Combinatorial algorithms for scheduling jobs to minimize server usage time

Ren, Runtian

2018

Ren, R. (2018). Combinatorial algorithms for scheduling jobs to minimize server usage time.
Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/88251>

<https://doi.org/10.32657/10220/46906>

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

SINGAPORE

COMBINATORIAL ALGORITHMS FOR SCHEDULING JOBS TO MINIMIZE SERVER USAGE TIME

REN Runtian

School of Computer Science and Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirement for the degree of
Doctor of Philosophy

2018

Abstract

This thesis is concerned with three combinatorial optimization problems for job scheduling, named the *MinUsageTime Dynamic Bin Packing* problem, the *Energy-Efficient Job Scheduling* problem and the *Flexible Job Scheduling* problem. These problems are motivated by emerging issues arising from cloud computing and energy-efficient computing. A central theme of these problems is to minimize the server usage time for processing jobs. In this thesis, we focus on the algorithmic aspects of each problem by proposing online and approximation algorithms in the online and offline settings respectively.

The MinUsageTime Dynamic Bin Packing problem aims at packing a set of items arriving and departing over time to minimize the accumulated bin usage time. We study the problem in three different settings. In the offline setting, the information of all the items to pack is assumed known, whereas in the online setting, the items must be placed into bins as they arrive without any knowledge of future item arrivals. The online setting can be further divided into non-clairvoyant and clairvoyant cases. In the non-clairvoyant case, the departure time of each item is not known at the time of its arrival and cannot be used for packing purposes. In the clairvoyant case, the departure time of each item is known for packing purposes. In this thesis, we first show that the First Fit packing algorithm achieves a competitive ratio of $\mu + 3$ in the non-clairvoyant online setting, where μ is the max/min item duration ratio. This competitive ratio closely matches a known lower bound μ on the competitiveness of any deterministic online algorithm and shows that First Fit packing is near optimal. In the clairvoyant online setting, we establish a lower bound of $\Omega(\sqrt{\frac{\log \mu}{\log \log \mu}})$ on the competitive ratio of any deterministic online algorithm. We also propose a classify-by-duration strategy, which can be applied in First Fit packing to achieve a competitive ratio of $O(\log \mu)$. In the offline setting, we propose two $O(1)$ -approximation algorithms, including a 5-approximation Duration Descending First Fit algorithm and a 4-approximation Dual Coloring algorithm.

The Energy-Efficient Job Scheduling problem aims at scheduling a set of jobs to minimize the energy consumption of the machines used. We assume the following energy model. When a machine is “on”, its power usage rate is given by a base rate representing the power consumed by an idle machine plus a variable rate that is proportional to the machine’s instantaneous load. When a machine is “off”, it does not consume energy. State transitions between “on” and “off” incur energy overheads. We show that this Energy-Efficient Job Scheduling problem is a generalization of the MinUsageTime Dynamic Bin Packing problem. In a special case where each job has a unit size equal to the machine capacity, we propose an optimal offline algorithm and an optimal 2-competitive online algorithm. For the general case where jobs can have arbitrary sizes, we first establish a non-trivial lower bound on the optimal solution. Based on this lower bound, we develop a 5-approximation algorithm in the offline setting. In the non-clairvoyant online setting, we design a Modified First Fit algorithm which is $O(\mu)$ -competitive, where μ is the max/min job processing length ratio. We show that the Modified First Fit strategy can be adopted to further construct an $O(\sqrt{\log \mu})$ -competitive algorithm in the clairvoyant online setting which is asymptotically tight.

The Flexible Job Scheduling problem aims at determining the starting times of flexible jobs that do not have to be started immediately at their arrivals to minimize the span of all the jobs, where the span is the time duration in which at least one job is running. In the non-clairvoyant online setting, we first show that no deterministic online scheduler can achieve a competitive ratio less than μ . Then, we propose two $O(\mu)$ -competitive schedulers: Batch and Batch+. The Batch+ scheduler is proved to have a tight competitive ratio of $\mu + 1$. In the clairvoyant online setting, we establish a lower bound of 2 on the competitive ratio of any deterministic online scheduler, and propose two $O(1)$ -competitive schedulers: Classify-by-Duration Batch+ and Profit. The Profit scheduler can achieve a competitive ratio of $4 + 2\sqrt{2}$. These results can be used to extend the MinUsageTime Dynamic Bin Packing problem to model jobs that have laxity in starting.

Acknowledgments

I want to express my gratitude to my supervisor **Prof. Tang Xueyan** for guiding me on my research and mentoring me on my critical thinking. Without his guidance, the achievements of my research work are impossible. I also want to thank my colleagues and friends for their encouragements during the past four years. Working and living with them makes my past four years lively and interesting. Finally, I want to dedicate the thesis to my parents. They always support me without condition no matter when and where I am, which makes me fearless on facing any challenge.

Contents

Abstract	i
Acknowledgments	iii
List of Figures	vii
1. Introduction	1
1.1 Applications	3
1.2 Approximation Algorithms and Online Algorithms	5
1.3 Thesis Contributions	7
2. Literature Review	10
2.1 The Basic Interval Scheduling Problem	10
2.1.1 Offline Setting	11
2.1.2 Online Setting	12
2.2 Interval Scheduling on Non-Identical Machines	14
2.3 Dynamic Storage Allocation Problem and Dynamic Bin Packing Problem	16
2.4 Bandwidth Allocation Problem and Storage Allocation Problem	18
2.5 Machine Minimization Problem and Real-Time Scheduling Problem	19
3. The MinUsageTime Dynamic Bin Packing Problem	22
3.1 Preliminaries	23
3.2 Non-Clairvoyant Online Setting	25
3.2.1 The Any Fit Family of Packing Algorithms	25
3.2.2 The First Fit Packing Algorithm	27
3.3 Clairvoyant Online Setting	34

3.3.1	A Lower Bound on Competitiveness of Any Deterministic Online Packing Algorithm	34
3.3.2	A Classify-by-Duration Strategy	39
3.4	Offline Setting	45
3.4.1	The Duration Descending First Fit Packing Algorithm	45
3.4.2	The Dual Coloring Algorithm	49
3.5	Summary	59
4.	The Energy-Efficient Job Scheduling Problem	61
4.1	Preliminaries	62
4.1.1	Notations and Definitions	62
4.1.2	A Simplification of the Target of the Problem	64
4.2	Unit Job Size Case	64
4.3	Arbitrary Job Size Case	68
4.3.1	A Lower Bound on Optimal Solution	68
4.3.2	Offline Setting	71
4.3.3	Non-Clairvoyant Online Setting	76
4.3.4	Clairvoyant Online Setting	86
4.4	Summary	87
5.	The Flexible Job Scheduling Problem	88
5.1	Preliminaries	89
5.2	Non-Clairvoyant Online Setting	90
5.2.1	A Lower Bound on Competitiveness of Any Deterministic Online Algorithm	90
5.2.2	Batch Scheduler and Batch+ Scheduler	95
5.2.3	A Uniform Laxity Case	99
5.3	Clairvoyant Online Setting	105
5.3.1	A Lower Bound on Competitiveness of Any Deterministic Online Algorithm	105
5.3.2	A Classify-by-Duration Strategy	107
5.3.3	Profit Scheduler	113

5.4	The Generalized MinUsageTime DBP Problem	125
5.5	Summary	127
6.	Conclusion and Future Work	128
6.1	Conclusion	128
6.2	Future Directions	128
	Publications	131
	References	132

List of Figures

2.1	Relationships between the optimization problems discussed in the chapter	11
3.1	Span of an item list	24
3.2	An instance for establishing the lower bound on the competitiveness of the Any Fit family of packing algorithms	26
3.3	An example of usage periods	28
3.4	Splitting V_k for bin B_k where $\mathcal{R}'_k = \{r_{k,1}, r_{k,2}, r_{k,3}, r_{k,4}, r_{k,5}, r_{k,6}, r_{k,7}\}$. . .	30
3.5	An example of supplier bins	31
3.6	An example of $P^{-1}(\hat{r}) = \{r_1, r_2, r_3, r_4, r_5, r_6\}$	32
3.7	An example of the first four iterations in the item releasing process when $k = 3$	36
3.8	First instance of Classify-by-Duration First Fit packing	42
3.9	Second instance of Classify-by-Duration First Fit packing	43
3.10	An example of splitting $\bigcup_{r \in \mathcal{R}'_k} I(r)$ for the items placed in bin B_k where $\mathcal{R}'_k = \{r_1, r_2, r_3, r_4, r_5, r_6, r_7\}$	47
3.11	An example of $P^{-1}(\hat{r}) = \{r_1, r_2, r_3, r_4, r_5\}$	48
3.12	Item placement in the demand chart	50
3.13	Item placement in the demand chart	53
3.14	Item overlapping in the demand chart	56
3.15	Item packing based on placement in the demand chart	57
4.1	An example of EEJS-Unit	66
4.2	An optimal REEJS schedule	69
4.3	An example for scheduling small jobs in the offline setting	74
4.4	An example for scheduling small jobs by Modified First Fit	77

5.1	Jobs released in iterations i and $i + 1$	92
5.2	An example schedule induced by the Batch scheduler	97
5.3	An example schedule induced by the Batch+ scheduler	99
5.4	Definition of I_j	111
5.5	Constructing a graph for flag jobs	117
5.6	An optimal schedule on tree T	121
6.1	A summary of the competitiveness and approximation bounds	129

Chapter 1

Introduction

This thesis is concerned with the topic of job scheduling to minimize server usage time. The topic is motivated by issues arising from cloud computing and energy-efficient computing, both of which become popular and are highly concerned in recent years. Specifically, we study three combinatorial optimization problems in this thesis: the *MinUsage-Time Dynamic Bin Packing (DBP)* problem, the *Energy-Efficient Job Scheduling* problem and the *Flexible Job Scheduling* problem. We focus on the algorithmic aspects of the three problems and consider the online and offline versions of each problem separately. In the online setting, the inputs are revealed incrementally and the algorithms need to make decisions without knowing the rest of the inputs. The performance of an online algorithm is often characterized by its *competitive ratio*, i.e., the worst-case ratio between the objective function values of an online solution and an optimal offline solution [20]. We propose online algorithms and conduct the *competitive analysis* for the online algorithms. In the offline setting, the complete problem instance is given and the algorithms need to produce solutions which meet or optimize the given criteria. Our optimization problems are mostly NP-hard and hence it is difficult to compute an optimal solution for these problems in polynomial time. Thus, we focus on proposing *approximation algorithms* for the offline version of the problems, which run in polynomial time and are guaranteed to produce solutions not too far away from the optimal solution. The performance of an approximation algorithm is often characterized by its *approximation ratio*, i.e., the worst-case ratio between the objective function values of an approximate solution and an optimal solution [74]. In this thesis, we study the aforementioned three problems

by developing online and approximation algorithms and carry out the competitive and approximation analysis in the online and offline settings respectively.

The first part of the thesis is concerned with the MinUsageTime DBP problem defined as follows. We are given a set of items, each specified by a size, an arrival time and a departure time. Each item is to be placed in a bin upon its arrival and stay in the bin till its departure. The total size of the items placed in any bin cannot exceed the bin's capacity at any time. A bin is opened when it receives the first item and is closed when all the items in the bin depart. The usage time of a bin is the duration from its opening to its closing. The target is to pack all the items into bins with the total bin usage time minimized. The MinUsageTime DBP problem models a server acquisition and job scheduling problem in cloud computing for monetary cost optimization under “pay-as-you-go” billing [1]. We study the MinUsageTime DBP problem in Chapter 3.

The second part of the thesis is concerned with the Energy-Efficient Job Scheduling problem, which aims at scheduling a set of jobs to minimize the energy consumption of the machines used. We focus on interval jobs, each specified by a size, an arrival time and a processing length. Once a job arrives, it has to be placed on a machine immediately to run for a period of its processing length without interruption. Each machine has a fixed capacity such that the total size of the jobs running on it cannot exceed its capacity at any time. When a machine is “on”, its power usage rate is given by a base rate representing the power consumed by an idle machine plus a variable rate that is proportional to the machine's instantaneous load. When a machine is “off”, it does not consume energy. State transitions between “on” and “off” incur energy overheads. As shall be shown later, the Energy-Efficient Job Scheduling problem is a generalization of the MinUsageTime DBP problem. We study the Energy-Efficient Job Scheduling problem in Chapter 4.

The third part of the thesis is concerned with the Flexible Job Scheduling problem. The input of this problem is a set of jobs, each having an arrival time, a starting deadline and a processing length. Each job has to be started by the scheduler between its arrival and its starting deadline. Once started, the job runs for a period of the processing length without interruption. The target is to minimize the span of all the jobs, i.e., the time duration in which at least one job is running. The Flexible Job Scheduling problem can be used to extend the MinUsageTime DBP problem to model jobs that have laxity in starting. We study the Flexible Job Scheduling problem in Chapter 5.

In the rest of this chapter, we first present some application scenarios of the above problems in Section 1.1. Next, in Section 1.2, we introduce some basics of approximation algorithms and online algorithms. Finally, in Section 1.3, we give an outline and the main results of the thesis.

1.1 Applications

Cloud Computing Cloud-based systems often face the problem of dispatching jobs to run on cloud servers. In a typical scenario, each cloud server has a fixed capacity of computational resources. Each job requires a certain amount of computational resources to run on a cloud server from its arrival time to its departure time. Since the total resource demand of the jobs running on a server cannot exceed its capacity, a limited number of jobs can be processed concurrently on each cloud server. Thus, a sufficient number of cloud servers need to be acquired to handle a given set of jobs. On the other hand, a cloud server rented from public clouds such as Amazon EC2 is normally charged according to its running hours by “pay-as-you-go” billing [1]. Therefore, to minimize the total renting cost, it is equivalent to minimize the total running hours of the cloud servers. Such a server acquisition and job dispatching problem can be modeled exactly by the MinUsageTime DBP problem, where the jobs and cloud servers correspond to the items and bins respectively.

A typical application of the above job dispatching problem is cloud gaming [67]. Cloud gaming is a promising application of the rapidly expanding cloud computing infrastructure and has attracted a large amount of interests among entrepreneurs and researchers. In a cloud gaming system, games are run and rendered on cloud servers. Players interact with the games via networked thin clients [41, 54]. Running each game instance demands a certain amount of GPU resources. When a playing request is received, it should be immediately assigned to a cloud server that has enough GPU resources to run the requested game instance. Several game instances can share the same cloud server provided that the server’s GPU resources are not saturated. Each game instance keeps running on the assigned server until the player stops the game. Migrating a game instance from one server to another during its execution is usually not allowed due to interruption to game

play. To serve numerous playing requests in an online manner, cloud gaming providers such as GaiKai [2] and OnLive [3] rent a large number of servers from public clouds to run game instances [71]. Then, a natural **issue** faced by the cloud gaming provider is how to dispatch the playing requests to cloud servers in an online manner to minimize the total renting cost of the servers used.

Here we use an example to illustrate the above issue. Suppose the renting cost rate of cloud servers is 1 and each cloud server can serve at most 10 requests concurrently. Since the gaming provider generally does not know when a player will quit the game, the ending time of a request is not known at its arrival and thus cannot be used for its dispatching purpose. Suppose 1000 requests arrive at the same time 0. An intuitive greedy strategy is to rent exactly 100 cloud servers for serving these requests. For every 10 requests dispatched to the same cloud server, suppose 9 players quit the game at time 1 and only one player quits the game at time 100. Then, each cloud server has a usage time of 100, so the total renting cost is 10000. However, an optimal dispatching strategy is to use one server to run all the requests with the ending time 100 and use 99 servers to run the remaining requests. In this way, the total renting cost is only 199. We can see that there is a huge gap between the actual cost and the minimum cost.

The following questions arise naturally based on the above example:

1. When dispatching jobs in an online manner with the ending time of each job not known at its arrival, does there exist a job dispatching strategy that can guarantee to produce the minimum server usage time or a usage time bounded by a constant times the minimum server usage time?
2. If the above strategy does not exist, what would be the performance bound of an online job dispatching strategy?
3. How can a good job dispatching strategy be designed to optimize the performance of a job dispatching strategy?
4. In the case that jobs need to be dispatched in an online manner with the ending time of each job known at its arrival, what are the answers to the above questions?

We shall answer these questions in Chapter 3.

Energy-Efficient Computing Energy management in computing servers and data centers has become a key issue faced by industries. It is observed that up to 50% budget of a data center is invested on electricity costs, and about 1.5% of the total electricity worldwide is used by data centers nowadays [73]. Moreover, it is seen that the servers of a data center are typically utilized only 20-40% of the time [7, 16]. Barroso and Hölzle [16] showed that the power usage rate of an “on” server follows a linear model that includes a base component representing the power consumed by an idle server and a variable component proportional to the server’s load. The base component can be as high as the peak of the variable component. Thus, keeping servers “on” when they do not have work to do can lead to substantial waste of energy. A widely used strategy for energy conservation is to switch the servers off (or convert them to sleep states) when they are not in use. While powering idle servers down can save energy, the servers will have to be powered up again when they are needed for processing new jobs. State transitions themselves also incur non-negligible energy overheads. Therefore, it is important to schedule jobs in the servers and manage the server states in an effective way to minimize the energy consumption of the servers. Motivated by such issues, we study the Energy-Efficient Job Scheduling problem in this thesis.

1.2 Approximation Algorithms and Online Algorithms

We begin with the concept of an **optimization problem**, which can be either a minimization problem or a maximization problem. A minimization problem is defined as follows. Given an input $\mathcal{I}$ and a cost function c , a set of feasible solutions $F(\mathcal{I})$ is associated with the input $\mathcal{I}$ and each feasible solution $O \in F(\mathcal{I})$ is associated with a cost $c(\mathcal{I}, O)$. The minimization problem is concerned with finding a feasible solution with the minimum cost [20]. Similarly, given any input $\mathcal{I}$ of a maximization problem, each feasible solution $O \in F(\mathcal{I})$ is associated with a weight $w(\mathcal{I}, O)$. The target of the maximization problem is to find a feasible solution with the maximum weight.

Algorithms for solving an optimization problem are general, precise, step-by-step procedures which produce feasible solutions given any instance. Specifically for a minimization problem, given any input $\mathcal{I}$, an algorithm ALG computes a feasible solution

$ALG[\mathcal{I}] \in F(\mathcal{I})$ associated with its cost $ALG(\mathcal{I}) = c(\mathcal{I}, ALG[\mathcal{I}])$. An optimal algorithm OPT is the algorithm which satisfies for any input $\mathcal{I}$,

$$OPT(\mathcal{I}) = \min_{O \in F(\mathcal{I})} c(\mathcal{I}, O).$$

It is well known that for many optimization problems, an optimal algorithm may be extremely time-consuming due to the computational complexity. The class of *NP-complete problems* represents a large collection of such problems, which are all related in the sense that the existence of a polynomial-time optimal algorithm for any one of them implies the polynomial-time solvability for the whole class [37]. So far, no polynomial-time optimal algorithm for any NP-complete problem is known. As an alternative, researchers relax the requirement of finding an optimal solution for these problems and focus on proposing **approximation algorithms**, which run in polynomial time and are guaranteed to produce a cost not too far away from the optimal solution [74, 75]. Formally, a ρ -approximation algorithm ALG for a minimization problem satisfies

$$ALG(\mathcal{I}) \leq \rho \cdot OPT(\mathcal{I})$$

for any instance $\mathcal{I}$. The parameter ρ is called the **approximation ratio** or **approximation factor** of ALG . Similarly, a $1/\rho$ -approximation algorithm for a maximization problem satisfies

$$ALG(\mathcal{I}) \geq \frac{1}{\rho} \cdot OPT(\mathcal{I})$$

for any instance $\mathcal{I}$.

Besides the offline setting in which the complete input is given, an optimization problem can also be studied in the online setting. In the online setting, the input is revealed incrementally. An algorithm must make a sequence of decisions based on partial input without the knowledge of the rest of the input. Each decision made has an impact on the final quality of the algorithm's output. Such algorithms are called **online algorithms**. A **deterministic online algorithm** is an algorithm which, given a particular input, always makes the same decision. A **randomized online algorithm**, on the other hand, employs a degree of randomness in its logic and can make different decisions based on the same partial input. In this thesis, we focus on deterministic online algorithms.

