedy algorithm some times causes the performance drastically decreases.

9.3 Drawbacks of online scheduling problems

First, the more complex the machine environment is, the less results online algorithms have. For example, for the problem of online scheduling on two flow shop, there are scare results. Normally, an online algorithm cannot be too complex since the proof of competitive ratio could be dramatically difficult. This reason causes the limited usage of online algorithms.

Second, the input sequence of online problem could be infinite. This yields that one cannot firstly consider a simple case with several jobs in order to attack the difficult case with infinite jobs.

Third, competitive analysis is worst-case analysis. Unlike average-case analysis, it gives a rather pessimistic impression. It often yields unrealistically high competitive ratios for typical instances. An algorithm which does not consider the worst case works well in practice.

9.4 Future works

Although a lot of results have been obtained in recent years, there exist numbers of unexploited areas and some unsolved problems.

As we all know, when the machine environment gets complicated, the results of online algorithms get scare. In a decade, there are less results about online scheduling on open shop, flow shop and job shop. Considering this situation, the following problems are my future works.

Problem 1: Design a better algorithm for online scheduling on two-machine flow shop.

Problem 2: Establish a lower bound and an upper bound for online scheduling on m-machine flow shop and open shop.

Problem 3: Consider batch-machine environment in flow shop and open shop scheduling problems.

Problem 4: Consider availability constraint in two-machine flow shop and open shop scheduling problems.

Problem 5: Consider delivery time in two-machine flow shop and open shop scheduling problems.

Problem 6: Design semi-online algorithm for flow shop and open shop scheduling problems.

Bibliography

[1] S. Albers. Better bounds for online scheduling. In *SIAM Journal on Computing*, pages 130–139, 1997.

[2] E.J. Anderson and C.N. Potts. Online scheduling of a single machine to minimize total weighted completion time. *Mathematics of Operations Research*, 29:686–697, 2004.

[3] E. Angelelli, M.G. Speranza, and Z. Tuza. Semi-online scheduling on two uniform processors. *Theoretical Computer Science*, 393:211–219, 2008.

[4] Y. Azar and L. Epstein. On-line machine covering. *Journal of Scheduling*, 1:67–77, 1998.

[5] Y. Azar and L. Epstein. Online scheduling with precedence constraints. *Discrete Applied Mathematics*, 119:169–180, 2002.

[6] Y. Azar and O. Regev. On-line bin-stretching. *Theoretical Computer Science*, 168:17–41, 2001.

[7] P. Baptiste. An $O(n^4)$ algorithm for preemptive scheduling of a single machine to minimize the number of late jobs. *Operations Research Letters*, 24:175–180, 1999.

[8] P. Baptiste. On minimizing the weighted number of late jobs in unit execution time open-shops. *European Journal of Operational Research*, 149:344–354, 2003.

[9] Y. Bartal, A. Fiat, H. Karloff, and R. Vohra. New algorithms for an ancient scheduling problem. *Journal of Computer and System Sciences*, 51:359–366, 1995.

[10] Y. Bartal, S. Leonardi, A.M. Spaccamela, J. Sgall, and L. Stougie. Multiprocessor scheduling with rejection. *SIAM Journal on Discrete Mathematics*, 13:64–78, 2000.

[11] D.K. Friesen B.L. Deuermeyer and M.A. Langston. Scheduling to maximize the minimum processor finish time in a multiprocessor system. *SIAM Journal on Algebraic and Discrete Methods*, 3:190–196, 1982.

[12] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[13] P. Brucker. *Scheduling algorithms*. Springer, 2006.

[14] P. Brucker, A. Gladky, H. Hoogeveen, M.Y. Kovalyov, C.N. Potts, T. Tautenhahn, and S.L. van de Velde. Scheduling a batching machine. *Journal of Scheduling*, 1:31–54, 1998.

[15] S. Cao and Z. Tan. Online uniform machine covering with the known largest size. *Progress in Natural Science*, 17:1271–1278, 2007.

[16] W. Chan, F.Y.L. Chin, G. Ye, D.and Zhang, and Y. Zhang. On-line scheduling of parallel jobs on two machines. *Journal of Discrete Algorithms*, 6:3–10, 2008.