A standard approach to study the performance of an online algorithm is called **competitive analysis** [20]. Formally, for an online minimization problem, a ρ -competitive online algorithm ALG satisfies

$$ALG(\mathcal{I}) \leq \rho \cdot OPT(\mathcal{I})$$

for any instance $\mathcal{I}$. The parameter ρ is known as the **competitive ratio** of the online algorithm ALG . A ρ -competitive online algorithm for an online maximization problem can be defined similarly.

1.3 Thesis Contributions

We now give an outline of the thesis and its main results.

In Chapter 3, we study the MinUsageTime DBP problem in the non-clairvoyant and clairvoyant online settings as well as the offline setting:

1. In the non-clairvoyant online setting, the departure time of an item is not known when it arrives and thus cannot be used for packing purposes. We show that no Any Fit packing algorithm can achieve a competitive ratio less than $\frac{4\mu+3}{\mu+2}$ when $\mu < \frac{1+\sqrt{5}}{2}$ and less than $\mu + 1$ when $\mu \geq \frac{1+\sqrt{5}}{2}$. Here, μ represents the max/min item duration ratio among all the items to pack. We also show that the First Fit packing algorithm, a particular Any Fit packing algorithm, achieves a competitive ratio of $\mu + 3$ in this setting.
2. In the clairvoyant online setting, the departure time of an item is known when it arrives and can be used for packing purposes. We establish a lower bound of $\Omega\left(\sqrt{\frac{\log \mu}{\log \log \mu}}\right)$ on the competitive ratio of any deterministic online algorithm. We also introduce a classify-by-duration strategy which can be applied in First Fit packing to achieve a competitive ratio of $O(\log \mu)$ in this setting.
3. In the offline setting, the information of all the items to pack is assumed known before the packing process. We propose two $O(1)$ -approximation algorithms in this setting, including a 5-approximation Duration Descending First Fit algorithm and a 4-approximation Dual Coloring algorithm.

In Chapter 4, we study the Energy-Efficient Job Scheduling problem:

1. We first consider a special case of the problem, where each job has a unit size equal to the machine capacity so that each machine can run at most one job at any time. We propose an optimal algorithm in the offline setting and a 2-competitive online algorithm in the online setting. An instance is constructed to show that no deterministic online algorithm can achieve a competitive ratio less than 2 and thus prove that our online algorithm is optimal.
2. Next, we consider the general case of the problem, where jobs can have arbitrary sizes such that each machine can run multiple jobs concurrently provided that the total size of these jobs does not exceed the machine capacity. We show that the general case is an extension of the MinUsageTime DBP problem by including the energy overheads for powering machines up and down. We study the general case in the offline, non-clairvoyant online and clairvoyant online settings respectively. In the offline setting, we propose a 5-approximation algorithm based on the Dual Coloring algorithm introduced in Chapter 3. In the non-clairvoyant online setting, we propose a Modified First Fit algorithm which is $O(\mu)$ -competitive for the general case (where μ is the max/min job processing length ratio) and is $O(1)$ -competitive when each job has a processing length for which the base energy consumption of a machine is no larger than the energy cost to power up a machine. We finally show that the Modified First Fit algorithm can be adopted to further construct an asymptotically tight $O(\sqrt{\log \mu})$ -competitive algorithm in the clairvoyant online setting.

In Chapter 5, we study the Flexible Job Scheduling problem:

1. We first study the problem in the non-clairvoyant online setting. We show that no deterministic online scheduler (algorithm) can achieve a competitive ratio less than μ , where μ is max/min job processing length ratio. We propose two $O(\mu)$ -competitive schedulers: Batch and Batch+. The Batch scheduler achieves a competitive ratio between 2μ and $2\mu + 1$, and the Batch+ scheduler achieves a tight competitive ratio of $\mu + 1$. We also consider a special case in which all the jobs have

the same laxity. We show that in this special case, no deterministic online scheduler can achieve a competitive ratio less than 2, and the Batch scheduler achieves a tight competitive ratio of 3.

2. Next, we study the problem in the clairvoyant online setting. We establish a lower bound of 2 on the competitive ratio of any deterministic online scheduler. We propose two $O(1)$ -competitive schedulers: Classify-by-Duration Batch+ and Profit. We show that the Classify-by-Duration Batch+ scheduler achieves a competitive ratio of $7 + 2\sqrt{6}$, and the Profit scheduler achieves a competitive ratio of $4 + 2\sqrt{2}$.
3. Finally, we show that the algorithmic results of the Flexible Job Scheduling problem can be applied to generalize the MinUsageTime DBP problem to model flexible jobs that have laxity in starting. For the generalized MinUsageTime DBP problem, we propose an $O(\mu)$ -competitive algorithm in the non-clairvoyant online setting and an $O(\log \mu)$ -competitive algorithm in the clairvoyant online setting.

Besides the above three chapters, Chapter 2 of the thesis reviews the related work, and Chapter 6 concludes the thesis and outlines some future research directions.

Chapter 2

Literature Review

In this chapter, we introduce the previous work related to the problems studied in this thesis, most of which focus on the topic of interval scheduling. The most defining characteristic of the interval scheduling problems is that each job must be started immediately at its arrival and run continuously without interruption until it departs. So, each job can be interpreted as an interval. Such a simple model is widely adopted by a large number of problems in operations research and computer science, like flight/vehicle scheduling, transport assignment between terminals, bandwidth allocation of communication channels and so on. In this thesis, the MinUsageTime DBP problem and the Energy-Efficient Job Scheduling problem also adopt this interval job model, while the Flexible Job Scheduling problem considers a variant of the interval job model. Thus, we review several classical optimization problems of interval scheduling, and summarize the algorithmic results for each problem. Figure 2.1 illustrates all the problems discussed in this chapter.

2.1 The Basic Interval Scheduling Problem

We start by introducing the basic interval scheduling problem defined as follows: we are given n interval jobs, where each job J has an interval $I(J) = [I(J)^-, I(J)^+)$ for its execution. A machine can run at most one job at any time, so that any two overlapping interval jobs (i.e., J and J' with $I(J) \cap I(J') \neq \emptyset$) cannot be scheduled on the same machine. Job execution is not allowed to be interrupted, i.e., if a job J is placed on a machine, J has to run on this machine throughout the time interval $I(J)$. There are

Figure 2.1: Relationships between the optimization problems discussed in the chapter

two versions of the basic interval scheduling problem: a cost minimization version and a revenue maximization version. In the cost minimization problem, all the jobs are to be scheduled on machines and the target is to minimize the number of machines used for processing all the jobs. In the revenue maximization problem, the number of machines available is limited. Each job J is associated with a weight $w(J)$. The target is to choose a subset of jobs to be scheduled on the given machines with their total weight maximized. We introduce the algorithmic results for these two problems in this section.

2.1.1 Offline Setting

The cost minimization problem is rather easy to solve: a simple greedy strategy is enough to compute an optimal solution. Suppose all the jobs are sorted in the order of their arrivals. The greedy strategy schedules jobs one after another. The first job, of course, is scheduled onto a new machine. To schedule any subsequent job J , if there exists at least one available machine (i.e., no job is running on it at time $I(J)^-$) ever used before, the algorithm arbitrarily chooses one such machine for processing J . Otherwise, it starts a new machine for processing J . By induction it is easy to prove the optimality. This greedy strategy can be applied in an online setting.

The revenue maximization problem is not so easy. In a special case where only one machine is available and the jobs are unweighted (i.e., each job is associated with the same weight and thus the target is equivalent to maximizing the number of jobs chosen), there

exists an optimal algorithm based on a last-fit greedy strategy. All the jobs are sorted in the order of their departures and scheduled one by one. The first job is directly chosen and placed onto the given machine. Any subsequent job is chosen and placed onto the machine if it does not overlap with any job already placed. The optimality of this strategy can also be verified by induction. When only one machine is available and the jobs are weighted, selecting a subset of jobs to maximize the total weight is equivalent to finding the maximum-weight independent set in an interval graph. A dynamic programming strategy running in $O(n^2)$ time can be applied to find the optimal solution.

For the general case of the revenue maximization problem, Arkin and Silverberg [6] formulated it as a binary linear program and showed that the constraint matrix is totally unimodular, thereby proving that the problem is polynomially solvable. Based on a minimum cost flow transformation, they proposed a $O(n^2 \log n)$ -time algorithm given any number of machines available. The main idea is to delete a subset of low-weight jobs such that all remaining cliques in the interval graph have sizes bounded by the number of machines. Here, a clique in the interval graph represents a set of jobs that share at least one common time point (i.e., there exists a time point t such that every job's interval contains t). Bouzina and Emmons [21] developed a more efficient $O(mn \log n)$ -time algorithm, since the number of machines available m is smaller than the number of the jobs n in any non-trivial problem instances.

2.1.2 Online Setting

Now, we introduce the online version of the revenue maximization problem with one machine available. In the online setting, whether to accept a job has to be decided immediately when it arrives without knowing future job arrivals. Once accepted, no other job can be accepted before this job departs. Obviously, no deterministic online algorithm can achieve a bounded competitive ratio. Even for the unweighted case, any deterministic online strategy can hardly be interesting: on one hand, no deterministic online algorithm can achieve a competitive ratio less than μ , where μ is the max/min job length ratio; on the other hand, a simple greedy strategy which always accepts a job if the machine is available at its arrival already yields a competitive ratio of $\lceil \mu \rceil$. Existing work has focused on developing randomized online algorithms. In a special case where

the weight of each job is proportional to its length, Lipton and Tomkins [57] proposed a 2-competitive online algorithm when each job has a length of either 1 or k ($k > 1$) and showed that no online algorithm can achieve a competitive ratio better than 2 in this case. Based on this 2-competitive algorithm, they further proposed a $O(\log \mu)^{1+\varepsilon}$ -competitive online algorithm when jobs can have arbitrary lengths, where ε is a positive value that can be arbitrarily small. They also showed that no $O(\log \mu)$ -competitive online algorithm exists, which suggests that the $O(\log \mu)^{1+\varepsilon}$ -competitive online algorithm is near optimal.

Some work studied the revenue maximization problem with abortion. To accept an incoming job, the abortion technique can terminate the execution of an accepted job and discard its weight. Such a technique is useful when a more profitable job arrives and there is already an accepted job running on the machine. The target of the online problem becomes to maximize the accepted jobs which finish their executions without interruption. For the unweighted case, Faigle and Nawijn [33] proposed an optimal algorithm based on a last-fit greedy strategy: accept a job J if the machine is available when it arrives. If a job J' is running on the machine at J 's arrival and $I(J)^+ < I(J')^+$, J' is aborted for accepting J . This idea can also be extended to handle the problem with multiple machines available. In a special case where each job's weight is exactly its length, Baruah *et al.* [17] proposed a 4-competitive deterministic online algorithm. They also showed that no deterministic online algorithm can achieve a competitive ratio less than 4, which suggests the optimality of their online algorithm. Seiden [69] improved the competitiveness by proposing a $(2 + \sqrt{3})$ -competitive randomized online algorithm.

Another version of the online revenue maximization problem is to consider each job as a connection request between two nodes along a line network $P(V, E)$. This can be used to model an online virtual circuit routing problem arising in communication networks and is also known as the online call control problem. Here, the arriving sequence of the requests can be unrelated to the left endpoints of the intervals. Once a request J is accepted, any future request using any edge $e \in I(J)$ must be dropped. Note that the line network is given in advance and has a fixed length in the online call control problem, which is different from the earlier version where the timeline can be extended to infinity.

For the unweighted case of the online call control problem, Awerbuch *et al.* [9] established a lower bound of $\Omega(|V|)$ on the competitive ratio of any deterministic online

algorithm. They also proposed a $O(\log |V|)$ -competitive randomized online algorithm based on a “classify and randomly select” strategy. Finally, they established a lower bound of $\Omega(\log |V|)$ on the competitive ratio of any online algorithm, which implies the optimality of their randomized online algorithm.

There are also several studies on the online call control problem with abortion. For the unweighted case, Garay *et al.* [36] proposed an asymptotically tight $O(\log |V|)$ -competitive deterministic online algorithm. In a special case where each request’s weight is given by its length, they proposed an optimal $(2 + \sqrt{5})$ -competitive deterministic online algorithm.

2.2 Interval Scheduling on Non-Identical Machines

In this section, we introduce a variant of the basic interval scheduling problem such that the machines used for processing jobs are non-identical. This variant also has a cost minimization version and a revenue maximization version. In the cost minimization problem, we are given m machine types, such that using a machine of type i ($1 \leq i \leq m$) costs c_i . Each job J can only be processed by a subset of machine types. The target is to minimize the total cost of the machines used to process a given set of jobs. In the revenue maximization problem, we are directly given m machines. Each job is specified by a weight and a subset of these machines on which it can be processed. The target is to select jobs to be processed by the given machines with their total weight maximized.

The cost minimization problem is equivalent to the classical set cover problem [19]. A simple reduction is as follows. Suppose all the jobs are non-overlapping with each other, so that at most one machine of each type is needed. Let the set elements be the given n jobs. Let the collection of subsets be the m machine types, with each subset i containing all the jobs that can be processed by machine type i and having a cost c_i . In this way, minimizing the total cost for processing all the jobs is exactly finding a minimum-cost set cover from the collection of subsets. It is well known that the set cover problem admits logarithmic approximability [34]. Bhatia *et al.* [19] studied two special cases of the cost minimization problem, which have applications in bandwidth trading and wavelength assignment in optical line systems. One case is called cost minimization with machine time intervals, and the other is called cost minimization with machine strengths.

In cost minimization with machine time intervals, machine types are defined by time intervals. Each machine type i is associated with an available time interval Q_i and can only accommodate the jobs with their intervals fully contained in Q_i . Bhatia *et al.* [19] focused on designing approximation algorithms for this problem. Based on the linear programming relaxation and the primal-dual technique, they proposed a 3-approximation algorithm for the weighted case where machine costs are arbitrary and a 2-approximation algorithm for the unweighted case where all machine types have the same cost. They also showed how to extend their algorithms to a prize collection version, where it is not necessary to schedule all the jobs but there is a penalty to pay if a job is discarded.

In cost minimization with machine strengths, there is a linear order of strengths defined on the machine types, such that a job that can be processed by a machine type can also be processed by a stronger machine type. Of course, the higher strength a machine has, the higher cost it has. Bhatia *et al.* [19] proposed a 2-approximation algorithm for this problem based on some simple combinatorial lower bounds.

Finally, we introduce the revenue maximization problem. Bouzina and Emmons [21] showed that this problem is NP-hard through a reduction from the 3-SAT problem. In the 3-SAT problem, we are given m boolean variables, $v_1, \dots, v_m$ and r clauses $C_1, \dots, C_r$, with each clause consisting of three literals. A literal is either a variable or its negation. The target is to find a truth assignment to each variable such that each clause contains at least one true literal. In the reduction, a job $A_j = [j - 1, j)$ is defined for each clause C_j , and a job $B_i = [0, r)$ is defined for each variable v_i . Suppose two machines M_i and $\bar{M}_i$ can be used to process B_i , and machine M_i ($\bar{M}_i$) can process A_j if v_i ($\bar{v}_i$) is a literal in clause C_j . In this way, a feasible schedule exists for all jobs if and only if there is a truth assignment for the 3-SAT instance. Bar-Noy *et al.* [15] implied a 2-approximation algorithm for the revenue maximization problem. Bhatia *et al.* [19] improved the result by proposing a $\frac{e}{e-1}$ -approximation algorithm.

All the work discussed in Section 2.1 and 2.2 assumes that each machine can process only one job at a time. In contrast, in this thesis, we consider machines that can process multiple jobs concurrently. Moreover, instead of assuming a static and fixed cost for using each machine, we focus on the usage time of each machine which is determined by the jobs assigned to it. Our target is to minimize the total usage time of the machines used.

2.3 Dynamic Storage Allocation Problem and Dynamic Bin Packing Problem

In this section, we introduce the Dynamic Storage Allocation problem [37] and the Dynamic Bin Packing problem [31], both of which generalize the cost minimization version of the basic interval scheduling problem.

In the Dynamic Storage Allocation (DSA) problem, we are given a set of jobs $\mathcal{J}$, with each job $J \in \mathcal{J}$ specified by an interval $I(J)$ and an integer size $s(J)$. An allocation α is defined as a mapping from $\mathcal{J}$ to nonnegative integers $\mathbb{N}$, such that for any two different jobs $J_i, J_j \in \mathcal{J}$, either $I(J_i)$ and $I(J_j)$ are disjoint or $[\alpha(J_i), \alpha(J_i) + s(J_i))$ and $[\alpha(J_j), \alpha(J_j) + s(J_j))$ are disjoint. The target is to find an allocation α with $\max_{J \in \mathcal{J}} \alpha(J) + s(J)$ minimized. A natural geometrical definition of DSA is to treat each job as a rectangle (horizontally ranging over a fixed interval $I(J)$ and vertically having a integer height $s(J)$). The set of rectangles $\mathcal{J}$ is to be packed into a horizontal stripe of minimum height by sliding the rectangles vertically but not horizontally, such that no two rectangles can overlap with each other. DSA is used to model the memory allocation for arrays in computer programs [47], where the size of a job represents the dimension of an array and the job's interval represents the period during which this array is active. The objective of DSA is equivalent to determining the minimum amount of memory needed such that each array can be allocated continuous memory sub-blocks during its active interval and no two arrays active at the same time are assigned overlapping sub-blocks.

Garey and Johnson [37] showed that the offline version of DSA is NP-complete, even if the size of each job is either 1 or 2. Thus, existing work focused on proposing approximation algorithms for DSA. An instance of DSA is called aligned if the size of each job is a power of 2. Based on a First Fit coloring algorithm designed for weighted interval graph coloring, Kierstead [47] designed a 40-approximation algorithm for any aligned instance of DSA, which implied a 80-approximation algorithm for DSA. Later, Slusarek [72] and Kierstead [48] independently proposed approximation algorithms to achieve a better approximation ratio of 6. Based on a 2-allocation technique, Gergov [38, 39] designed a new approximation algorithm, which further improved the approximation ratio to 3. The current best approximation algorithm is designed by Buchsbaum *et al.* [22] based on a

boxing technique, which achieves an approximation ratio of $2 + \varepsilon$, where ε is a positive value that can be arbitrarily small.

For the online version of DSA, the allocation α on a job must be determined at its arrival with no information of the jobs arriving in the future. Robson [64, 65] established a lower bound of $\Omega(\log \omega)$ on the competitive ratio of any deterministic online algorithm, where ω is the max/min job size ratio. He also proposed an optimal $O(\log \omega)$ -competitive online algorithm. Later in [66], he showed that the classical First Fit algorithm also achieves a competitive ratio of $\Theta(\log \omega)$. Based on a segregated storage method which partitions the memory into blocks with each block only serving the requests with similar sizes, Luby *et al.* [58] showed that the First Fit algorithm achieves a competitive ratio of $\Theta(\min\{\log \kappa, \log \omega\})$, where κ is the maximum number of concurrently running requests. By applying a “classification on job length” strategy, Naor *et al.* [60] proposed a $O(\min\{\log \mu, \log \omega\})$ -competitive online algorithm, where μ represents the max/min job length ratio.

In the Dynamic Bin Packing (DBP) problem, we are given a set of items arriving and departing over time. Each item is specified by a size no larger than 1. All the items are to be packed into a minimum number of bins with unit capacity, such that the total size of the items placed in any bin never exceeds 1 at any time. Since DBP generalizes the classical bin packing problem, many classical bin packing algorithms can be directly applied to DBP. Coffman *et al.* [31] studied the competitiveness of the First Fit packing algorithm and showed that First Fit packing achieves a competitive ratio between 2.75 and 2.897. Besides, they also showed that no online packing algorithm can achieve a competitive ratio less than 2.5 against an optimal offline adversary that can repack items at any time for free. Chan *et al.* [24] studied a special case of DBP in which each item’s size is a multiple of $\frac{1}{k}$ for some given integer k . They showed that no online packing algorithm can achieve a competitive ratio less than 2.428 and any Any Fit packing algorithm achieves a competitive ratio no less than 3. They also showed that First Fit packing achieves a competitive ratio between 2.45 and 2.492. Ivkovic *et al.* [43] studied a more general problem called the fully DBP problem where the online algorithm is also allowed to repack items at any time for free. They proposed a 1.25-competitive online algorithm for this problem.

In essence, DBP generalizes the cost minimization version of the basic interval scheduling problem by assuming each job has an arbitrary size. Standard DBP does not consider the duration of bin usage. Consequently, an optimal solution to DBP may incur a large amount of unnecessary bin usage time. Consider an instance in which $2n$ items arrive at the same time 0. Among these items, n items have the same size $1 - \frac{1}{n}$ and the same departure time 1, while the other n items have the same size $\frac{1}{n}$ and the same departure time $n > 1$. An optimal DBP solution for this instance is to use n bins to pack all the items, which however would incur a total bin usage time of n^2 . To minimize the total bin usage time for packing all the items, an optimal solution is to pack all the items of size $\frac{1}{n}$ into one bin and pack each item of size $1 - \frac{1}{n}$ into a separate bin. In this way, a total bin usage time is only $2n$, but there are $n + 1$ bins used. We can see that in terms of bin usage time, the ratio between these two packing solutions is $\frac{n^2}{2n}$, which can be made arbitrarily large when n goes towards infinity. This suggests that a good packing solution for standard DBP can be very bad for minimizing the total bin usage time. Thus, we need new algorithms and analysis for the dynamic bin packing problem to minimize the total bin usage time. In Chapter 3 of this thesis, we shall study a variant of DBP called the MinUsageTime DBP problem, which aims at packing a set of items to minimize the accumulated bin usage time.

2.4 Bandwidth Allocation Problem and Storage Allocation Problem

In this section, we introduce the Bandwidth Allocation Problem [62] and the Storage Allocation Problem [13].

In the Bandwidth Allocation Problem (BAP), we are given a set of jobs $\mathcal{J}$, with each job $J \in \mathcal{J}$ specified by an interval $I(J)$, a size $s(J) \in (0, 1]$ and a weight $w(J)$. The target is to choose a maximum-weight subset $\mathcal{J}'$ such that at any time, the total size of the jobs in $\mathcal{J}'$ never exceeds 1. Note that when each job has the same size 1, BAP reduces to the revenue maximization version of the basic interval scheduling problem with one machine available. In the offline setting, it is easy to prove the NP-hardness of BAP through a reduction from the classical knapsack problem. Phillips *et al.* [62] proposed a 6-approximation algorithm. Based on the local ratio technique, Bar-Noy *et al.* [13] proposed

a 3-approximation algorithm. Calinescu *et al.* [23] proposed a randomized approximation algorithm, which achieves the current best approximation ratio of $2 + \varepsilon$. Besides, they also proposed a 3-approximation deterministic algorithm based on the coloring algorithm designed by Phillips *et al.* [62]. Chen *et al.* [26] proposed a 3-approximation algorithm for a special case where the weight of each job J is given by its “area” $\text{len}(I(J)) \times s(J)$. In the online setting, Bar-Noy *et al.* [14] proposed a $\frac{2}{1-2\delta}$ -competitive algorithm when each job has a size bounded by $\delta \leq \frac{1}{2}$. They also proposed a $\frac{2+\sqrt{5}}{1-\delta}$ -competitive algorithm when each job has a size bounded by δ .

In the Storage Allocation Problem (SAP), each job J is also specified by an interval $I(J)$, a size $s(J) \in (0, 1]$ and a weight $w(J)$. The target is also to choose a maximum-weight subset $\mathcal{J}'$ from a given set of jobs $\mathcal{J}$. Different from BAP, an allocation α , which is a mapping from $\mathcal{J}'$ to $[0, 1)$, must be determined, such that $\alpha(J) + s(J) \leq 1$ holds for any job $J \in \mathcal{J}'$, and for any two different jobs $J_i, J_j \in \mathcal{J}'$, either $I(J_i)$ and $I(J_j)$ are disjoint or $[\alpha(J_i), \alpha(J_i) + s(J_i))$ and $[\alpha(J_j), \alpha(J_j) + s(J_j))$ are disjoint. Obviously, a feasible subset $\mathcal{J}'$ in SAP must be feasible in BAP, but not vice versa. It is also easy to see that SAP shares some similarities with DSA: both problems can be seen as a rectangle packing problem in which each rectangle can only be moved vertically but not horizontally. SAP can be seen as a revenue maximization variant of DSA. In the offline setting, Bar-Noy *et al.* [13] designed a 7-approximation algorithm based on the 3-approximation BAP algorithm and the 2-allocation technique introduced in [38, 39]. Chen *et al.* [26] considered a special case of SAP where each job’s size is a multiple of $\frac{1}{k}$ for some given integer k . They developed a $O(n^{k+1}k^k)$ -time dynamic programming algorithm to solve this case. In addition, they also proposed a $(\frac{e}{e-1} + \varepsilon)$ -approximation algorithm when each job’s size is $\frac{O(1)}{k}$.

2.5 Machine Minimization Problem and Real-Time Scheduling Problem

In this section, we introduce the machine minimization problem [63] and the real-time scheduling problem [15], both of which adopt a flexible job model to generalize the basic interval scheduling problem. In this model, each job J is specified by an arrival time

$a(J)$, a deadline $d(J)$ and a processing length $p(J)$, such that J has to run for $p(J)$ time units continuously on a machine during the time interval $[a(J), d(J))$. The time interval $[a(J), d(J))$ is known as J 's window and $d(J) - a(J) - p(J)$ is known as J 's laxity. If the laxity of a flexible job is non-zero, the job does not have to be placed onto a machine and start its execution immediately at its arrival. If the laxity is zero, a flexible job reduces to an interval job. In Chapter 5 of this thesis, we also consider a problem adopting this flexible job model.

In the machine minimization problem [63], a set of n flexible jobs is to be processed on a minimum number of machines, such that at most one job is running on any machine at any time. In the offline setting, Chuzhoy and Naor [29] studied the approximability of the machine minimization problem. Raghavan and Thompson [63] proposed a $O(\frac{\log n}{\log \log n})$ -approximation algorithm by formulating any given instance as a linear program and finding a feasible solution through the randomized rounding technique. Chuzhoy *et al.* [28] confirmed that the natural linear programming formulation for this problem has an integrality gap of $\Theta(\frac{\log n}{\log \log n})$. To further improve the approximation, they strengthened the natural linear programming relaxation in a recursive manner by forbidding certain configurations which are infeasible in an integral solution, and proposed a $O(\sqrt{\frac{\log n}{\log \log n}})$ -approximation algorithm. So far, no $O(1)$ -approximation algorithm has been found for the machine minimization problem. In the online setting, Saha [68] established a lower bound of $\Omega(\log \mu)$ on the competitive ratio of any deterministic online algorithm and proposed an optimal $O(\log \mu)$ -competitive online algorithm, where μ is the max/min job processing length ratio. Devanur *et al.* [32] studied online machine minimization when each job has the same processing length. They proposed a e -competitive online algorithm and showed that no deterministic online algorithm can achieve a competitive ratio better than e in this case.

In the real-time scheduling problem [15], each job J is associated with an arrival $a(J)$, a deadline $d(J)$, a processing length $p(J)$ and a weight $w(J)$. The target is to choose a maximum-weight subset of jobs to be processed on a given number of machines, such that at most one job is running on any machine at any time. For the unweighted case with one machine available, Bar-Noy *et al.* [15] showed that a simple greedy strategy is 2-approximate: once the machine becomes idle, choose the flexible job that can finish

first among all the pending jobs. For the unweighted case with m machine available, they proposed a $\frac{(1+1/m)^m}{(1+1/m)^m-1}$ -approximation strategy by recursively applying the above strategy to schedule jobs onto each machine one after another. Later, Chuzhoy [30] proposed an improved $\frac{e}{e-1}$ -approximation algorithm for the unweighted case with one machine available. For the weighted case, Bar-Noy *et al.* [15] proposed a 2-approximation algorithm when one machine is available and a 3-approximation algorithm when multiple machines are available. Later, based on the local ratio and primal-dual techniques, Bar-Noy *et al.* [13] and Berman and DasGupta [18] independently proposed 2-approximation algorithms for the weighted case with multiple machines available.

The machine minimization problem and the real-time scheduling problem also have preemptive versions, which allow each job J to be preempted on a machine and resume its execution later on the same or any other machine to complete its processing length within its window. Horn [40] showed that the preemptive machine minimization problem can be solved in polynomial time via the minimum cost flow technique. For the semi-online version of this problem where the optimal number of machines used for an instance is known before online scheduling, Phillips *et al.* [61] established a lower bound of $\Omega(1)$ on the competitive ratio of any deterministic online algorithm. They also showed that the Least Laxity First algorithm achieves a competitive ratio of $O(\log \mu)$, where μ represents the max/min job processing length ratio. Besides, they also showed that the Earliest Deadline First algorithm achieves a competitive ratio of $\Omega(\log \mu)$. Chen *et al.* [27] proposed a $O(\log m)$ -competitive online algorithm for this semi-online version, where m denotes the optimal number of machines used for scheduling all the jobs. Recently, Azar and Cohen [11] proposed an improved $O(\frac{\log m}{\log \log m})$ -competitive online algorithm. For the preemptive real-time scheduling problem, Lawler [51] proposed a polynomial time algorithm via the dynamic programming technique to solve the unweighted case given only one machine available. Based on this technique, they proposed a pseudo-polynomial algorithm for the general case with multiple machines available. Later, Lawler *et al.* [52] proved the NP-hardness of this problem.

Chapter 3

The MinUsageTime Dynamic Bin Packing Problem

In this chapter, we study the MinUsageTime Dynamic Bin Packing (DBP) problem defined as follows. We are given a set of items, each specified by a size, an arrival time and a departure time. The items are to be packed into bins such that the total size of the items placed in each bin does not exceed the bin capacity at any time. A bin is opened when it receives the first item and is closed when all the items in the bin depart. The usage time of a bin is defined as the duration from its opening to its closing. The target is to minimize the total bin usage time for packing all the items.

The MinUsageTime DBP problem is closely related to the interval job scheduling problem with bounded parallelism [5, 25, 35, 50, 59, 70, 76] defined as follows. A set of interval jobs are to be processed. Each job is to be assigned to run on a machine from its arrival time to its departure time. Each machine can simultaneously process at most a fixed number of g jobs, where g stands for the machine capacity. A machine is considered busy when there is at least one job running on it. The objective of scheduling is to minimize the total busy time of all the machines for processing the given set of interval jobs. This problem is also known as the Fiber Minimization problem in [5, 50, 76]. The MinUsageTime DBP problem generalizes the above problem in that items are allowed to have arbitrary sizes, so the maximum number of items that can be placed in a bin is not fixed. Winker and Zhang [76] first defined the interval job scheduling problem with bounded parallelism and proved its NP-hardness through a reduction from the Circular Arc Coloring problem. In the offline setting, Alicherry and Bhatia [5] developed a 2-approximation algorithm through a network flow formulation. Kumar and Rudra [50] also

proposed a 2-approximation algorithm based on the 2-allocation technique introduced by Gergov [38, 39]. Flammini *et al.* [35] introduced a greedy First-Fit algorithm which gives a 4-approximation. Recently, Chang and Khuller [25] proposed another 3-approximation algorithm called GreedyTracking. In addition, several special cases (the proper case, the clique case, etc.) of the problem were investigated by Flammini *et al.* [35] and Mertzios *et al.* [59]. In the online setting, Shalom *et al.* [70] established a tight bound g on the competitiveness of this problem and also studied several special cases where better competitiveness can be achieved.

The MinUsageTime DBP problem was first brought up by Li *et al.* [55, 56]. In the previous work, Li *et al.* studied the problem in the non-clairvoyant online setting, in which the item duration is not known when an item arrives and thus cannot be used for packing purposes. They showed that no deterministic online packing algorithm can achieve a competitive ratio less than μ [55], where μ is the max/min item duration ratio among all the items to pack. They also showed that no Any Fit packing algorithm can achieve a competitive ratio less than $\mu + 1$, and that the First Fit packing algorithm, which is a specific Any Fit packing algorithm, achieves a competitive ratio of $2\mu + 7$ [56]. Our work in this chapter substantially improves the results of Li *et al.* [55, 56]. In particular, we prove that First Fit packing achieves a competitive ratio of $\mu + 3$, indicating that it is near optimal for the MinUsageTime DBP problem in the non-clairvoyant online setting. We further study the problem in the clairvoyant online setting and the offline setting.

In the following text, we first define the notations and provide some preliminaries for studying the MinUsageTime DBP problem in Section 3.1. Then, we study the problem in the non-clairvoyant online, clairvoyant online, and offline settings in Sections 3.2, 3.3 and 3.4 respectively.

3.1 Preliminaries

We first define some key notations used in this chapter. For any time interval I , let I^- and I^+ denote the left and right endpoints of I respectively. For technical reasons, we shall view intervals as half-open, i.e., $I = [I^-, I^+)$. Let $\text{len}(I) = I^+ - I^-$ denote the length of time interval I .

$$\text{span}(\{r_1, r_2, r_3\}) = I(r_2)^+ - I(r_1)^- + I(r_3)^+ - I(r_3)^-$$

Figure 3.1: Span of an item list

For any item r , let $I(r)$ denote the time interval from r 's arrival to its departure. We say that item r is *active* during the interval $I(r)$ and we refer to $I(r)$ as r 's *active interval*. The length of $I(r)$ is called the *item duration*. Let $s(r)$ denote the size of item r . The product of the size and duration of item r , i.e., $s(r) \cdot \text{len}(I(r))$, represents its *time-space demand*. Given a set of items $\mathcal{R}$, we use $s(\mathcal{R}, t)$ to denote the total size of all the items in $\mathcal{R}$ that are active at time t , i.e., $s(\mathcal{R}, t) = \sum_{r \in \mathcal{R}: t \in I(r)} s(r)$. We refer to the time duration in which at least one item in $\mathcal{R}$ is active as the *span* of $\mathcal{R}$ and denote it by $\text{span}(\mathcal{R})$ (see Figure 3.1). In addition, let $\mu = \frac{\max_{r \in \mathcal{R}} \text{len}(I(r))}{\min_{r \in \mathcal{R}} \text{len}(I(r))}$ denote the ratio of the maximum item duration to the minimum item duration among all the items in $\mathcal{R}$.

Without loss of generality, we assume that each bin has a capacity 1. In the packing process, a bin is *opened* when it receives the first item. When all the items in a bin depart, the bin is *closed*. The duration from the opening to the closing of a bin is referred to as its *usage time*. At any time, the total size of all the active items placed in an open bin is referred to as the *bin level*.

It is easy to establish the following three lower bounds on the optimal bin usage time $\text{OPT}_{DBP}(\mathcal{R})$ for packing a set of items $\mathcal{R}$:

Proposition 3.1. $\text{OPT}_{DBP}(\mathcal{R}) \geq \int_{\bigcup_{r \in \mathcal{R}} I(r)} s(\mathcal{R}, t) \, dt = \sum_{r \in \mathcal{R}} (s(r) \cdot \text{len}(I(r)))$.

Proposition 3.2. $\text{OPT}_{DBP}(\mathcal{R}) \geq \int_{\bigcup_{r \in \mathcal{R}} I(r)} 1 \, dt = \text{span}(\mathcal{R})$.

Proposition 3.3. $\text{OPT}_{DBP}(\mathcal{R}) \geq \int_{\bigcup_{r \in \mathcal{R}} I(r)} \lceil s(\mathcal{R}, t) \rceil \, dt$.

The first bound is derived by assuming that no capacity of any bin is wasted at any time. The second bound is derived from the fact that at least one bin must be open at any time when at least one item is active. The third bound is derived by exploiting that the number of open bins must be at least $\lceil s(\mathcal{R}, t) \rceil$ at any time t . Obviously, the third bound is tighter than the first two bounds.

3.2 Non-Clairvoyant Online Setting

We start by considering the MinUsageTime DBP problem in the non-clairvoyant online setting, where the departure time of each item is not known at its arrival and thus cannot be used for packing purposes. We first establish a lower bound on the competitiveness of the Any Fit family of packing algorithms in Section 3.2.1. We improve the lower bound from $\mu + 1$ to $\frac{4\mu+3}{\mu+2}$ when $\mu < \frac{1+\sqrt{5}}{2}$, where μ is the max/min item duration ratio. In Section 3.2.2, we show that the First Fit packing algorithm, which is a particular Any Fit packing algorithm, achieves a competitive ratio of $\mu + 3$.

3.2.1 The Any Fit Family of Packing Algorithms

Any Fit refers to a family of packing algorithms, which never open a new bin for an incoming item if the item can fit into any current open bin. Several classical packing algorithms fall into this family, including First Fit, Best Fit and Worst Fit. Li *et al.* [56] has shown that no Any Fit packing algorithm can achieve a competitive ratio less than $\mu + 1$. In this section, we establish an improved lower bound on the competitive ratio of the Any Fit family of packing algorithms.

Let k be an integer at least 3. At time 0, let $k - 1$ items of size $\frac{1}{k}$ arrive, followed by $k - 1$ items of size $\frac{k-1}{k}$. As shown in Figure 3.2(a), Any Fit packing needs to open k bins to pack all these items. The first bin is used to pack the $k - 1$ items of size $\frac{1}{k}$ and the other $k - 1$ bins are used to pack one item of size $\frac{k-1}{k}$ each. At time 1, let three items of size $\frac{2}{3k}$ and $k - 2$ items of size $\frac{1}{k}$ arrive in sequence. Then, each open bin has enough space left to pack only one of these items. Thus, as shown in Figure 3.2(b), a new bin has to be opened to pack the item arrived last (of size $\frac{1}{k}$). As soon as these $k + 1$ items are placed into the bins, let all the items arriving at time 0 depart (see Figure 3.2(c)). At time $\mu + 1$, let all the items arriving at time 1 depart.

Figure 3.2: An instance for establishing the lower bound on the competitiveness of the Any Fit family of packing algorithms

As shown in Figure 3.2, in the above packing process, k bins are opened from time 0 to 1, and $k + 1$ bins are opened from time 1 to $\mu + 1$. Therefore, the total bin usage time by applying any Any Fit packing algorithm is $AF_{DBP}(\mathcal{R}) = k + (k + 1) \cdot \mu$. On the other hand, in an optimal packing, from time 0 to 1, every pair of items with sizes $\frac{1}{k}$ and $\frac{k-1}{k}$ can be packed into one bin so that only $k - 1$ bins are needed. From time 1 to $\mu + 1$, all the items arriving at time 1 can be packed into one bin since their total size is $3 \cdot \frac{2}{3k} + (k - 2) \cdot \frac{1}{k} = 1$. Therefore, the total bin usage time of the optimal packing is $OPT_{DBP}(\mathcal{R}) = (k - 1) + \mu$. It follows that

$$\frac{AF_{DBP}(\mathcal{R})}{OPT_{DBP}(\mathcal{R})} = \frac{k + (k + 1) \cdot \mu}{(k - 1) + \mu} = \mu + 1 - \frac{\mu^2 - \mu - 1}{k - 1 + \mu}.$$

It is easy to see that when $\mu^2 - \mu - 1 < 0$ (i.e., $1 \leq \mu < \frac{1+\sqrt{5}}{2}$), we have $\frac{AF_{DBP}(\mathcal{R})}{OPT_{DBP}(\mathcal{R})} > \mu + 1$ and the ratio $\frac{AF_{DBP}(\mathcal{R})}{OPT_{DBP}(\mathcal{R})}$ decreases with k . When k is set to 3, the above ratio

reaches the maximum value at $\mu + 1 - \frac{\mu^2 - \mu - 1}{k - 1 + \mu} = \frac{4\mu + 3}{\mu + 2}$.