[17] C. Chekuri, R. Motwani, B. Natarajan, and C. Stein. Approximation techniques for average completion time scheduling. *SIAM Journal on Computing*, 31:146–166, 2001.

[18] B. Chen, D. Du, J. Han, and J. Wen. On-line scheduling of small open shops. *Discrete Applied Mathematics*, 110:133–150, 2001.

[19] B. Chen, A. Van Vliet, and G.J. Woeginger. New lower and upper bounds for online scheduling. *Operation Research Letters*, 16:221–230, 1994.

[20] B. Chen and A.P.A. Vestjens. Scheduling on identical machines: How good is lpt in an on-line setting? *Operations Research Letters*, 21:165–169, 1996.

[21] B. Chen and G.J. Woeginger. A study of on-line scheduling two-stage shops. In *Minimax and Applications*, pages 97–107. Kluwer Academic Publishers, 1995.

[22] T.C.E. Cheng, H. Kellerer, and V. Kotov. Semi-online multiprocessor scheduling with given total processing time. *Theoretical Computer Science*, 337:134–146, 2005.

[23] T.C.E. Cheng, C.T. Ng, and V. Kotov. A new algorithm for online uniform-machine scheduling to minimize the makespan. *Information Processing Letters*, 99:102–105, 2006.

[24] Y. Cho and S. Shani. Bounds for list schedules on uniform processors. *SIAM Jounral of Computing*, 9:91–103, 1980.

[25] J. Csirik, H. Kellerer, and G.J. Woeginger. The exact lpt bound for maximizing the minimum completion time. *Operations Research Letters*, 11:281–287, 1992.

[26] X. Deng, C.K. Poon, and Y. Zhang. Approximation algorithms in batch processing. *Journal of Combinatorial Optimization*, 7:247–257, 2003.

[27] D. Du. Optimal preemptive semi-online scheduling on two uniform processors. *Information Processing Letters*, 92:219–223, 2004.

[28] D. Du and K. Ko. *Theory of Computational Complexity*. John-Wiley, 2000.

[29] L. Epstein. Tight bounds for online bandwidth allocation on two links. *Discrete Applied Mathematics*, 148:181–188, 2005.

[30] L. Epstein, J. Noga, S. Seden, J. Sgall, and G. Woeginger. Randomized on-line scheduling on two uniform machines. *Journal of Scheduling*, 4:71–92, 2001.

[31] L. Epstein, J. Noga, and G.J. Woeginger. Online scheduling of unit time jobs with rejection: minimizing the total completion time. *Operations Research Letters*, 30:415–420, 2002.

[32] L. Epstein and R.V. Stee. Lower bounds for online single-machine scheduling. *Theoretical Computer Science*, 299:439–450, 2003.

[33] U. Faigle, W. Kern, and G. Turan. On the performance of on-line algorithms for partition problems. *Acta Cybernetica*, 9(2):107–119, 1989.

[34] R. Fu, J. Tian, J. Yuan, and C. He. On-line scheduling on a batch machine to minimize makespan with limited restarts. *Operational Research Letters*, 36:255–258, 2008.

[35] R. Fu, J. Tian, J. Yuan, and Y. Lin. On-line scheduling in a parallel batch processing system to minimize makespan using restarts. *Theoretical Computer Science*, 374:196–202, 2007.

[36] G. Galambos and G.J. Woeginger. An on-line scheduling heuristic with better worst case ratio than graham's list scheduling. *SIAM Journal on Computing*, 22(2):349–355, 1993.

[37] M.R. Garey and D.S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[38] T.F. Gonzales and D.B. Johnson. A new algorithm for preemptive scheduling of trees. *Journal of the ACM*, 27:287–312, 1980.

[39] T. Gonzalez and S. Sahni. Open shop scheduling to minimize finish time. *Journal of the ACM*, 23:665–679, 1976.

[40] R.L. Graham. Bounds for certain multiprocessor anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

[41] R.L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17:416–429, 1969.

[42] J.N. Gyorgy and C. Imreh. Online scheduling with machine cost and rejection. *Discrete Applied Mathematics*, 155:2546–2554, 2007.

[43] J.T. Havill and W. Mao. Competitive online scheduling of perfectly malleable jobs with setup times. *European Journal of Operational Research*, 187:1126–1142, 2008.

[44] Y. He. The optimal on-line parallel machine scheduling. *Computers & Mathematics with Applications*, 39:117–121, 2000.

[45] Y. He and G. Dosa. Semi-online scheduling jobs with tightly-grouped processing times on three identical machines. *Discrete Applied Mathematics*, 150:140–159, 2005.

[46] Y. He and G. Zhang. Semi on-line scheduling on two identical machines. *Computing*, 62:179–187, 1999.

[47] Y. Hendel and F. Sourd. Efficient neighborhood search for the one-machine earliness-tardiness scheduling problem. *European Journal of Operational Research*, 173:108–116, 2006.

[48] Y. Hendel and F. Sourd. An improved earliness-tardiness timing algorithm. *Computers & operations research*, 34:2931–2938, 2007.

[49] K.S. Hong and J.Y.T. Leung. Online scheduling of real-time tasks. *IEEE Transactions on Computers*, 41:1326–1331, 1992.

[50] H. Hoogeveen, C.N. Potts, and G.J. Woeginger. On-line scheduling on a single machine: maximizing the number of early jobs. *Operation Research Letters*, 27:193–197, 2000.

[51] J.A. Hoogeveen and A.P.A. Vestjens. Optimal on-line algorithms for single-machine scheduling. In *Proceedings of the Fifth Conference On Integer Programming and Combinatorial Optimization*, pages 404–414. Springer, 1996.

[52] J.A. Hoogeveen and A.P.A. Vestjens. A best possible deterministic on-line algorithm for minimizing maximum delivery time on a single machine. *SIAM Journal on Discrete Mathematics*, 13:56–63, 2000.

[53] Y. Huo, J.Y.T. Leung, and X. Wang. Online scheduling of equal-processing-time task systems. *Theoretical Computer Science*, 401:85–95, 2008.

[54] J.L. Hurink and J.J. Paulus. Online scheduling of parallel jobs on two machines is 2-competitive. *Operations Research Letters*, 36:51–56, 2008.

[55] H. Hwang, S. Chang, and K. Lee. Parallel machines scheduling under a grade of service provision. *Computers and operations research*, 31:2055–2061, 2004.

[56] C. Imreh. Online scheduling with general machine cost functions. *Discrete Applied Mathematics*, 157:2070–2077, 2009.

[57] Y. Jiang. Online scheduling on parallel machines with two gos levels. *Journal of Combinatorial Optimization*, 16:28–38, 2008.

[58] B. Johannes. Scheduling parallel jobs to minimize the makespan. *Journal of Scheduling*, 9:433–452, 2006.

[59] D.R. Karger, S.J. Phillips, and E. Torng. A better algorithm for an ancient scheduling problem. *Journal of Algorithms*, 20:400–430, 1996.

[60] H. Kellerer, V. Kovov, and M.R. Speranza. Semi on-line algorithms for the partition problem. *Operation Research Letters*, 21:235–242, 1997.

[61] H. Kise, T. Iberaki, and H. Mine. Performance analysis of six approximation algorithms for the one-machine maximum lateness scheduling problem with ready times. *Journal of Operation Research Society of Japan*, 22:205–224, 1979.

[62] C. Koulamas and G.J. Kyparisis. A modified lpt algorithm for the two uniform parallel machine makespan minimization problem. *SIAM Journal on Applied Mathematics*, 196:61–68, 2009.

[63] E.L. Lawler. A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs. *Annals of Operations Research*, 26:125–133, 1990.

[64] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys. Sequencing and scheduling: Algorithms and complexity. In *Handbooks in OR & MS*.

[65] C.Y. Lee and R. Uzsoy. Minimizing makespan on a single batch processing machine with dynamic job arrivals. *International Journal of Production Research*, 37:219–236, 1999.

[66] C.Y. Lee, R. Uzsoy, and L.A. Martin-Vega. Efficient algorithms for scheduling semiconductor burn-in operations. *Operations Research*, 40:764–775, 1992.

[67] K. Lee, J.Y.T. Leung, and M.L. Pinedo. Online scheduling on two uniform machines subject to eligibility constraints. to appear in Theoretical Computer Science.