When $\mu^2 - \mu - 1 \geq 0$ (i.e., $\mu \geq \frac{1 + \sqrt{5}}{2}$), we have $\frac{AF_{DBP}(\mathcal{R})}{OPT_{DBP}(\mathcal{R})} \leq \mu + 1$ and the ratio $\frac{AF_{DBP}(\mathcal{R})}{OPT_{DBP}(\mathcal{R})}$ increases with k . As k goes towards infinity, the above ratio can be made arbitrarily close to $\mu + 1$. Hence, we have the following conclusion.

Theorem 3.1. *For the MinUsageTime DBP problem in the non-clairvoyant online setting, no Any Fit packing algorithm can achieve a competitive ratio less than $\frac{4\mu + 3}{\mu + 2}$ when $\mu < \frac{1 + \sqrt{5}}{2}$ and less than $\mu + 1$ when $\mu \geq \frac{1 + \sqrt{5}}{2}$, where μ is the max/min item duration ratio among all the items to pack.*

3.2.2 The First Fit Packing Algorithm

We now analyze the competitive ratio of the First Fit packing algorithm for the MinUsageTime DBP problem. The First Fit packing algorithm is a particular Any Fit packing algorithm and works as follows. Each time when a new item arrives, if there are one or more open bins that can accommodate the new item, First Fit packing places the item in the bin which was opened earliest among these bins. If no open bin can accommodate the new item, a new bin is opened to receive the item.

The main idea of the competitive analysis is to divide the bin usage time resulting from First Fit packing into two portions: the first portion is equal to the span of all the items and is thus capped by the lower bound of Proposition 3.2. The second portion consists of the bin usage periods in which there are at least two open bins. According to the First Fit packing rule, whenever a new bin is opened for an incoming item r , the sum of r 's size and the level of any open bin must exceed the bin capacity. Exploiting this observation, we break the usage period of each bin into subperiods and strategically charge the length of each subperiod to the time-space demand of relevant items. In this way, we bound the second portion of bin usage time with respect to the total time-space demand of all the items and hence the lower bound of Proposition 3.1. Combining the two portions of bin usage time, we show that First Fit packing achieves a competitive ratio of $\mu + 3$.

Suppose a total of m bins $B_1, B_2, \dots, B_m$ are used by First Fit packing to pack a set of items $\mathcal{R}$. For each bin B_k , let $U_k = [U_k^-, U_k^+)$ denote the usage period of B_k , i.e.,

Figure 3.3: An example of usage periods

the period from the time when B_k is opened to the time when B_k is closed. Then, the total bin usage time of First Fit packing is given by the total length of the usage periods of all the bins used, i.e.,

$$FF_{DBP}(\mathcal{R}) = \sum_{k=1}^m \text{len}(U_k).$$

Without loss of generality, assume that the bins are indexed in the chronological order of their openings, i.e., $U_1^- \leq U_2^- \leq \dots \leq U_m^-$. For each bin B_k ($k \geq 2$), let e_k be the latest closing time of all the bins that are opened before B_k , i.e.,

$$e_k = \max \{ \{U_i^+ : 1 \leq i < k\} \cup \{U_k^-\} \}.$$

We divide the usage period U_k of each bin into two parts: V_k and W_k . V_k is the period $[U_k^-, \min\{U_k^+, e_k\})$. $W_k = U_k - V_k = [\min\{U_k^+, e_k\}, U_k^+)$ is the remaining period. For bin B_1 , define $V_1 = \emptyset$ and $W_1 = U_1$. Figure 3.3 shows an example of these definitions.

According to the definitions, we have $\text{len}(U_k) = \text{len}(V_k) + \text{len}(W_k)$. Clearly, for any two different bins B_{k_1} and B_{k_2} , $W_{k_1} \cap W_{k_2} = \emptyset$. It is also easy to see that

$$\text{span}(\mathcal{R}) = \text{len} \left(\bigcup_{k=1}^m W_k \right) = \sum_{k=1}^m \text{len}(W_k).$$

Moreover, $V_1 = \emptyset$ and hence $\text{len}(V_1) = 0$. Therefore,

$$\begin{aligned}
 FF_{DBP}(\mathcal{R}) &= \sum_{k=1}^m \text{len}(U_k) \\
 &= \sum_{k=1}^m \left(\text{len}(V_k) + \text{len}(W_k) \right) \\
 &= \sum_{k \geq 2} \text{len}(V_k) + \text{span}(\mathcal{R}).
 \end{aligned} \tag{3.1}$$

For each $k \in \{2, \dots, m\}$, let $\mathcal{R}_k$ denote the set of all the items placed in bin B_k by First Fit packing. Let $\mathcal{R}'_k$ be a minimal subset of $\mathcal{R}_k$ such that the union of their active intervals completely covers the period V_k . That is, for any subset $\mathcal{R}''_k \subset \mathcal{R}'_k$, $\bigcup_{r \in \mathcal{R}''_k} I(r)$ cannot fully cover the period V_k . It is easy to see that all the items in $\mathcal{R}'_k$ must have distinct arrival times. This is because if two items share the same arrival times, the active interval of one item must be fully contained in that of the other item and thus can be removed from $\mathcal{R}'_k$, contradicting that $\mathcal{R}'_k$ is minimal.

Let $n(k)$ denote the number of items in $\mathcal{R}'_k$ and let $r_{k,1}, r_{k,2}, \dots, r_{k,n(k)}$ be the items in $\mathcal{R}'_k$ sorted according to their arrival times, i.e., $I(r_{k,1})^- < I(r_{k,2})^- < \dots < I(r_{k,n(k)})^-$. Then, these items must satisfy the following properties. First, it holds that

$$I(r_{k,1})^+ < I(r_{k,2})^+ < \dots < I(r_{k,n(k)})^+.$$

Otherwise, if $I(r_{k,i})^+ \geq I(r_{k,j})^+$ for some $i < j$, the active interval of item $r_{k,j}$ is fully contained in the active interval of item $r_{k,i}$, which contradicts the definition of $\mathcal{R}'_k$. Second, for each $r_{k,i}$, we have $I(r_{k,i})^- < V_k^+$, i.e., $r_{k,i}$ arrives during interval V_k . Otherwise, $r_{k,i}$ can be removed from $\mathcal{R}'_k$ without compromising its coverage over V_k .

If $V_k = \emptyset$, we have $\mathcal{R}'_k = \emptyset$. If $V_k \neq \emptyset$, as shown in Figure 3.4, we can split V_k into $n(k)$ disjoint sub-intervals at the arrivals of the items in $\mathcal{R}'_k$:

$$\begin{aligned}
 X(r_{k,1}) &= [I(r_{k,1})^-, I(r_{k,2})^-), \\
 X(r_{k,2}) &= [I(r_{k,2})^-, I(r_{k,3})^-), \\
 &\dots\dots\dots \\
 X(r_{k,n(k)-1}) &= [I(r_{k,n(k)-1})^-, I(r_{k,n(k)})^-), \\
 X(r_{k,n(k)}) &= [I(r_{k,n(k)})^-, V_k^+].
 \end{aligned}$$

Figure 3.4: Splitting V_k for bin B_k where $\mathcal{R}'_k = \{r_{k,1}, r_{k,2}, r_{k,3}, r_{k,4}, r_{k,5}, r_{k,6}, r_{k,7}\}$

We refer to the above intervals as the X-intervals of items $r_{k,1}, r_{k,2}, \dots, r_{k,n(k)}$ respectively. Obviously, the total length of the X-intervals is $\text{len}(V_k)$, i.e.,

$$\sum_{r \in \mathcal{R}'_k} \text{len}(X(r)) = \text{len}(V_k). \quad (3.2)$$

We define

$$d_k = \sum_{r \in \mathcal{R}'_k} \left(s(r) \cdot \text{len}(X(r)) \right).$$

Clearly, d_k is a lower bound on the total time-space demand of all the items placed on bin B_k since the X-interval of each item is shorter than or equal to its active interval:

$$d_k \leq \sum_{r \in \mathcal{R}'_k} \left(s(r) \cdot \text{len}(I(r)) \right) \leq \sum_{r \in \mathcal{R}_k} \left(s(r) \cdot \text{len}(I(r)) \right). \quad (3.3)$$

Recall that each item r in $\mathcal{R}'_k$ is placed in bin B_k during interval V_k . According to the definition of V_k , there must exist at least one open bin with an index lower than k at time $X(r)^- = I(r)^-$ when r arrives. Among all the open bins with indexes lower than k at time $X(r)^-$, we define the last open bin (the bin with the highest index) as the *supplier bin* of $X(r)$. Note that different items in $\mathcal{R}'_k$ may have different supplier bins as shown in Figure 3.5. By the definition of First Fit packing, the level of $X(r)$'s supplier bin at time $X(r)^-$ plus r 's size must be larger than 1 (the bin capacity). Let $P(r)$ be the set of all the active items already packed in $X(r)$'s supplier bin at r 's arrival time. Then,

$$s(r) + \sum_{\hat{r} \in P(r)} s(\hat{r}) > 1. \quad (3.4)$$

Figure 3.5: An example of supplier bins

We define

$$d^* = \sum_{k \geq 2} \sum_{r \in \mathcal{R}'_k} \sum_{\hat{r} \in P(r)} \left(s(\hat{r}) \cdot \text{len}(X(r)) \right).$$

It then follows from (3.2) and (3.4) that

$$\begin{aligned} \sum_{k \geq 2} d_k + d^* &= \sum_{k \geq 2} \left(\sum_{r \in \mathcal{R}'_k} \left(s(r) + \sum_{\hat{r} \in P(r)} s(\hat{r}) \right) \cdot \text{len}(X(r)) \right) \\ &> \sum_{k \geq 2} \left(\sum_{r \in \mathcal{R}'_k} \text{len}(X(r)) \right) \\ &= \sum_{k \geq 2} \text{len}(V_k). \end{aligned} \tag{3.5}$$

We next show that d^* is bounded by $(\mu + 1) \cdot \sum_{r \in \mathcal{R}} \left(s(r) \cdot \text{len}(I(r)) \right)$.

Lemma 3.1. $d^* \leq (\mu + 1) \cdot \sum_{r \in \mathcal{R}} \left(s(r) \cdot \text{len}(I(r)) \right)$.

Proof: We define

$$\widehat{\mathcal{R}} = \bigcup_{k \geq 2} \left(\bigcup_{r \in \mathcal{R}'_k} P(r) \right).$$

Obviously, $\widehat{\mathcal{R}} \subseteq \mathcal{R}$. For each item $\hat{r} \in \widehat{\mathcal{R}}$, let $P^{-1}(\hat{r})$ denote the set of all the items r in $\bigcup_{k \geq 2} \mathcal{R}'_k$ such that $\hat{r} \in P(r)$. Note that the items in $P^{-1}(\hat{r})$ may come from different bins. Figure 3.6 shows an example of $P^{-1}(\hat{r})$.

 Figure 3.6: An example of $P^{-1}(\hat{r}) = \{r_1, r_2, r_3, r_4, r_5, r_6\}$

We consider the X-interval lengths of the items in $P^{-1}(\hat{r})$. For each item $r \in P^{-1}(\hat{r})$, since item $\hat{r}$ has already been packed when r arrives, $\hat{r}$ cannot arrive later than r . Thus,

$$X(r)^- = I(r)^- \geq I(\hat{r})^-. \quad (3.6)$$

By definition, $I(\hat{r})$ overlaps with the X-interval of each item in $P^{-1}(\hat{r})$, which suggests that $X(r)^- < I(\hat{r})^+$. Recall that the max/min item duration ratio is μ . This implies that

$$\text{len}(X(r)) \leq \text{len}(I(r)) \leq \mu \cdot \text{len}(I(\hat{r})).$$

Therefore,

$$X(r)^+ = X(r)^- + \text{len}(X(r)) < I(\hat{r})^+ + \mu \cdot \text{len}(I(\hat{r})) = I(\hat{r})^- + (\mu + 1) \cdot \text{len}(I(\hat{r})). \quad (3.7)$$

We now show that all the X-intervals of the items in $P^{-1}(\hat{r})$ are disjoint. Consider any two items r_1 and r_2 in $P^{-1}(\hat{r})$. If r_1 and r_2 are placed in the same bin, by the definition of X-intervals, $X(r_1)$ cannot intersect with $X(r_2)$. If r_1 and r_2 are placed on two different bins, without loss of generality, suppose r_1 and r_2 are placed on bins B_{j_1} and B_{j_2} respectively where $j_1 < j_2$. Note that $X(r_1)$ and $X(r_2)$ share the same supplier bin that has an index lower than j_1 and j_2 . According to the definition of supplier bin, when r_2 is placed into bin B_{j_2} , bin B_{j_1} must be closed. Otherwise, r_2 's supplier bin must have an index at least j_1 , which leads to a contradiction. Since B_{j_1} is closed when r_2

arrives, r_1 must have departed when r_2 arrives. This suggests that $X(r_1)$ cannot overlap with $X(r_2)$.

Therefore, it follows from (3.6) and (3.7) that

$$\begin{aligned} \sum_{r \in P^{-1}(\hat{r})} \text{len}(X(r)) &\leq \max_{r \in P^{-1}(\hat{r})} X(r)^+ - \min_{r \in P^{-1}(\hat{r})} X(r)^- \\ &< I(\hat{r})^- + (\mu + 1) \cdot \text{len}(I(\hat{r})) - I(\hat{r})^- \\ &= (\mu + 1) \cdot \text{len}(I(\hat{r})). \end{aligned}$$

Thus, we have

$$\begin{aligned} d^* &= \sum_{k \geq 2} \sum_{r \in \mathcal{R}'_k} \sum_{\hat{r} \in P(r)} \left(s(\hat{r}) \cdot \text{len}(X(r)) \right) \\ &= \sum_{r \in \bigcup_{k \geq 2} \mathcal{R}'_k} \sum_{\hat{r} \in P(r)} \left(s(\hat{r}) \cdot \text{len}(X(r)) \right) \\ &= \sum_{\hat{r} \in \hat{\mathcal{R}}} \left(s(\hat{r}) \cdot \sum_{r \in P^{-1}(\hat{r})} \text{len}(X(r)) \right) \\ &\leq \sum_{\hat{r} \in \hat{\mathcal{R}}} \left(s(\hat{r}) \cdot (\mu + 1) \cdot \text{len}(I(\hat{r})) \right) \\ &\leq (\mu + 1) \cdot \sum_{r \in \mathcal{R}} \left(s(r) \cdot \text{len}(I(r)) \right). \end{aligned}$$

Hence, the lemma is proven. $\square$

Following from (3.1), (3.3), (3.5) and Lemma 3.1, the total bin usage time by applying First Fit packing satisfies

$$\begin{aligned} FF_{DBP}(\mathcal{R}) &= \left(\sum_{k \geq 2} \text{len}(V_k) \right) + \text{span}(\mathcal{R}) \\ &< \left(\sum_{k \geq 2} d_k \right) + d^* + \text{span}(\mathcal{R}) \\ &\leq \sum_{k \geq 2} \sum_{r \in \mathcal{R}_k} \left(s(r) \cdot \text{len}(I(r)) \right) + (\mu + 1) \cdot \sum_{r \in \mathcal{R}} \left(s(r) \cdot \text{len}(I(r)) \right) + \text{span}(\mathcal{R}) \\ &\leq \sum_{r \in \mathcal{R}} \left(s(r) \cdot \text{len}(I(r)) \right) + (\mu + 1) \cdot \sum_{r \in \mathcal{R}} \left(s(r) \cdot \text{len}(I(r)) \right) + \text{span}(\mathcal{R}) \\ &= (\mu + 2) \cdot \sum_{r \in \mathcal{R}} \left(s(r) \cdot \text{len}(I(r)) \right) + \text{span}(\mathcal{R}) \\ &\leq (\mu + 2) \cdot OPT_{DBP}(\mathcal{R}) + OPT_{DBP}(\mathcal{R}) \\ &= (\mu + 3) \cdot OPT_{DBP}(\mathcal{R}), \end{aligned} \tag{3.8}$$

where the last inequality follows from Propositions 3.1 and 3.2. Thus, First Fit packing achieves a competitive ratio of $\mu + 3$.

Theorem 3.2. *For the MinUsageTime DBP problem in the non-clairvoyant online setting, the First Fit packing algorithm achieves a competitive ratio of $\mu + 3$, where μ is the max/min item duration ratio among all the items to pack.*

Recall that Li *et al.* [55] showed that no deterministic online packing algorithm can achieve a competitive ratio less than μ . Thus, our result indicates that the First Fit packing algorithm is near optimal for the MinUsageTime DBP problem in the non-clairvoyant online setting.

3.3 Clairvoyant Online Setting

In this section, we study the MinUsageTime DBP problem in the clairvoyant online setting, where the departure time of each item is known at its arrival and thus can be used for packing purposes. We first establish a lower bound $\Omega\left(\sqrt{\frac{\log \mu}{\log \log \mu}}\right)$ on the competitive ratio of any deterministic online packing algorithm in Section 3.3.1. Next, we introduce a classification strategy based on item durations and apply it to the First Fit packing algorithm. We show that the resultant algorithm is $O(\log \mu)$ -competitive in general and can become $O\left(\frac{\log \mu}{\log \log \mu}\right)$ -competitive if μ is known a priori.

3.3.1 A Lower Bound on Competitiveness of Any Deterministic Online Packing Algorithm

For the interval job scheduling problem with bounded parallelism, Shalom *et al.* [70] derived a lower bound of g on the competitive ratio of any deterministic online algorithm, where g represents the maximum number of interval jobs that can be concurrently processed on one machine. This lower bound is not applicable to our MinUsageTime DBP problem because items can have arbitrary sizes in our problem so that the maximum number of items that can be placed in a bin is not fixed. Inspired by the study of [70], we construct a new instance to bound the competitive ratio of any deterministic online packing algorithm by a function of the max/min item duration ratio μ . Our instance also

uses items with the same size $\frac{1}{k}$ such that each bin can accommodate at most k items at any moment. However, we shall consider k as a variable whose value will be properly set later to derive our lower bound of $\Omega\left(\sqrt{\frac{\log \mu}{\log \log \mu}}\right)$.

In our instance, all the items arrive and depart in a time interval $[0, T]$ for some given T . Our goal is to produce a set of items (to be called earmarked items) with a total duration of kT such that there are exactly k items active at any time in $[0, T]$ and these items are placed in k different bins by any deterministic online packing algorithm. We release items in iterations and maintain a parameter t to indicate the release time of the next iteration. At the beginning, t is initialized to 0.

Let k be an integer larger than 1. In the first iteration, we start by releasing a sequence of items at time $t = 0$ with durations $\frac{T}{k(k-1)k}, \frac{T}{k(k-1)k-1}, \frac{T}{k(k-1)k-2}, \dots$ until k bins are opened. We need to release at most $(k-1)k + 1$ items to achieve this, since the first $k-1$ bins can accommodate at most $(k-1) \cdot k$ items. Thus, the last item in the iteration has a duration at most T . We refer to the first item placed in each bin as *an earmarked item*. Then, there are k earmarked items placed in k different bins. Figure 3.7 shows an example for $k = 3$, where $r_{i,j}$ denotes the j th item released in the i th iteration. In the first iteration, the three earmarked items generated are $r_{1,1}$, $r_{1,2}$ and $r_{1,4}$.

In the second iteration, we advance the release time t to the earliest departure time of the k earmarked items. Since all the items released in the first iteration have distinct durations and thus departure times, exactly one earmarked item would depart at time t and the other $k-1$ earmarked items remain active. At time t , we release a new sequence of items with durations $\min\{\frac{T}{k(k-1)k}, T-t\}, \min\{\frac{T}{k(k-1)k-1}, T-t\}, \min\{\frac{T}{k(k-1)k-2}, T-t\}, \dots$ until an item is packed in a bin different from where the remaining $k-1$ earmarked items are packed. The last item packed is again called *an earmarked item*. Note that this new earmarked item may either be packed in the bin of the departing earmarked item or be packed in another open bin. We need to release at most $(k-1)^2 + 1$ items to generate the new earmarked item, since the $k-1$ bins of existing earmarked items can accommodate at most $k-1$ additional items each. After the second iteration, there are again k earmarked items placed in k different bins. In the example of Figure 3.7, the release time of the second iteration is $t = \frac{T}{36}$ when $r_{1,1}$ departs. In the second iteration, the first item released, i.e., $r_{2,1}$, is the new earmarked item.