[68] M. Liu, C. Chu, and Y. Xu. Optimal online algorithm for two-machine open shop scheduling with bounded processing times. submitted to Optimization Letters.

[69] M. Liu, C. Chu, Y. Xu, and F. Zheng. Online scheduling on two uniform machines to minimize the makespan. doi:10.1007/s10878-009-9231-z.

[70] M. Liu, C. Chu, Y. Xu, and F. Zheng. An optimal online algorithm for single machine scheduling with bounded delivery times. doi:10.1016/j.ejor.2009.03.028.

[71] M. Liu, Y. Xu, C. Chu, and L. Wang. Online scheduling on two uniform machines to minimize the makespan. *Discrete Mathematics, Algorithms and Applications*, 1:219–226, 2009.

[72] M. Liu, Y. Xu, C. Chu, and F. Zheng. Online scheduling on m uniform machines to minimize total (weighted) completion time. doi: 10.1016/j.tcs.2009.05.023.

[73] M. Liu, Y. Xu, C. Chu, and F. Zheng. Online scheduling parallel machines to maximize the number of early jobs. submitted to Journal of Global Optimization.

[74] M. Liu, Y. Xu, C. Chu, and F. Zheng. Online scheduling on two uniform machines to minimize the makespan. *Theoretical Computer Science*, 410:2099–2109, 2009.

[75] P. Liu and X. Lu. Online scheduling of parallel machines to minimize total completion times. *Computer and Operations Research*, 36:2647–2652, 2009.

[76] P. Liu and X. Lu. Online scheduling of two uniform machines to minimize total completion times. *Journal of Industrial and Management Optimization*, 5:95–102, 2009.

[77] X. Lu, R.A. Sitters, and L. Stougie. A class of online scheduling algorithms to minimize total completion time. *Operations Research Letters*, 31:232–236, 2003.

[78] N. Megow and A.S. Schulz. Online scheduling to minimize average completion time revisited. *Operations Research Letters*, 32:485–490, 2004.

[79] J.M. Moore. An n job, one machine sequencing algorithm for minimizing the number of late jobs. *Management Science*, 15:102–109, 1968.

[80] Q. Nong, J. Yuan, R. Fu, L. Lin, and J. Tian. The single-machine parallel-batching on-line scheduling problem with family jobs to minimize makespan. *International Journal of Production Economics*, 111:435–440, 2008.

[81] Q.Q. Nong, T.C.E. Cheng, and C.T. Ng. An improved online algorithm for scheduling on two unrestrictive parallel batch processing machines. *Operations Research Letters*, 36:584–588, 2008.

[82] J. Park, S.Y. Chang, and K. Lee. Online and semi-online scheduling of two machines under a grade of service provision. *Operations Research Letters*, 34:692–696, 2006.

[83] C. Philips, C. Stein, and J. Wein. Minimizing average completion time in the presence of release dates. *Mathematical Programming*, 82:199–223, 1998.

[84] M. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Prentice Hall Englewood Cliffs, 2001.

[85] C.K. Poon and W. Yu. Online scheduling algorithms for a batch machine with finite capacity. *Journal of Combinatorial Optimization*, 9:167–186, 2005.

[86] C.N. Potts and L.N. Van Wassenhove. Algorithms for scheduling a single machine to minimize the weighted number of late jobs. *Management Science*, 34:843–858, 1988.

[87] K. Pruhs, J. Sgall, and E. Torng. Online scheduling. In *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*.

[88] F. Ridouard, P. Richard, and P. Martineau. Online scheduling on a batch processing machine with unbounded batch size to minimize the makespan. *European Journal of Operational Research*, 189:1327–1342, 2008.

[89] N. Runge and F. Sourd. A new model for the preemptive earliness-tardiness scheduling problem. *Computers & operations research*, 36:2242–2249, 2009.

[90] P. Sanders, N. Sivadasan, and M. Skutella. Online scheduling with bounded migration. In *Proceedings of the 31st International Colloquium on Automata, Languages and Programming*, volume 3142, pages 1111–1122, 2004.

[91] S. Seiden, J. Sgall, and G.J. Woeginger. Semi-online scheduling with decreasing job sizes. *Operations Research Letters*, 27:215–227, 2000.