Figure 3.7: An example of the first four iterations in the item releasing process when $k = 3$

The above process is repeated iteratively. In each iteration, we advance the release time t to the earliest departure time of the k earmarked items active at the previous release time. Different from the second iteration, in subsequent iterations, there may be multiple earmarked items departing together at time t . Suppose that h ($1 \leq h \leq k$) earmarked items depart together and the other $k - h$ earmarked items remain active. At time t , we release a new sequence of items with durations $\min\{\frac{T}{k(k-1)k}, T - t\}$, $\min\{\frac{T}{k(k-1)k-1}, T - t\}$, $\min\{\frac{T}{k(k-1)k-2}, T - t\}, \dots$ until h items are packed in h distinct bins different from where the remaining $k - h$ earmarked items are placed. These h bins may either be the bins of departing earmarked items or other open bins. The first item placed in each of these h bins are again called *earmarked items*. We need to release at most

$$\begin{aligned} (k-1)(k-h) + k \cdot (h-1) + 1 &= k^2 - 2k + h + 1 \\ &\leq k^2 - k + 1 \\ &= (k-1) \cdot k + 1 \end{aligned}$$

items to generate h new earmarked items, since the $k - h$ bins of existing earmarked items can accommodate at most $k - 1$ additional items each and the bins of the first $h - 1$ new earmarked items can accommodate at most k items each. After the iteration, there are still k earmarked items packed in k different bins. In the example of Figure 3.7, two earmarked items $r_{1,2}$ and $r_{3,1}$ depart together at time $\frac{T}{3^5}$. In the next iteration, two new earmarked items $r_{4,3}$ and $r_{4,4}$ are generated and both of them are placed in the bins of departing earmarked items.

The iterative process proceeds until the release time t reaches T . This can always be achieved eventually since each earmarked item either has a length at least $\frac{T}{k(k-1)k}$ (a fixed value) or keeps active till time T (when its duration takes the form of $T - t$). Note that the departure time of any item does not exceed T . Thus, when the iterative process completes, there are exactly k earmarked items departing together at time T .

With the above adversary, the total bin usage time by applying any deterministic online packing algorithm is at least kT , i.e., the total duration of all the earmarked items. Now, we consider the optimal packing solution for all the items released. We prove that the total bin usage time of the optimal packing solution must be bounded by $T + \frac{2kT}{k-1}$. Obviously, all the earmarked items can be packed in one bin with a usage time

of T since there are exactly k earmarked items active at any time. What is left is to show that all the non-earmarked items can be packed in bins with a total usage time at most $\frac{2kT}{k-1}$.

Note that each iteration ends with releasing a new earmarked item. We divide all the iterations into two groups: the first group contains iterations ending with releasing an earmarked item departing at time T and the second group contains the remaining iterations. For each iteration of the first group, the total duration of all the non-earmarked items released in the iteration is bounded by

$$\sum_{i=1}^{(k-1)k} \min\left\{\frac{T}{k^i}, T - t\right\} \leq \sum_{i=1}^{(k-1)k} \frac{T}{k^i} < \left(\sum_{i=1}^{\infty} \frac{1}{k^i}\right) \cdot T = \frac{T}{k-1},$$

where t is the release time of the iteration. There are at most k iterations in the first group since we have exactly k earmarked items departing at time T . Thus, the aggregate duration of all the non-earmarked items released in these iterations is bounded by $\frac{kT}{k-1}$.

For each iteration of the second group, all the items released in the iteration must have exponentially increasing durations. Thus, if the earmarked item released last has a duration τ , the total duration of all the non-earmarked items released in the iteration is bounded by

$$\sum_{i=1}^{(k-1)k} \frac{\tau}{k^i} < \left(\sum_{i=1}^{\infty} \frac{1}{k^i}\right) \cdot \tau = \frac{\tau}{k-1}.$$

Since the total duration of all the earmarked items is kT , this suggests that the aggregate duration of all the non-earmarked items released in the iterations of the second group is bounded by $\frac{kT}{k-1}$. Putting the two groups of iterations together, the total duration of all the non-earmarked items released in all the iterations is bounded by $\frac{2kT}{k-1}$, which implies that the total bin usage time for optimally packing these items is no more than $\frac{2kT}{k-1}$ according to Proposition 3.2. Thus, the optimal packing solution has a total bin usage time at most $T + \frac{2kT}{k-1}$.

Therefore, the competitive ratio of any deterministic online packing algorithm is at least

$$\frac{kT}{T + \frac{2kT}{k-1}} = \frac{1}{\frac{1}{k} + \frac{2}{k-1}} > \frac{k-1}{3}.$$

Obviously, in our instance, the maximum possible item duration is T . Note that in the first iteration, the first item released has a duration $\frac{T}{k^{(k-1)k}}$ and all the subsequent items released have durations that are multiples of $\frac{T}{k^{(k-1)k}}$. Thus, the release time of the second iteration must be a multiple of $\frac{T}{k^{(k-1)k}}$ and as a result, all the items released in the second iteration have durations that are multiples of $\frac{T}{k^{(k-1)k}}$. By induction, the durations of all the items released in each subsequent iteration must be multiples of $\frac{T}{k^{(k-1)k}}$. This implies that the minimum item duration in our instance is $\frac{T}{k^{(k-1)k}}$. Thus, the max/min item duration ratio μ satisfies $\mu \leq k^{(k-1)k}$, which suggests $k = \Omega\left(\sqrt{\frac{\log \mu}{\log \log \mu}}\right)$. Hence, we have the following conclusion.

Theorem 3.3. *For the MinUsageTime DBP problem in the clairvoyant online setting, any deterministic online packing algorithm has a competitive ratio of $\Omega\left(\sqrt{\frac{\log \mu}{\log \log \mu}}\right)$, where μ is the max/min item duration ratio among all the items to pack.*

3.3.2 A Classify-by-Duration Strategy

In Section 3.2.2, we have established an upper bound of $\mu + 3$ on the competitive ratio of First Fit packing in the non-clairvoyant online setting. Observing that the competitive ratio of First Fit packing is a linear function of the max/min item duration ratio μ , we propose a classify-by-duration strategy to make the packing algorithm more competitive by reducing the max/min ratio of item durations. Specifically, we classify all the items into categories such that the max/min item duration ratio for each category is a given constant α . Given a base item duration b , each category includes all the items with durations in the range $[b \cdot \alpha^{k-1}, b \cdot \alpha^k)$ for an integer k . Note that the max/min item duration ratio for the entire item set $\mathcal{R}$ is μ . Then, there can be up to $\lceil \log_\alpha \mu \rceil + 1$ non-empty item categories produced by the classification.¹ Let $\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3, \dots, \mathcal{R}_{\lceil \log_\alpha \mu \rceil + 1}$ denote these item categories. Since the departure time and thus the duration of each item is known at its arrival, the category of the item can be determined at the time of its assignment. We use First Fit packing to pack the items in each category $\mathcal{R}_k$ separately.

¹For example, if $\alpha = 2$ and the minimum and maximum item durations are 1.5 and 4.5 respectively, there are $\lceil \log_2 3 \rceil + 1 = 3$ non-empty item categories: $[1, 2)$, $[2, 4)$ and $[4, 8)$.

In Section 3.2.2, we have shown in (3.8) that given any set of items $\mathcal{R}$, the total bin usage time by applying First Fit packing is bounded by

$$(\mu + 2) \cdot \sum_{r \in \mathcal{R}} \left(s(r) \cdot \text{len}(I(r)) \right) + \text{span}(\mathcal{R}).$$

By applying the above classify-by-duration strategy, the bin usage time for packing all the items in each category $\mathcal{R}_k$ ($1 \leq k \leq \lceil \log_\alpha \mu \rceil + 1$) is thus bounded by

$$(\alpha + 2) \cdot \sum_{r \in \mathcal{R}_k} \left(s(r) \cdot \text{len}(I(r)) \right) + \text{span}(\mathcal{R}_k).$$

Therefore, the total bin usage time for packing the entire item set $\mathcal{R}$ is bounded by

$$(\alpha + 2) \cdot \sum_{k=1}^{\lceil \log_\alpha \mu \rceil + 1} \left(\sum_{r \in \mathcal{R}_k} \left(s(r) \cdot \text{len}(I(r)) \right) \right) + \sum_{k=1}^{\lceil \log_\alpha \mu \rceil + 1} \text{span}(\mathcal{R}_k). \quad (3.9)$$

Note that

$$\sum_{k=1}^{\lceil \log_\alpha \mu \rceil + 1} \left(\sum_{r \in \mathcal{R}_k} \left(s(r) \cdot \text{len}(I(r)) \right) \right) = \sum_{r \in \mathcal{R}} \left(s(r) \cdot \text{len}(I(r)) \right)$$

and for $1 \leq k \leq \lceil \log_\alpha \mu \rceil + 1$, by definition,

$$\text{span}(\mathcal{R}_k) \leq \text{span}(\mathcal{R}).$$

As a result, the total bin usage time by applying Classify-by-Duration First Fit packing is bounded by

$$\begin{aligned} & (\alpha + 2) \cdot \sum_{r \in \mathcal{R}} \left(s(r) \cdot \text{len}(I(r)) \right) + (\lceil \log_\alpha \mu \rceil + 1) \cdot \text{span}(\mathcal{R}) \\ & \leq (\alpha + \lceil \log_\alpha \mu \rceil + 3) \cdot \text{OPT}_{DBP}(\mathcal{R}). \end{aligned}$$

where the last inequality follows from Proposition 3.1 and Proposition 3.2.

This shows that the Classify-by-Duration First Fit packing algorithm achieves a competitive ratio of $O(\log \mu)$. We remark that the above result significantly improves the existing one of online interval job scheduling with bounded parallelism. Shalom *et al.* [70] proposed a BucketFirstFit algorithm that classifies interval jobs according to their lengths and applies First Fit to each category of jobs. They showed that BucketFirstFit has a

competitive ratio of $(2\alpha + 2) \cdot \lceil \log_\alpha \mu \rceil$, where μ is the max/min length ratio of all job intervals and α is the max/min length ratio of job intervals in each category. Our result implies that BucketFirstFit can achieve a competitive ratio of $\alpha + \lceil \log_\alpha \mu \rceil + 3$ which is dominated by $(2\alpha + 2) \cdot \lceil \log_\alpha \mu \rceil$ asymptotically. Our result is also much more general in that we allow items to have arbitrary sizes while the analysis of [70] is restricted to jobs with the same resource demands.

There exists an optimal value of the algorithm parameter α that minimizes the factor $\alpha + \lceil \log_\alpha \mu \rceil + 3$. If the max/min item duration ratio μ is known before the packing process, α can be set to satisfy $\alpha = \log_\alpha \mu$ such that $\alpha = \Theta(\frac{\log \mu}{\log \log \mu})$. Hence, we have the following result.

Theorem 3.4. *For the MinUsageTime DBP problem in the clairvoyant online setting, the Classify-by-Duration First Fit packing algorithm achieves a competitive ratio of $O(\log \mu)$ when μ is not known and $O(\frac{\log \mu}{\log \log \mu})$ when μ is known a priori, where μ is the max/min item duration ratio among all the items to pack.*

Next, we show that the competitive ratio of the Classify-by-Duration First Fit packing algorithm has a lower bound of $\Omega(\frac{\log \mu}{\log \log \mu})$ even with the minimum and maximum item durations known a priori. Suppose that Δ is the minimum item duration and $\mu\Delta$ is the maximum item duration. Let b be the base item duration and α be the max/min item duration ratio for each item category chosen by the Classify-by-Duration First Fit algorithm. Note that b and α may be chosen based on the knowledge of Δ and $\mu\Delta$. We construct two instances for which the ratios between the Classify-by-Duration First Fit packing algorithm and the optimal packing solution are at least $\Omega(\log_\alpha \mu)$ and $\Omega(\alpha)$ respectively. This suggests that the Classify-by-Duration First Fit packing algorithm has a competitive ratio at least $\Omega(\frac{\log \mu}{\log \log \mu})$, regardless of the b and α values chosen.

Let $m = \lceil \log_\alpha \mu \rceil$. In our first instance, we release $(m + 1)$ series of items. All the items have the same size $\frac{1}{m+1}$. As shown in Figure 3.8, each series i ($1 \leq i \leq m$) contains $\lceil \alpha^{m-i+1} \rceil$ items with the same durations $\alpha^{i-1}\Delta$. The first item arrives at time 0 and each subsequent item arrives as soon as the previous item departs. The last item departs at time $\lceil \alpha^{m-i+1} \rceil \cdot \alpha^{i-1}\Delta$. The last series, i.e., series $(m + 1)$ contains one item with a duration $\mu\Delta$ arriving at time 0.

Figure 3.8: First instance of Classify-by-Duration First Fit packing

By applying the classify-by-duration strategy, each of the first m item series is classified into a separate category. Since the items of each category have a span of $\lceil \alpha^{m-i+1} \rceil \cdot \alpha^{i-1} \Delta \geq \alpha^m \Delta$, the total bin usage time of the Classify-by-Duration First Fit packing algorithm is at least $m\alpha^m \Delta$. On the other hand, the optimal solution is to pack all the items into one bin. Note that the span of the last series is $\mu\Delta \leq \alpha^m \Delta$. Thus, the optimal solution yields a total bin usage time of

$$\begin{aligned} \max_{1 \leq i \leq m} \lceil \alpha^{m-i+1} \rceil \cdot \alpha^{i-1} \Delta &< \max_{1 \leq i \leq m} (\alpha^{m-i+1} + 1) \cdot \alpha^{i-1} \Delta \\ &= \max_{1 \leq i \leq m} (\alpha^m \Delta + \alpha^{i-1} \Delta) \\ &< 2\alpha^m \Delta. \end{aligned}$$

This suggests that the ratio between the two solutions is at least $\frac{m}{2} \geq \frac{1}{2} \log_{\alpha} \mu$.

Recall that given b and α , the duration range of the i th item category is $[b\alpha^{i-1}, b\alpha^i)$. Suppose that the minimum item duration Δ falls in the j th item category (i.e., $b\alpha^{j-1} \leq \Delta < b\alpha^j$) and the maximum item duration $\mu\Delta$ falls in the k th category (i.e., $b\alpha^{k-1} \leq \mu\Delta < b\alpha^k$). Our second instance is constructed as follows. In the case that $j < k-1$, we let n groups of items arrive in sequence at time 0. As shown in Figure 3.9, each group contains n items, among which one item has a duration $b\alpha^{j+1} - \delta$ (where δ is a small value) and all the other items have durations $b\alpha^j$. After that, we let two final items of respective durations Δ and $\mu\Delta$ arrive. All the items have the same size $\frac{1}{n}$. By applying

Figure 3.9: Second instance of Classify-by-Duration First Fit packing

the classify-by-duration strategy, all the items except the final two are classified into the same category. By First Fit packing, each group of items is placed in a separate bin. Thus, the total bin usage time of packing all the items is at least $n(b\alpha^{j+1} - \delta)$. On the other hand, the optimal solution is to pack all the items with durations $b\alpha^j$ into $(n - 1)$ bins, pack all the items with durations $b\alpha^{j+1} - \delta$ into one bin, and pack the final two items into another bin. This leads to a total bin usage time of $(n - 1)b\alpha^j + (b\alpha^{j+1} - \delta) + \mu\Delta$. Thus, the ratio between the Classify-by-Duration First Fit algorithm and the optimal solution is at least

$$\frac{n(b\alpha^{j+1} - \delta)}{(n - 1)b\alpha^j + (b\alpha^{j+1} - \delta) + \mu\Delta},$$

which can be made arbitrarily close to α as n goes towards infinity and δ approaches 0.

In the case that $j = k - 1$, one of the two ratios $\frac{b\alpha^j}{\Delta}$ and $\frac{\mu\Delta}{b\alpha^j}$ must be at least $\sqrt{\mu}$ since $\frac{b\alpha^j}{\Delta} \cdot \frac{\mu\Delta}{b\alpha^j} = \mu$. Without loss of generality, suppose that $\frac{b\alpha^j}{\Delta} \geq \sqrt{\mu}$. Then, we can construct an instance similar to the above by replacing the item durations $b\alpha^j$ and $b\alpha^{j+1} - \delta$ in each group with Δ and $b\alpha^j$. It can be shown that the ratio between the Classify-by-Duration First Fit packing algorithm and the optimal solution is at least $\frac{b\alpha^j}{\Delta} \geq \sqrt{\mu}$. Similarly, in the case that $j = k$, we can set the item durations to Δ and $\mu\Delta$ in each group so that the ratio between the two solutions is at least $\frac{\mu\Delta}{\Delta} = \mu$. In summary, for our second instance, the ratio between the Classify-by-Duration First Fit packing algorithm and the optimal solution is at least $\min\{\alpha, \sqrt{\mu}, \mu\} = \min\{\alpha, \sqrt{\mu}\}$.

From the two instances constructed above, given any value of the algorithm parameters b and α , the competitive ratio of the Classify-by-Duration First Fit packing algorithm is at least $\max\{\frac{1}{2}\log_\alpha \mu, \min\{\alpha, \sqrt{\mu}\}\}$. If $\alpha \geq \sqrt{\mu}$, this yields a lower bound of $\max\{\frac{1}{2}\log_\alpha \mu, \sqrt{\mu}\} \geq \sqrt{\mu}$ on the competitive ratio of the algorithm, which dominates $\frac{\log \mu}{\log \log \mu}$ asymptotically. If $\alpha < \sqrt{\mu}$, the lower bound is given by $\max\{\frac{1}{2}\log_\alpha \mu, \alpha\}$. Note that $\max\{\frac{1}{2}\log_\alpha \mu, \alpha\}$ can be minimized when $\frac{1}{2}\log_\alpha \mu = \alpha$, in which case $\alpha = \Theta(\frac{\log \mu}{\log \log \mu})$. Thus, $\max\{\frac{1}{2}\log_\alpha \mu, \alpha\} = \Omega(\frac{\log \mu}{\log \log \mu})$. Hence, we can have the following conclusion.

Theorem 3.5. *For the MinUsageTime DBP problem in the clairvoyant online setting, the competitive ratio of the Classify-by-Duration First Fit packing algorithm has a lower bound of $\Omega(\frac{\log \mu}{\log \log \mu})$ even when the minimum item duration and the maximum item duration are known a priori, where μ is the max/min item duration ratio among all the items to pack.*

The above result, together with Theorem 3.4, suggests that when setting $\alpha = \log_\alpha \mu$, the competitive ratio $O(\frac{\log \mu}{\log \log \mu})$ of the Classify-by-Duration First Fit packing algorithm is tight.

3.4 Offline Setting

In this section, we study the MinUsageTime DBP problem in the offline setting, where the information of all the items is given before the packing process. Note that when all the items have the same arrival time and the same departure time, the MinUsageTime DBP problem reduces to the classical bin packing problem, which is NP-hard. We thus focus on designing approximation algorithms for the problem. We propose a 5-approximation Duration Descending First Fit algorithm in Section 3.4.1, and a 4-approximation Dual Coloring algorithm in Section 3.4.2.

3.4.1 The Duration Descending First Fit Packing Algorithm

The Duration Descending First Fit algorithm works as follows. All the items are first sorted in the descending order of their durations and then packed one by one according to the First Fit rule. That is, each item is placed in the bin of the smallest index that can accommodate the item throughout its duration, where the bins are indexed according to their opening order by the execution of the packing algorithm. If no opened bin can accommodate the item throughout its duration, a new bin is opened for it. This algorithm is also used by [35] for the interval job scheduling problem with bounded parallelism and shown to have an approximation ratio of 4. In interval job scheduling with bounded parallelism, all the jobs have the same resource demands. If each machine can run up to g jobs concurrently, the machine can essentially be viewed as a collection of g mini-machines each for running one job exclusively. These g mini-machines are then considered separately in the analysis of [35]. In contrast, in the MinUsageTime DBP problem, items can have arbitrary sizes so that the maximum number of items that can be placed in each bin is not predictable and it may also vary at different times. Thus, the analysis of [35] is not directly applicable to the MinUsageTime DBP problem since it is impossible to divide each bin into mini-bins.