[92] S.S. Seiden. Preemptive multiprocessor scheduling with rejection. *Theoretical Computer Science*, 262:437–458, 2001.

[93] J. Sgall. On-line scheduling. In *On-line Algorithms: The state of the Art*, volume 1442, pages 196–231, 1998.

[94] D.B. Shmoys, J. Wein, and D.P. Williamson. Scheduling parallel machines on-line. *SIAM Journal on Computing*, 24:1313–1331, 1995.

[95] D.B. Shmoys, J. Wein, and D.P. Williamson. Scheduling parallel machines on-line. *SIAM Journal on Computing*, 24:1313–1331, 1995.

[96] F. Sourd. Earliness-tardiness scheduling with setup considerations. *Computers & operations research*, 32:1849–1865, 2005.

[97] F. Sourd and S. Kedad-Sidhoum. The one-machine problem with earliness and tardiness penalties. *Journal of Scheduling*, 6:533–549, 2003.

[98] R.V. Stee and H.L. Poutre. Minimizing the total completion time online on a single machine, using restarts. *Journal of Algorithms*, 57:95–129, 2005.

[99] Z. Tan and S. Cao. Semi-online machine covering on two uniform machines with known total size. *Computing*, 78:369–378, 2006.

[100] Z. Tan and Y. He. Semi-online problems on two identical machines with combined partial information. *Operations Research Letters*, 30:408–414, 2002.

[101] Z. Tan and Y. He. Semi-online scheduling problems on two identical machines with inexact partial information. *Theoretical Computer Science*, 377:110–125, 2007.

[102] Z. Tan and Y. Wu. Optimal semi-online algorithms for machine covering. *Theoretical Computer Science*, 372:69–80, 2007.

[103] Z. Tan and S. Yu. Online scheduling with reassignment. *Operations Research Letters*, 36:250–254, 2008.

[104] J. Tian, R. Fu, and J. Yuan. Online scheduling with delivery time on a single batch machine. *Theoretical Computer Science*, 374:49–57, 2007.

[105] J. Tian, R. Fu, and J. Yuan. A best on-line algorithm for single machine scheduling with small delivery times. *Theoretical Computer Science*, 393:287–293, 2008.

[106] J. Tian, R. Fu, and J. Yuan. A best online algorithm for scheduling on two parallel batch machines. *Theoretical Computer Science*, 410:2291–2294, 2009.

[107] A.P.A. Vestjens. Scheduling uniform machines on-line requires nondecreasing speed ratios. Technical report, Mathematical Programming, 1994.

[108] A.P.A. Vestjens. *On-line machine scheduling*. PhD thesis, Eindhoven University of Technology, 1997.

[109] J. Wen and D. Du. Preemptive on-line scheduling for two uniform processors. *Operations Research Letters*, 23:113–116, 1998.

[110] G.J. Woeginger. A polynomial time approximation scheme for maximizing the minimum machine completion time. *Operations Research Letters*, 20:149–154, 1997.

[111] D. Ye and G. Zhang. On-line scheduling of parallel jobs. In *Proceedings of the 11th International Colloquium of Structural Information and Communication Complexity (SIROCCO 2004)*, pages 279–290. Springer, 2004.

[112] J. Yuan, S. Li, J. Tian, and R. Fu. A best online algorithm for the single machine parallel-batch scheduling with restricted delivery times. *Journal of Combinatorial Optimization*, 17:206–213, 2009.

[113] G. Zhang, X. Cai, and C. Wong. On-line algorithms for minimizing makespan on batch processing machines. *Naval Research Logistics*, 48:241–258, 2001.

[114] G. Zhang, X. Cai, and C. Wong. Optimal online algorithms for scheduling on parallel batch processing machines. *IIE Transactions*, 35:175–181, 2003.

[115] F. Zheng, F.Y.L. Chin, S.P.Y. Fung, C.K. Poon, and Y. Xu. A tight lower bound for job scheduling with cancellation. *Information Processing Letters*, 97:1–3, 2006.

[116] F. Zheng, Y. Xu, and E. Zhang. How much can lookahead help in online single machine scheduling. *Information Processing Letters*, 106:70–74, 2008.