Inspired by the method used in Section 3.2.2, in what follows, we present a new analysis to show that the Duration Descending First Fit algorithm achieves an approximation ratio of 5 for the MinUsageTime DBP problem. Our key observation is that whenever a new bin is opened to pack an item r , there must exist at least one moment in r 's active interval such that the level of an earlier opened bin plus the size of r is larger than the bin capacity.

Suppose that a total of m bins $B_1, B_2, \dots, B_m$ are used by the Duration Descending First Fit algorithm to pack a set of items $\mathcal{R}$. The bins are indexed in their opening order, i.e., B_1 is the first bin opened by the algorithm and B_m is the last. For each bin B_k , let $\mathcal{R}_k$ denote the set of the items packed into B_k . Then, by definition, the usage time of B_k is $\text{span}(\mathcal{R}_k)$. Thus, the total bin usage time of the Duration Descending First Fit algorithm is given by $\sum_{k=1}^m \text{span}(\mathcal{R}_k)$.

Consider any item set $\mathcal{R}_k$ ($k \geq 2$). We reduce $\mathcal{R}_k$ to a subset $\mathcal{R}'_k$ by removing any item whose active interval is completely contained in that of another item. In this way, the items left in $\mathcal{R}'_k$ satisfy the following property: let n denote the number of items in $\mathcal{R}'_k$ and let $r_1, r_2, \dots, r_n$ be the items sorted according to their arrival times, i.e., $I(r_1)^- < I(r_2)^- < \dots < I(r_n)^-$, then it holds that $I(r_1)^+ < I(r_2)^+ < \dots < I(r_n)^+$. Otherwise, if $I(r_x)^+ \geq I(r_y)^+$ for some $x < y$, the active interval of item r_y is fully contained in the active interval of item r_x , which contradicts the definition of $\mathcal{R}'_k$. It is easy to see that reducing $\mathcal{R}_k$ to $\mathcal{R}'_k$ does not shorten the usage time of bin B_k , i.e., $\text{span}(\mathcal{R}_k) = \text{span}(\mathcal{R}'_k)$.

We next split $\bigcup_{r \in \mathcal{R}'_k} I(r)$ into n disjoint intervals:

$$\begin{aligned} X(r_1) &= [I(r_1)^-, \min\{I(r_2)^-, I(r_1)^+\}), \\ X(r_2) &= [I(r_2)^-, \min\{I(r_3)^-, I(r_2)^+\}), \\ &\dots\dots\dots \\ X(r_{n-1}) &= [I(r_{n-1})^-, \min\{I(r_n)^-, I(r_{n-1})^+\}), \\ X(r_n) &= [I(r_n)^-, I(r_n)^+). \end{aligned}$$

We refer to the above intervals as the X-intervals of items $r_1, r_2, \dots, r_n$ respectively. Figure 3.10 shows an example of splitting $\bigcup_{r \in \mathcal{R}'_k} I(r)$ into X-intervals. Obviously, the

Figure 3.10: An example of splitting $\bigcup_{r \in \mathcal{R}'_k} I(r)$ for the items placed in bin B_k where $\mathcal{R}'_k = \{r_1, r_2, r_3, r_4, r_5, r_6, r_7\}$

total length of the X-intervals is $\text{span}(\mathcal{R}'_k)$, i.e., $\sum_{i=1}^n \text{len}(X(r_i)) = \text{span}(\mathcal{R}'_k)$. We define

$$d_k = \sum_{i=1}^n \left(s(r_i) \cdot \text{len}(X(r_i)) \right),$$

which is a lower bound on the total time-space demand of all the items placed in bin B_k since the duration of each item's X-interval is shorter than or equal to the item duration:

$$\begin{aligned} d_k &\leq \sum_{i=1}^n \left(s(r_i) \cdot \text{len}(I(r_i)) \right) \\ &\leq \sum_{r \in \mathcal{R}_k} \left(s(r) \cdot \text{len}(I(r)) \right). \end{aligned} \quad (3.10)$$

By the definition of Duration Descending First Fit, when an item $r_i \in \mathcal{R}'_k$ is placed in bin B_k ($k \geq 2$), there must exist at least one moment t_i , such that the level of bin B_{k-1} exceeds $1 - s(r_i)$ at time t_i . Let $P(r_i)$ be the set of all the items active at time t_i and placed in bin B_{k-1} before item r_i is placed in bin B_k by the Duration Descending First Fit algorithm. By the algorithm definition, each item in $P(r_i)$ must have a duration no shorter than that of r_i . We define

$$d_k^* = \sum_{i=1}^n \left(\sum_{r \in P(r_i)} \left(s(r) \cdot \text{len}(X(r_i)) \right) \right).$$

It then follows that

$$d_k + d_k^* = \sum_{i=1}^n \left(s(r_i) + \sum_{r \in P(r_i)} s(r) \right) \cdot \text{len}(X(r_i)) > \sum_{i=1}^n \text{len}(X(r_i)) = \text{span}(\mathcal{R}_k). \quad (3.11)$$

We next show that d_k^* is bounded by $3 \cdot \sum_{r \in \mathcal{R}_{k-1}} \left(s(r) \cdot \text{len}(I(r)) \right)$. We define $\hat{\mathcal{R}}_{k-1} = \bigcup_{i=1}^n P(r_i)$, which is a subset of $\mathcal{R}_{k-1}$. For each item $\hat{r} \in \hat{\mathcal{R}}_{k-1}$, let $P^{-1}(\hat{r})$ denote the set

 Figure 3.11: An example of $P^{-1}(\hat{r}) = \{r_1, r_2, r_3, r_4, r_5\}$

of all the items r in $\mathcal{R}'_k$ such that $\hat{r} \in P(r)$. Figure 3.11 shows an example of $P^{-1}(\hat{r})$. It follows from the earlier explanation that each item in $P^{-1}(\hat{r})$ must have a duration no longer than that of $\hat{r}$. Moreover, by definition, $\hat{r}$'s active interval must overlap with each item in $P^{-1}(\hat{r})$. Thus, for each item $r \in P^{-1}(\hat{r})$, we have $I(r)^- \geq I(\hat{r})^- - \text{len}(I(\hat{r}))$ and $I(r)^+ \leq I(\hat{r})^+ + \text{len}(I(\hat{r}))$. Note that the X-intervals of all the items in $\mathcal{R}'_k$ are disjoint. Therefore,

$$\begin{aligned} \sum_{r \in P^{-1}(\hat{r})} \text{len}(X(r)) &\leq \max_{r \in P^{-1}(\hat{r})} I(r)^+ - \min_{r \in P^{-1}(\hat{r})} I(r)^- \\ &\leq \left(I(\hat{r})^+ + \text{len}(I(\hat{r})) \right) - \left(I(\hat{r})^- - \text{len}(I(\hat{r})) \right) \\ &= 3 \cdot \text{len}(I(\hat{r})). \end{aligned}$$

As a result,

$$\begin{aligned} d_k^* &= \sum_{i=1}^n \left(\sum_{r \in P(r_i)} \left(s(r) \cdot \text{len}(X(r_i)) \right) \right) \\ &= \sum_{\hat{r} \in \hat{\mathcal{R}}_{k-1}} \left(s(\hat{r}) \cdot \left(\sum_{r \in P^{-1}(\hat{r})} \text{len}(X(r)) \right) \right) \\ &\leq \sum_{\hat{r} \in \hat{\mathcal{R}}_{k-1}} \left(s(\hat{r}) \cdot 3 \cdot \text{len}(I(\hat{r})) \right) \\ &= 3 \cdot \sum_{r \in \mathcal{R}_{k-1}} \left(s(r) \cdot \text{len}(I(r)) \right). \end{aligned}$$

It follows from (3.10) and (3.11) that

$$\text{span}(\mathcal{R}_k) < d_k + d_k^* \leq \sum_{r \in \mathcal{R}_k} \left(s(r) \cdot \text{len}(I(r)) \right) + 3 \cdot \sum_{r \in \mathcal{R}_{k-1}} \left(s(r) \cdot \text{len}(I(r)) \right).$$

Note that the above inequality holds for any item set $\mathcal{R}_k$ where $k \geq 2$. As a result, we have

$$\begin{aligned}
 \sum_{k=2}^m \text{span}(\mathcal{R}_k) &< \sum_{k=2}^m \left(\sum_{r \in \mathcal{R}_k} (s(r) \cdot \text{len}(I(r))) + 3 \cdot \sum_{r \in \mathcal{R}_{k-1}} (s(r) \cdot \text{len}(I(r))) \right) \\
 &< \sum_{k=1}^m \left(\sum_{r \in \mathcal{R}_k} (s(r) \cdot \text{len}(I(r))) \right) + 3 \cdot \sum_{k=1}^m \left(\sum_{r \in \mathcal{R}_k} (s(r) \cdot \text{len}(I(r))) \right) \\
 &= 4 \cdot \sum_{r \in \mathcal{R}} (s(r) \cdot \text{len}(I(r))).
 \end{aligned}$$

Therefore, the total bin usage time by applying the Duration Descending First Fit algorithm satisfies

$$\begin{aligned}
 \sum_{k=1}^m \text{span}(\mathcal{R}_k) &= \left(\sum_{k=2}^m \text{span}(\mathcal{R}_k) \right) + \text{span}(\mathcal{R}_1) \\
 &< 4 \cdot \sum_{r \in \mathcal{R}} (s(r) \cdot \text{len}(I(r))) + \text{span}(\mathcal{R}_1) \\
 &\leq 4 \cdot \sum_{r \in \mathcal{R}} (s(r) \cdot \text{len}(I(r))) + \text{span}(\mathcal{R}) \\
 &\leq 4 \cdot \text{OPT}_{DBP}(\mathcal{R}) + \text{OPT}_{DBP}(\mathcal{R}) \\
 &= 5 \cdot \text{OPT}_{DBP}(\mathcal{R}),
 \end{aligned}$$

where the last inequality follows from the bounds given in Propositions 3.1 and 3.2. Hence, we have the following conclusion.

Theorem 3.6. *For the MinUsageTime DBP problem in the offline setting, the Duration Descending First Fit algorithm achieves an approximation ratio of 5.*

3.4.2 The Dual Coloring Algorithm

In this section, we present a 4-approximation Dual Coloring algorithm, which is inspired by the Simple algorithm introduced in [50] for the Fiber Minimization problem. We remark that our Dual Coloring algorithm and the Simple algorithm are both based on the 2-allocation technique introduced in [38, 39] for approximating the classical Dynamic Storage Allocation problem.

The Dual Coloring algorithm works as follows. The items in an item set $\mathcal{R}$ are divided into two groups $\mathcal{R}^s$ and $\mathcal{R}^l$, where $\mathcal{R}^s$ contains all the small items of size no larger than

Figure 3.12: Item placement in the demand chart

$\frac{1}{2}$ and $\mathcal{R}^l$ contains all the large items with size strictly larger than $\frac{1}{2}$. These two groups of items are packed separately. For each large item, a new bin is opened to accommodate it from its arrival to its departure. The bins used to pack the large items do not further pack any small item. The small items are packed by constructing a demand chart that describes the total size of the active small items as a function of time (see Figure 3.12 as an example). The horizontal dimension of the demand chart represents the timeline, while the vertical dimension of the demand chart represents the total size of the small items active at any given time. The small items are packed in two phases. In Phase 1, all the small items are placed inside the demand chart such that no three items overlap together. In Phase 2, the small items are packed into bins according to their positions in the demand chart. A formal description of the algorithm for packing all the small items is given in Algorithm 1.

In Phase 1, each item r is represented by a rectangle ranging over its active interval $I(r)$ in the time dimension and of height $s(r)$ in the demand dimension. Thus, the area of the rectangle of item r is equal to its time-space demand $s(r) \cdot \text{len}(I(r))$. The algorithm places the rectangle of each item inside the demand chart. To simplify presentation, we say that item r is placed at altitude h in the demand chart if it occupies the range $(h - s(r), h]$ in the demand dimension. The area of the demand chart is gradually colored either red or blue during the item placement process. To place the items, the algorithm examines a collection of altitudes from high to low in the demand chart. Let *Altitudes* denote the collection of altitudes to examine. Initially, *Altitudes* contains all possible

Algorithm 1: Packing all the small items

Input: $\mathcal{R}^s = \{r \in \mathcal{R} : s(r) \leq \frac{1}{2}\}$.
Output: A packing for all the small items.

- 1 —**Place the small items inside the demand chart**—
- 2 $\mathcal{Altitudes} = \{s(\mathcal{R}^s, t) : t \in \bigcup_{r \in \mathcal{R}^s} I(r)\};$
- 3 **while** $\mathcal{Altitudes} \neq \emptyset$ **do**
- 4 Determine $h = \max_{h' \in \mathcal{Altitudes}} h';$
- 5 Let $\mathcal{Red}$, $\mathcal{Blue}$ and $\mathcal{Uncolored}$ be the respective sets of all maximal red, blue and uncolored intervals at altitude h ;
- 6 **while** $\mathcal{Uncolored} \neq \emptyset$ **do**
- 7 Pick an uncolored interval $I_u \in \mathcal{Uncolored}$;
- 8 **if** $\exists r \in \mathcal{R}^s$ such that $I(r) \cap I_u \neq \emptyset$ and
- 9 $\forall I \in \mathcal{Uncolored} \cup \mathcal{Red} \setminus \{I_u\}, I(r) \cap I = \emptyset$ **then**
- 10 Place item r at altitude h and delete r from $\mathcal{R}^s$;
- 11 Color the rectangle $\{I(r) \cap I_u\} \times (h - s(r), h]$ red;
- 12 Delete I_u from $\mathcal{Uncolored}$;
- 13 Add $[I_u^-, I(r)^-)$ and $[I(r)^+, I_u^+)$ into $\mathcal{Uncolored}$ if they are not empty;
- 14 Add $h - s(r)$ into $\mathcal{Altitudes}$ if $h > s(r)$ and $h - s(r) \notin \mathcal{Altitudes}$;
- 15 **else**
- 16 Color the rectangle $I_u \times (0, h]$ blue;
- 17 Delete I_u from $\mathcal{Uncolored}$;
- 18 Delete h from $\mathcal{Altitudes}$;
- 19 —**Slice the demand chart and pack all the small items**—
- 20 Slice the demand chart into horizontal stripes of height $\frac{1}{2}$ each;
- 21 **for each stripe do**
- 22 Use one bin to pack all the small items placed completely within the stripe;
- 23 **for every two neighboring stripes do**
- 24 Use one bin to pack all the small items placed across the two stripes;

total sizes of active small items over time (step 2), i.e., all the ceiling altitudes in the demand chart. In each iteration, the algorithm examines the next (maximum) altitude h in $\mathcal{Altitudes}$ (step 4). Based on its coloring status, the horizontal line at altitude h in the demand chart is divided into three sets of intervals: red, blue and uncolored (step 5). For each uncolored interval I_u , the algorithm looks for an item r (not yet placed inside the demand chart) whose active interval $I(r)$ intersects with I_u but does not intersect with any other uncolored interval and any red interval (step 9). If there is no such item, all the area below I_u in the demand chart is colored blue (step 16). If such an item r exists, r is placed at altitude h (step 10) (see Figure 3.12 for an example). The uncolored area covered by r is then colored red (step 11). For convenience of algorithm presentation, we intentionally leave r 's lower boundary uncolored for identifying uncolored intervals in the subsequent examination. The altitude of r 's lower boundary, i.e., $h - s(r)$, is added into $\mathcal{Altitudes}$ (step 14) as other items may be able to be placed immediately below r at altitude $h - s(r)$. The examination of an altitude h proceeds until there is no more uncolored interval (step 6). Phase 1 completes when all altitudes in $\mathcal{Altitudes}$ have been examined (step 3).

It can be shown that Phase 1 runs in polynomial time. Note that the total size of active small items can change only upon item arrivals and departures. Thus, the initial set $\mathcal{Altitudes}$ (the set of all possible total sizes of active small items over time) contains at most $2 \cdot |\mathcal{R}^s|$ different altitudes, where $|\mathcal{R}^s|$ is the number of the small items. Subsequently, the placement of each item introduces at most one additional altitude to examine (step 14). Therefore, the total number of altitudes examined in Phase 1 is bounded by $3 \cdot |\mathcal{R}^s|$. In the examination of each altitude, the total number of red, blue and uncolored intervals is bounded by $O(|\mathcal{R}^s|)$ because the endpoints of each interval must correspond to item arrivals or departures. There are at most $|\mathcal{R}^s|$ items to check for each uncolored interval, and checking the eligibility of each item for placement takes $O(|\mathcal{R}^s|)$ time. So, the time complexity for examining each altitude is bounded by $O(|\mathcal{R}^s|^3)$. As a result, the total time complexity of Phase 1 is bounded by $O(|\mathcal{R}^s|^4)$.

Phase 1 gives rise to the following properties of item placement in the demand chart.

Lemma 3.2. *After Phase 1, the entire area of the demand chart is colored.*

Figure 3.13: Item placement in the demand chart

Proof: We prove this claim by showing that when the examination of an altitude h completes, all the area between altitude h and the next altitude h^- to examine ($h^- < h$) is colored.

Consider the coloring status of the horizontal line at altitude h after h has been examined. It is obvious that this line must be fully colored since by definition, the examination terminates only when the uncolored interval set is empty. Thus, after the examination, the horizontal line at altitude h consists of blue intervals and red intervals only. A blue interval b may be colored during the examination of altitude h or an earlier examination of another altitude h^+ ($h^+ > h$). Note that blue coloring can only be done by step 16 of Algorithm 1 which colors blue all the area below the altitude under examination to the bottom of the demand chart. Thus, in either case, all the area below the blue interval b at altitude h must have been colored blue.

Likewise, a red interval e may be colored during the examination of altitude h or an earlier examination of another altitude h^+ ($h^+ > h$). Note that red coloring can only be done due to item placement (step 11). Suppose that e is colored due to the placement of an item r . This implies that r 's lower boundary (at altitude $h - s(r)$ or $h^+ - s(r)$) must be below h . According to step 14 of Phase 1, the altitude of r 's low boundary would be added to *Altitudes* for examination. Since h^- is the next altitude to examine after h , it must hold that $h - s(r) \leq h^-$ or $h^+ - s(r) \leq h^-$. Therefore, the area below the red interval e between altitudes h and h^- are colored red when the examination of altitude h completes.

By similar arguments, when the examination of the last (lowest) altitude h completes, all the area between altitude h and the bottom of the demand chart is colored. We remark that, strictly speaking, the bottom line of the demand chart at altitude 0 is not colored. But this does not affect the correctness of the following analysis for approximation ratio. $\square$

Lemma 3.3. *Whenever a small item r is placed at an altitude h in Phase 1, the rectangle occupied by r is within the demand chart.*

Proof: It is obvious from the algorithm that r 's left boundary (at time $I(r)^-$), right boundary (at time $I(r)^+$) and upper boundary (at altitude h) are within the demand chart. Thus, we only need to prove that r 's lower boundary (at altitude $h - s(r)$) is also within the demand chart.

Based on the proof of Lemma 3.2, at the beginning of an altitude h 's examination, all the area above altitude h has been colored. According to the algorithm, if an item r is allowed to be placed at altitude h , its active interval $I(r)$ must overlap with some uncolored interval I_u at altitude h . It can be inferred that the area above the intersection $I(r) \cap I_u$ at altitude h cannot be colored blue. This is because when blue coloring is performed over some interval, all the area below the interval to the bottom of the demand chart is colored blue (step 16). Thus, the area above the intersection $I(r) \cap I_u$ at altitude h can only be all colored red.

Consider a time $t \in I(r) \cap I_u$. Recall that the height of the demand chart at time t is the total size $s(\mathcal{R}^s, t)$ of all active small items at time t . Thus, at time t , the distance from altitude h to the ceiling of the demand chart is $s(\mathcal{R}^s, t) - h$. Since this range has all been colored red, by the algorithm definition, among all the small items that have been placed before examining altitude h , the total size of those active at time t is $s(\mathcal{R}^s, t) - h$. The new item r to be placed at altitude h is also active at time t . If $h < s(r)$, it implies that the total size of all active small items at time t is at least $s(\mathcal{R}^s, t) - h + s(r) > s(\mathcal{R}^s, t)$, which contradicts the definition of $s(\mathcal{R}^s, t)$. Hence, $h \geq s(r)$ and the lemma is proven. $\square$

Lemma 3.4. *All the small items are placed inside the demand chart after Phase 1.*

Proof: Assume on the contrary that there is an item r left not placed at the end of Phase 1. We check the coloring status of its active interval $I(r)$ at altitude $s(r)$, where $s(r)$ is the size of r . According to Lemma 3.2, $I(r)$ is colored at altitude $s(r)$.

If some part of $I(r)$ is colored red, let's consider a point $(t, s(r))$ that is colored red where $t \in I(r)$. Suppose that $(t, s(r))$ is colored red due to the placement of another item r' at altitude h' . Then, we have $t \in I(r')$ and $h' \geq s(r) > h' - s(r')$. It follows similar arguments to the proof of Lemma 3.3 that among all the small items that have been placed before examining altitude h' , the total size of those active at time t is $s(\mathcal{R}^s, t) - h'$. Note that items r and r' are also active at time t . Thus, the total size of all active small items at time t is at least $s(\mathcal{R}^s, t) - h' + s(r') + s(r) > s(\mathcal{R}^s, t)$, which contradicts the definition of $s(\mathcal{R}^s, t)$.

On the other hand, if the entire interval $I(r)$ is colored blue at altitude $s(r)$, let's consider the maximum altitude h ($h \geq s(r)$) at which the whole interval $I(r)$ is blue. Then, one or more parts of $I(r)$ at altitude h are colored blue during the examination of altitude h . Assume that the last part of $I(r)$ is colored blue due to an uncolored interval I_u satisfying $I(r) \cap I_u \neq \emptyset$. Then, before the last part of $I(r)$ is colored blue, r is an item satisfying the condition for placement (step 9) since $I(r) \cap I_u \neq \emptyset$ and $I(r) \setminus I_u$ has all been colored blue. Thus, instead of coloring I_u (and the area below it) blue, r should have been placed at altitude h , which again leads to a contradiction. Hence, the lemma is proven. $\square$

Lemma 3.5. *No three items overlap together in their placement of Phase 1.*

Proof: Lemma 3.2 has shown that the entire demand chart is colored after Phase 1. According to the algorithm definition, any red area cannot be further covered by any new item after it is colored red. Thus, no two items can overlap in red areas. It is left to show that no three items can overlap together in blue areas. Assume on the contrary that a blue area $[t_1, t_2] \times (h_1, h_2]$ (where $t_1 < t_2$ and $h_1 < h_2$) is covered by three items r_1, r_2 and r_3 in their placement (see Figure 3.14 for an example). Suppose that the placement of each item r_i causes an uncolored interval I_i to be colored red at some altitude. Obviously, I_1, I_2 and I_3 cannot intersect with one another along the time dimension. Moreover, since the area $[t_1, t_2] \times (h_1, h_2]$ is colored blue, we have $I_i \cap [t_1, t_2] = \emptyset$ ($i = 1, 2, 3$). Thus, at

Figure 3.14: Item overlapping in the demand chart

least two of I_1 , I_2 and I_3 must be on the same side of $[t_1, t_2]$. Without loss of generality, assume that I_2 and I_3 are both on the right side of $[t_1, t_2]$, i.e., $I_2^- \geq t_2$ and $I_3^- \geq t_2$. Since I_2 and I_3 do not intersect, assume without loss of generality that I_2 is on the left side of I_3 . Then, it follows that $t_2 \leq I_2^- < I_2^+ \leq I_3^- < I_3^+$. As a result, between r_2 and r_3 , the first item placed would have colored the area $[I_2^-, I_2^+) \times (h_1, h_2]$ red, making it impossible for the second item to be placed. This leads to a contradiction. Therefore, no three items can overlap together in blue areas. $\square$

In Phase 2, the demand chart is partitioned into stripes of height $\frac{1}{2}$ each (see Figure 3.15 for an example). The total number of stripes is $\lceil 2 \cdot \max_{t \in \bigcup_{r \in \mathcal{R}^s} I(r)} s(\mathcal{R}^s, t) \rceil$, where $\max_{t \in \bigcup_{r \in \mathcal{R}^s} I(r)} s(\mathcal{R}^s, t)$ is the maximum height of the demand chart. For each stripe, since the height of the stripe is $\frac{1}{2}$ and no three items overlap together in their placement (Lemma 3.5), the total size of all the small items placed completely within a stripe is bounded by 1 at any time, which suggests that they can be packed into one bin (step 22). On the other hand, note that each small item's size is bounded by $\frac{1}{2}$. This implies that an item can cross at most two stripes in the placement by Phase 1. For each pair of neighboring stripes, by Lemma 3.5, at most two small items can cross them concurrently at any time, so their total size is bounded by 1. Therefore, all the small items placed across a pair of neighboring stripes can be packed into one bin (step 24). It is easy to see that the time complexity of Phase 2 is $O(|\mathcal{R}^s|)$.

Figure 3.15: Item packing based on placement in the demand chart

Now, let's compute the bin usage time by applying the Dual Coloring algorithm. In Phase 2 of packing small items, the number of stripes produced at any time point t is $\lceil 2 \cdot s(\mathcal{R}^s, t) \rceil$. Thus, the number of open bins for packing small items at time t is at most $2 \cdot \lceil 2 \cdot s(\mathcal{R}^s, t) \rceil - 1$, among which $\lceil 2 \cdot s(\mathcal{R}^s, t) \rceil$ bins are used for items placed within individual stripes and $\lceil 2 \cdot s(\mathcal{R}^s, t) \rceil - 1$ bins are used for items placed across stripes. So, the total bin usage time for packing small items is bounded by $\int_{\bigcup_{r \in \mathcal{R}^s} I(r)} (2 \cdot \lceil 2 \cdot s(\mathcal{R}^s, t) \rceil - 1) dt$.

Since the size of each large item is strictly greater than $\frac{1}{2}$, the number of active large items at time t is smaller than $2 \cdot s(\mathcal{R}^l, t)$, where $s(\mathcal{R}^l, t)$ denotes the total size of all active large items at time t . Since each large item is packed into a distinct bin, the number of open bins for packing large items at time t is at most $\lfloor 2 \cdot s(\mathcal{R}^l, t) \rfloor$. Thus, the total bin usage time for packing large items is bounded by $\int_{\bigcup_{r \in \mathcal{R}^l} I(r)} \lfloor 2 \cdot s(\mathcal{R}^l, t) \rfloor dt$.

In summary, the total bin usage time by applying the Dual Coloring algorithm is bounded by

$$\int_{\bigcup_{r \in \mathcal{R}^l} I(r)} \lfloor 2 \cdot s(\mathcal{R}^l, t) \rfloor dt + \int_{\bigcup_{r \in \mathcal{R}^s} I(r)} (2 \cdot \lceil 2 \cdot s(\mathcal{R}^s, t) \rceil - 1) dt.$$

Now, we show that $\lfloor 2 \cdot s(\mathcal{R}^l, t) \rfloor + 2 \cdot \lceil 2 \cdot s(\mathcal{R}^s, t) \rceil - 1 \leq 4 \cdot s(\mathcal{R}, t)$ at any time t . It is easy to verify the above inequality if there are only large items or only small items active at time t . We thus only consider the case when there are both large and small items active at time t . Apparently, $0 \leq \lceil 2 \cdot s(\mathcal{R}^s, t) \rceil - 2 \cdot s(\mathcal{R}^s, t) < 1$. If $0 \leq \lceil 2 \cdot s(\mathcal{R}^s, t) \rceil - 2 \cdot s(\mathcal{R}^s, t) \leq \frac{1}{2}$, we can rewrite the total number of open bins as follows.

$$\begin{aligned} & \lfloor 2 \cdot s(\mathcal{R}^l, t) \rfloor + 2 \cdot \lceil 2 \cdot s(\mathcal{R}^s, t) \rceil - 1 \\ &= \lfloor 2 \cdot s(\mathcal{R}^l, t) \rfloor + 2 \cdot (2 \cdot s(\mathcal{R}^s, t) + (\lceil 2 \cdot s(\mathcal{R}^s, t) \rceil - 2 \cdot s(\mathcal{R}^s, t))) - 1 \\ &= \lfloor 2 \cdot s(\mathcal{R}^l, t) \rfloor + 4 \cdot s(\mathcal{R}^s, t) + 2 \cdot (\lceil 2 \cdot s(\mathcal{R}^s, t) \rceil - 2 \cdot s(\mathcal{R}^s, t)) - 1 \\ &\leq \lfloor 2 \cdot s(\mathcal{R}^l, t) \rfloor + 4 \cdot s(\mathcal{R}^s, t) + 1 - 1 \\ &\leq 4 \cdot s(\mathcal{R}^l, t) + 4 \cdot s(\mathcal{R}^s, t) \\ &\leq 4 \cdot \lceil s(\mathcal{R}^l, t) + s(\mathcal{R}^s, t) \rceil \\ &= 4 \cdot \lceil s(\mathcal{R}, t) \rceil. \end{aligned}$$

If $\frac{1}{2} < \lceil 2 \cdot s(\mathcal{R}^s, t) \rceil - 2 \cdot s(\mathcal{R}^s, t) < 1$, we have $\lceil 4 \cdot s(\mathcal{R}^s, t) \rceil = 2 \cdot \lceil 2 \cdot s(\mathcal{R}^s, t) \rceil - 1$. As a result,

$$\begin{aligned}
 & \lfloor 2 \cdot s(\mathcal{R}^l, t) \rfloor + 2 \cdot \lceil 2 \cdot s(\mathcal{R}^s, t) \rceil - 1 \\
 &= \lfloor 2 \cdot s(\mathcal{R}^l, t) \rfloor + \lceil 4 \cdot s(\mathcal{R}^s, t) \rceil \\
 &\leq \lfloor 2 \cdot s(\mathcal{R}^l, t) \rfloor + 4 \cdot s(\mathcal{R}^s, t) \\
 &\leq \lceil 4 \cdot s(\mathcal{R}^l, t) \rceil + 4 \cdot s(\mathcal{R}^s, t) \\
 &= 4 \cdot \lceil s(\mathcal{R}, t) \rceil.
 \end{aligned}$$

Consequently, we have the following result.

Theorem 3.7. *For the MinUsageTime DBP problem in the offline setting, the Dual Coloring algorithm achieves an approximation ratio of 4.*

3.5 Summary

In this chapter, we have studied the MinUsageTime DBP problem in the non-clairvoyant online, clairvoyant online, and offline settings respectively.

In the non-clairvoyant online setting, we have shown that no Any Fit packing algorithm can achieve a competitive ratio less than $\frac{4\mu+3}{\mu+2}$ when $\mu < \frac{1+\sqrt{5}}{2}$ and less than $\mu + 1$ when $\mu \geq \frac{1+\sqrt{5}}{2}$. We have also shown that the First Fit packing algorithm, which is a particular Any Fit packing algorithm, achieves a competitive ratio of $\mu + 3$. This competitive ratio closely matches a known lower bound μ on the competitiveness of any deterministic online packing algorithm. Besides, it is the first known upper bound for MinUsageTime DBP in the non-clairvoyant online setting that has a multiplicative factor 1 for μ , whereas all the existing upper bounds have multiplicative factors larger than 1 for μ . Our result shows that First Fit packing is near optimal.

In the clairvoyant online setting, we have established a lower bound $\Omega\left(\sqrt{\frac{\log \mu}{\log \log \mu}}\right)$ on the competitive ratio of any deterministic online packing algorithm. We have introduced a classify-by-duration strategy and applied it to the First Fit packing algorithm. We have shown that the resultant algorithm is $O(\log \mu)$ -competitive in general and can become $O\left(\frac{\log \mu}{\log \log \mu}\right)$ -competitive if μ is known a priori.

In the offline setting, we have presented two approximation algorithms: the Duration Descending First Fit algorithm and the Dual Coloring algorithm. We have shown that these two algorithms achieve approximation ratios of 5 and 4 respectively.

Chapter 4

The Energy-Efficient Job Scheduling Problem

In this chapter, we study an Energy-Efficient Job Scheduling problem defined as follows. The input is a set of interval jobs, with each job specified by a size, an arrival time and a processing length. Once a job arrives, it must be placed on a machine immediately and run for a period of its processing length without interruption. The machines to run jobs have capacity limits such that at any time, the total size of the jobs running on any machine cannot exceed its capacity. A machine must be “on” in order to process jobs. When a machine is “on”, its power usage rate is given by a base component representing the power consumed by an idle machine plus a variable component proportional to the machine’s instantaneous load [16]. When a machine is “off”, it does not consume energy. A commonly used strategy for energy saving is to switch the machines “off” when they are not in use. While powering idle machines down can save energy, the machines will have to be powered up again when they are needed for executing new jobs. State transitions between “on” and “off” incur energy overheads. The target is to minimize the total energy consumption of the machines used for processing all the jobs. We shall show that the Energy-Efficient Job Scheduling problem is generalization of the MinUsageTime DBP problem studied in Chapter 3.

Some work has studied the energy conservation problems for a single machine or a group of machines by taking the energy cost of power-up operations into consideration. A classical optimization problem is to consider the power-down strategy on a single machine that has an active state and a sleep state. The target is to minimize the energy

consumed during the machine's idle periods when there is no computation demand. Such a problem is equivalent to the ski-rental problem, a famous rent-or-buy problem. Karlin *et al.* [44] proposed an optimal 2-competitive deterministic online algorithm and an improved $\frac{e}{e-1}$ -competitive randomized online algorithm. Irani *et al.* [42] and Augustine *et al.* [8] studied the power-down strategies on a single machine with an active state and several low-power sleep states. Albers [4] studied an offline problem to minimize the total energy consumption for a group of heterogeneous machines to satisfy the varying total demand for computing capacity over time, but the study ignored specific jobs. Besides, Khuller *et al.* [46, 53] introduced the machine activation problems, where an activation budget is given for activating machines. The target is to minimize the makespan for a batch of jobs on a set of heterogeneous machines. Different from [46, 53], we focus on minimizing the energy usage for processing interval jobs. Azar *et al.* [10] modeled the setup time for booting a virtual machine and developed bi-objective algorithms to minimize the maximum delay of unit-size job completion and the total dollar expenditure of job processing in the cloud. In contrast, we study a unified energy metric that combines the costs for machines to power up and process jobs. While the study of [10] was limited to unit-size jobs, each with a demand equal to the machine capacity, we consider the general case where each job can have an arbitrary demand for machine capacity.

In the following text, we first introduce the definitions and notations in Section 4.1. We also provide a simplification on the target of the Energy-Efficient Job Scheduling problem, which suggests that it can be viewed as a generalization of the MinUsage-Time DBP problem. In Section 4.2, we consider a special case of Energy-Efficient Job Scheduling where each job to be scheduled has a unit size equal to the machine capacity. In Section 4.3, we study the problem in the general case where each job can have an arbitrary size.

4.1 Preliminaries

4.1.1 Notations and Definitions

We introduce the key notations used in this chapter. Again, for any time interval I , we use I^- and I^+ to denote the left and right endpoints of I respectively. Let $\text{len}(I) = I^+ - I^-$ denote the length of interval I .

For each job J , let $s(J)$ denote J 's size. Each job needs to run for a period of its processing length starting from its arrival. We refer to the time interval in which a job J is running (active) as its *active interval* and denote it by $I(J)$. Then, the arrival time and processing length of the job can be represented by $I(J)^-$ and $\text{len}(I(J))$ respectively. Given a set of jobs $\mathcal{J}$, we use $s(\mathcal{J}, t)$ to denote the total size of the jobs active at time t , i.e., $s(\mathcal{J}, t) = \sum_{J \in \mathcal{J}: t \in I(J)} s(J)$. We assume that all the machines have the same capacity. At any time, the total size of the active jobs placed on any machine cannot exceed its capacity. Without loss of generality, the capacity of each machine is assumed to be 1, and each job has a size no larger than 1. If a machine M is powered up for running jobs at time t_1 and then powered down at time t_2 , then we say that M is in the “on” state during the interval $[t_1, t_2)$ and refer to the interval as M 's “on” interval. During an “on” interval, a machine may not always be processing jobs. If at least one job is running on machine M , we say that M is *busy*. If M is “on” but there is no job running on M , we say that M is *idle*. If a machine M is busy (idle) throughout an interval I , we refer to I as a *busy (idle) interval* of machine M .

A machine's energy consumption model is as follows. When the machine is “on”, its power usage rate at any time is given by a base rate r_b representing the power consumed by an idle machine plus a variable rate $r_v \cdot L$ that is proportional to the machine's instantaneous load L (i.e., the total size of the active jobs on this machine at that time) [16]. When a machine is “off”, its power usage rate is 0. The energy overheads incurred for powering a machine up and down are given by C_{up} and C_{down} respectively. To simplify our discussion, we can assume alternatively an initial energy cost $C = C_{up} + C_{down}$ for powering a machine up and no energy cost for powering a machine down. This would not change the total energy expenditure because each machine ever used for running jobs must be powered down eventually.

The target of the Energy-Efficient Job Scheduling problem is to minimize the total energy consumed by the machines for processing all the jobs. To facilitate reasoning, we shall assume that there is a sufficiently large pool of machines and that once a machine used is powered down, the machine is never powered up again. In other words, each machine used has only one “on” interval. Such an assumption makes sense for our problem since after a machine used is powered down, it becomes indistinguishable for any unused machine.

4.1.2 A Simplification of the Target of the Problem

Now, we show that the objective of the Energy-Efficient Job Scheduling problem can be reduced to minimizing a weighted linear combination of the total “on” time and the total number of all the machines used. For each machine M used, let $Q(M)$ denote its “on” interval and $\mathcal{J}_M$ denote the set of jobs assigned to it. Then, the energy consumed by M over its “on” interval $Q(M)$ is given by $\int_{Q(M)} (r_b + r_v \cdot s(\mathcal{J}_M, t)) \, dt$ and thus the total energy consumed by M is

$$\begin{aligned} & C + r_b \cdot \text{len}(Q(M)) + \int_{Q(M)} r_v \cdot s(\mathcal{J}_M, t) \, dt \\ &= C + r_b \cdot \text{len}(Q(M)) + r_v \cdot \sum_{J \in \mathcal{J}_M} s(J) \cdot \text{len}(I(J)). \end{aligned}$$

Let $\mathcal{M}$ be the set of machines used for running a set of jobs $\mathcal{J}$. Since $\bigcup_{M \in \mathcal{M}} \mathcal{J}_M = \mathcal{J}$, the total energy consumed by all machines can be written as

$$C \cdot |\mathcal{M}| + r_b \cdot \sum_{M \in \mathcal{M}} \text{len}(Q(M)) + r_v \cdot \sum_{J \in \mathcal{J}} s(J) \cdot \text{len}(I(J)).$$

Note that the third item above is a constant for a given set of jobs $\mathcal{J}$. We can thus ignore it in the optimization target and consider only the first two items, i.e., $C \cdot |\mathcal{M}| + r_b \cdot \sum_{M \in \mathcal{M}} \text{len}(Q(M))$, where $|\mathcal{M}|$ is the number of machines used and $\sum_{M \in \mathcal{M}} \text{len}(Q(M))$ is the total “on” time of the machines. Without loss of generality, we shall assume the base power rate r_b to be 1 and focus on minimizing $C \cdot |\mathcal{M}| + \sum_{M \in \mathcal{M}} \text{len}(Q(M))$ in the rest of this chapter. Note that without the first item $C \cdot |\mathcal{M}|$, the problem is equivalent to the MinUsageTime DBP problem studied in Chapter 3 by considering each job as an item and each machine as a bin. Thus, the Energy-Efficient Job Scheduling problem can be viewed as a generalization of the MinUsageTime DBP problem.

4.2 Unit Job Size Case

We start by considering a simple case where each job has a size equal to the machine’s capacity. Thus, each machine can run at most one job at any time. We refer to this special case as EEJS-Unit. We propose an optimal scheduling algorithm in the offline setting and a 2-competitive algorithm in the online setting. An instance is also constructed to

show that no deterministic online algorithm can achieve a competitive ratio less than 2, which suggests that the 2-competitive online algorithm is optimal.

Our algorithms work by scheduling jobs in the order of their arrivals. When a job arrives, if there exist one or multiple machines whose last job was completed less than C time units ago, then the incoming job is placed on one of these machines whose last job was completed the latest. Otherwise, a new machine is powered up to run the incoming job. In the offline setting, each machine is powered down when it completes the last job placed on it. Thus, the “on” interval of each machine starts from the arrival of its first job and ends at the completion of its last job. In the online setting, whenever a machine completes a job and becomes idle, it is kept “on” for at most C time units for receiving new incoming jobs. If no incoming job is placed on it for C time units, the machine is then powered down. Therefore, the “on” interval of each machine starts from the arrival of its first job and ends C time units after the completion of its last job. An example of the offline and online schedules constructed are shown in Figure 4.1.

We first prove the optimality of the offline algorithm by converting any optimal schedule A for a set of jobs $\mathcal{J}$ into the schedule produced by our algorithm without incurring any extra energy usage during the conversion process. We index and examine the jobs in their arrival order. Assume that job J_j is the first job that is scheduled differently by the optimal schedule A and our algorithm. Since the jobs $\{J_1, J_2, \dots, J_{j-1}\}$ are scheduled identically by schedule A and our algorithm, before job J_j arrives, the set of machines used by schedule A is identical to the set of machines used by our algorithm. Now, suppose schedule A places job J_j on a machine M' , while our algorithm places job J_j on another machine M . We can convert schedule A to a new schedule that places J_j on M without increasing the energy usage. Specifically, let $\mathcal{J}_M$ and $\mathcal{J}_{M'}$ denote the jobs placed on machines M and M' respectively by schedule A . We apply the following two steps to convert schedule A :

1. Place jobs $\mathcal{J}_{M'} \cap \{J_1, J_2, \dots, J_{j-1}\}$ and $\mathcal{J}_M - \{J_1, J_2, \dots, J_{j-1}\}$ on machine M' .
2. Place jobs $\mathcal{J}_M \cap \{J_1, J_2, \dots, J_{j-1}\}$ and $\mathcal{J}_{M'} - \{J_1, J_2, \dots, J_{j-1}\}$ on machine M . Note that in this way, job J_j is placed on machine M .

Figure 4.1: An example of EEJS-Unit

Since our schedule is feasible, the jobs in $\mathcal{J}_M \cap \{J_1, J_2, \dots, J_{j-1}\}$ do not overlap with J_j and hence all the jobs in $\mathcal{J}_{M'} - \{J_1, J_2, \dots, J_{j-1}\}$. Similarly, since schedule A is feasible, the jobs in $\mathcal{J}_{M'} \cap \{J_1, J_2, \dots, J_{j-1}\}$ do not overlap with J_j and hence all the jobs in $\mathcal{J}_M - \{J_1, J_2, \dots, J_{j-1}\}$. Therefore, the new schedule after the above conversion is feasible. It is easy to infer that the total length of M and M' 's "on" intervals remains unchanged after conversion. Thus, no extra energy usage is incurred due to the conversion. So, we have obtained a new schedule in which jobs $\{J_1, J_2, \dots, J_j\}$ are scheduled identically to our algorithm. By performing similar conversions for every job in $\mathcal{J} - \{J_1, J_2, \dots, J_j\}$, we can obtain an optimal schedule exactly the same as the schedule produced by our algorithm. Thus, we can conclude that the offline algorithm is optimal for solving EEJS-Unit.

Next, we study the competitiveness of the online algorithm. In the online algorithm, if a machine is powered down at time t , it must be "on" and idle during the time interval

$[t - C, t)$. Thus, each machine has an energy usage that is C higher than that in the offline algorithm. Note that the energy usage of each machine is at least C in the offline algorithm since it needs to be powered up initially. Thus, the online algorithm is 2-competitive.

We remark here that the online algorithm does not make use of the processing length information of a job in scheduling it. Thus, it works in both the non-clairvoyant online setting and the clairvoyant online setting.

Finally, we construct an instance to show that no deterministic online algorithm can achieve a competitive ratio less than 2 for EEJS-Unit. In our instance, we release at most n jobs. First, a job J_1 of processing length ε is released at time 0. A new machine must be powered up to run this job and this machine becomes idle at time ε . Obviously, an optimal schedule for this single job is to power down the machine once J_1 is completed. Suppose an online algorithm powers down this machine at time t_1 . If $t_1 \geq C$, we terminate job releasing. Then, the energy usage ratio between the online algorithm and the optimal schedule is $\frac{C+t_1}{C+\varepsilon} \geq \frac{2C}{C+\varepsilon}$. Otherwise, if $t_1 < C$, we release a new job J_2 of processing length ε at time $t_1 + \frac{1}{2}\delta$, where $\delta > 0$ is a small value. Note that the machine used for J_1 has just been powered down at time t_1 and thus a new machine has to be powered up to run J_2 . Suppose the online algorithm powers the new machine down at time $t_1 + \frac{1}{2}\delta + t_2$. Similarly, if $t_2 \geq C$, we terminate job releasing process. The best strategy for scheduling $\{J_1, J_2\}$ is to place them on one machine and power down this machine at time $t_1 + \frac{1}{2}\delta + \varepsilon$. Thus, the energy ratio between the online algorithm and the optimal schedule is $\frac{C+t_1+C+t_2}{C+t_1+\frac{1}{2}\delta+\varepsilon} > \frac{3C+t_1}{C+t_1+\delta+\varepsilon} > \frac{4C}{2C+\delta+\varepsilon} > \frac{2C}{C+\delta+\varepsilon}$. Otherwise, if $t_2 < C$, we continue to release a new job J_3 of processing length ε at time $t_1 + \frac{1}{2}\delta + t_2 + \frac{1}{4}\delta$ and so on. In general, a job J_i of processing length ε is released at time $\sum_{j=1}^{i-1}(t_j + \frac{1}{2^j}\delta)$, after the machine used for J_{i-1} is powered down. Thus, a new machine must be powered on to run J_i . Suppose the online algorithm powers down this new machine at time $\sum_{j=1}^{i-1}(t_j + \frac{1}{2^j}\delta) + t_i$. If $t_i \geq C$, we terminate job releasing. The best strategy for scheduling jobs $\{J_1, \dots, J_i\}$ is to place them on one machine and power down this machine at time $\sum_{j=1}^{i-1}(t_j + \frac{1}{2^j}\delta) + \varepsilon$. So, the energy ratio between the online algorithm and the optimal schedule is $\frac{\sum_{j=1}^{i-1}(C+t_j)+(C+t_i)}{C+\sum_{j=1}^{i-1}(t_j+\frac{1}{2^j}\delta)+\varepsilon} > \frac{(i+1)C+\sum_{j=1}^{i-1}t_j}{C+\sum_{j=1}^{i-1}t_j+\delta+\varepsilon} > \frac{2i \cdot C}{i \cdot C+\delta+\varepsilon} > \frac{2C}{C+\delta+\varepsilon}$. Otherwise, job releasing continues. After n jobs are released, the energy usage of the online algorithm is at least

$\sum_{j=1}^{n-1} (C + t_j) + (C + \varepsilon) = n \cdot C + \sum_{j=1}^{n-1} t_j + \varepsilon$, whereas the energy usage of the optimal schedule is $C + \sum_{j=1}^{n-1} (t_j + \frac{1}{2^j} \delta) + \varepsilon < C + \sum_{j=1}^{n-1} t_j + \delta + \varepsilon$. Thus, the energy ratio is at least $\frac{n \cdot C + \sum_{j=1}^{n-1} t_j + \varepsilon}{C + \sum_{j=1}^{n-1} t_j + \delta + \varepsilon} > \frac{(2n-1) \cdot C + \varepsilon}{n \cdot C + \delta + \varepsilon}$. Therefore, the competitive ratio by applying any deterministic online algorithm is at least

$$\min \left\{ \frac{2C}{C + \varepsilon}, \frac{2C}{C + \delta + \varepsilon}, \frac{(2n-1) \cdot C + \varepsilon}{n \cdot C + \delta + \varepsilon} \right\}.$$

As ε and δ approach 0 and n goes towards infinity, the above bound can be made arbitrarily close to 2. Hence, our online algorithm is optimal for EEJS-Unit.

4.3 Arbitrary Job Size Case

4.3.1 A Lower Bound on Optimal Solution

Now, we study the Energy-Efficient Job Scheduling problem in the general case where jobs can have arbitrary sizes, and refer to it as EEJS-Arbitrary. We first investigate the lower bound on the optimal solution of EEJS-Arbitrary, which shall be used to design and analyze offline and online scheduling algorithms later. We establish a lower bound by relaxing the feasibility requirement of EEJS-Arbitrary. Specifically, we consider a Relaxed Energy-Efficient Job Scheduling (REEJS) problem. Given a set of jobs $\mathcal{J}$, the objective of the REEJS problem is to find a minimum-energy schedule of machines such that at any time t , the aggregate capacity of “on” machines is at least $s(\mathcal{J}, t)$, i.e., the total size of the jobs active at time t . Note that REEJS does not impose the feasibility of actually placing the jobs on the “on” machines. Since each job has to be placed on a single machine throughout its active interval, a feasible placement may not always exist even if the aggregate capacity of “on” machines can meet the total demand $s(\mathcal{J}, t)$ at all times. Thus, the REEJS problem is a relaxation of the EEJS-Arbitrary problem. We shall derive an optimal REEJS schedule for bounding the EEJS-Arbitrary problem.

We compute an optimal REEJS schedule in an iterative manner. In each iteration k , we first determine all the contiguous intervals in which $s(\mathcal{J}, t) > k - 1$. Next, we merge every two consecutive intervals I_a and I_b into one interval if they are no further than C apart, i.e., we replace I_a and I_b with $[I_a^-, I_b^+)$ if $I_b^- - I_a^+ \leq C$. This process is repeated recursively until no more intervals can be merged. Then, the resultant intervals

Figure 4.2: An optimal REEJS schedule

are all further than C apart from each other. For each resultant interval I , we schedule a new machine with an “on” interval I . All the machines used in iteration k , denoted by $\mathcal{M}_k^{REEJS}$, are referred to as the machines of level k . It is easy to see that $\max_t \lceil s(\mathcal{J}, t) \rceil$ iterations are needed by the scheduling process. The schedule produced is given by $\mathcal{M}^{REEJS} = \bigcup_{k=1}^{\max_t \lceil s(\mathcal{J}, t) \rceil} \mathcal{M}_k^{REEJS}$. In this schedule, at any time t , there is a machine with an “on” interval covering t at each of the levels $1, 2, \dots, \lceil s(\mathcal{J}, t) \rceil$. Thus, the schedule meets the capacity requirement of REEJS. Figure 4.2 gives an example to illustrate the construction of an optimal REEJS schedule.

Theorem 4.1. *The algorithm introduced above produces an optimal REEJS schedule that minimizes the total energy consumed by the machines used.*

Proof: We prove it by converting any optimal REEJS schedule for a set of jobs $\mathcal{J}$

into the schedule $\mathcal{M}^{REEJS}$ produced by our algorithm without incurring any extra energy usage.

Let $\mathcal{M}^{opt}$ denote the set of machines used by an optimal schedule for $\mathcal{J}$. Let $Q(M)$ denote the “on” interval of a machine M . Let $N(\mathcal{M}^{opt}, t) = |\{M : M \in \mathcal{M}^{opt}, t \in Q(M)\}|$ denote the number of “on” machines at time t in the schedule of $\mathcal{M}^{opt}$. For each k where $1 \leq k \leq \max_t \lceil s(\mathcal{J}, t) \rceil$, we examine the times t when $N(\mathcal{M}^{opt}, t) \geq k$. These times may constitute one or more contiguous intervals that are apart from each other. For each interval I , we construct a machine with an “on” interval I . All the machines constructed from a set $\mathcal{M}_k$. Then, we create a new schedule $\mathcal{M}^*$ by putting $\mathcal{M}_1, \mathcal{M}_2, \dots, \mathcal{M}_{\max_t \lceil s(\mathcal{J}, t) \rceil}$ together, i.e., $\mathcal{M}^* = \bigcup_{k=1}^{\max_t \lceil s(\mathcal{J}, t) \rceil} \mathcal{M}_k$. It is easy to verify that the schedule $\mathcal{M}^*$ meets the capacity requirement of REEJS. This is because at any time t , if $N(\mathcal{M}^{opt}, t) < \max_t \lceil s(\mathcal{J}, t) \rceil$, there are exactly the same number of “on” machines in $\mathcal{M}^{opt}$ and $\mathcal{M}^*$ at time t ; if $N(\mathcal{M}^{opt}, t) \geq \max_t \lceil s(\mathcal{J}, t) \rceil$, there are $\max_t \lceil s(\mathcal{J}, t) \rceil$ “on” machines in $\mathcal{M}^*$ which are also enough to meet the capacity requirement of $s(\mathcal{J}, t)$ at time t . It also follows from these facts that the total “on” time of the machines $\mathcal{M}^*$ cannot exceed that of the machines $\mathcal{M}^{opt}$. In addition, every time a new machine is powered up in $\mathcal{M}^*$, the number of “on” machines in $\mathcal{M}^{opt}$ must have been increased, so there must also be a machine powered up in $\mathcal{M}^{opt}$. Thus, $|\mathcal{M}^*| \leq |\mathcal{M}^{opt}|$. As a result, the total energy consumption of $\mathcal{M}^*$ is bounded by that of $\mathcal{M}^{opt}$. Therefore, $\mathcal{M}^*$ is also an optimal REEJS schedule for $\mathcal{J}$.

Next, we prove that the machines in $\mathcal{M}^*$ have the same “on” intervals as the machines $\mathcal{M}^{REEJS}$ produced by our algorithm. For each k where $1 \leq k \leq \max_t \lceil s(\mathcal{J}, t) \rceil$, consider the machine set $\mathcal{M}_k \subseteq \mathcal{M}^*$ and the level- k machines $\mathcal{M}_k^{REEJS}$ produced by our algorithm. We sort the machines in $\mathcal{M}_k$ and $\mathcal{M}_k^{REEJS}$ in increasing order of their power-up times, i.e., $\mathcal{M}_k = \{M_{k,1}, M_{k,2}, \dots\}$ and $\mathcal{M}_k^{REEJS} = \{M_{k,1}^{REEJS}, M_{k,2}^{REEJS}, \dots\}$. We first compare machine $M_{k,1}$ with machine $M_{k,1}^{REEJS}$. By the definition of our algorithm, the total demand $s(\mathcal{J}, t)$ must be greater than $k - 1$ at time $Q(M_{k,1}^{REEJS})^-$. Since $\mathcal{M}^{opt}$ meets the capacity requirement of REEJS, we have $N(\mathcal{M}^{opt}, Q(M_{k,1}^{REEJS})^-) \geq k$, which suggests that machine $M_{k,1}$ is powered up no later than $Q(M_{k,1}^{REEJS})^-$. Now, suppose $Q(M_{k,1})^- < Q(M_{k,1}^{REEJS})^-$. Then, according to our algorithm, the total demand $s(\mathcal{J}, t)$ is at most $k - 1$ during the interval $[Q(M_{k,1})^-, Q(M_{k,1}^{REEJS})^-)$. We can thus delay the power-up of

$M_{k,1}$ to reduce its energy usage, which contradicts the optimality of $\mathcal{M}^*$. Therefore, we have $Q(M_{k,1})^- = Q(M_{k,1}^{REEJS})^-$.

Next, we prove that $Q(M_{k,1})^+ = Q(M_{k,1}^{REEJS})^+$. If $Q(M_{k,1})^+ > Q(M_{k,1}^{REEJS})^+$, the total demand $s(\mathcal{J}, t)$ must exceed $k-1$ immediately before time $Q(M_{k,1})^+$. Otherwise, $M_{k,1}$ can be powered down earlier to save energy usage, which contradicts the optimality of $\mathcal{M}^*$. Since the total demand $s(\mathcal{J}, t)$ exceeds $k-1$ immediately before $Q(M_{k,1})^+$, $Q(M_{k,1})^+$ must fall inside or on the right endpoint of the “on” interval of some machine in $\mathcal{M}_k^{REEJS}$. Suppose that this machine is $M_{k,j}^{REEJS}$ where $j > 1$. Note that by our algorithm, the total demand $s(\mathcal{J}, t)$ is bounded by $k-1$ during the interval $[Q(M_{k,j-1}^{REEJS})^+, Q(M_{k,j}^{REEJS})^-)$ and $Q(M_{k,j}^{REEJS})^- - Q(M_{k,j-1}^{REEJS})^+ > C$. We can thus save energy usage by powering down $M_{k,1}$ earlier at time $Q(M_{k,j-1}^{REEJS})^+$ and powering up a new machine from time $Q(M_{k,j}^{REEJS})^-$ to $Q(M_{k,1})^+$ while keeping the schedule $\mathcal{M}^*$ meeting the capacity requirement. This leads to a contradiction to the optimality of $\mathcal{M}^*$. Similarly, if $Q(M_{k,1})^+ < Q(M_{k,1}^{REEJS})^+$, machine $M_{k,2}$ must be powered up before time $Q(M_{k,1}^{REEJS})^+$ as the total demand $s(\mathcal{J}, t)$ exceeds $k-1$ immediately before time $Q(M_{k,1}^{REEJS})^+$. Since $\mathcal{M}^*$ meets the capacity requirement, the total demand $s(\mathcal{J}, t)$ must be bounded by $k-1$ during the interval $[Q(M_{k,1})^+, Q(M_{k,2})^-)$. Since $[Q(M_{k,1})^+, Q(M_{k,2})^-) \subseteq Q(M_{k,1}^{REEJS})$, according to our algorithm, we have $Q(M_{k,2})^- - Q(M_{k,1})^+ \leq C$. This implies that machines $M_{k,1}$ and $M_{k,2}$ can be replaced by a machine with an “on” interval $[Q(M_{k,1})^-, Q(M_{k,2})^+)$ to save energy usage, which again contradicts the optimality of $\mathcal{M}^*$.¹ Therefore, we have $Q(M_{k,1})^+ = Q(M_{k,1}^{REEJS})^+$ and hence $Q(M_{k,1}) = Q(M_{k,1}^{REEJS})$.

Given $Q(M_{k,1}) = Q(M_{k,1}^{REEJS})$, by similar arguments, we can show that $Q(M_{k,2}) = Q(M_{k,2}^{REEJS})$. Inductively, it can be shown that $Q(M_{k,i}) = Q(M_{k,i}^{REEJS})$ for any i . Therefore, for each k , the machines in $\mathcal{M}_k$ and $\mathcal{M}_k^{REEJS}$ have the same “on” intervals, which implies that the schedules $\mathcal{M}^*$ and $\mathcal{M}^{REEJS}$ have the same energy consumption. $\square$

4.3.2 Offline Setting

In this section, we study the EEJS-Arbitrary problem in the offline setting where the information of all the jobs is known before scheduling. We propose a 5-approximation

¹If $Q(M_{k,2})^- - Q(M_{k,1})^+ = C$, the replacement does not change the energy usage but it extends the power-down time of $M_{k,1}$ to $Q(M_{k,2})^+$. We can then repeat the comparison of the new $Q(M_{k,1})^+$ with $Q(M_{k,1}^{REEJS})^+$ until either the optimality of $\mathcal{M}^*$ is violated or $Q(M_{k,1})^+ = Q(M_{k,1}^{REEJS})^+$.

algorithm which is based on the Dual Coloring algorithm introduced in Section 3.4.2. We start by giving a high level description of the algorithm. Jobs to be scheduled are first classified into two sets $\mathcal{J}^l$ and $\mathcal{J}^s$ according to their sizes, where $\mathcal{J}^l$ ($\mathcal{J}^s$) includes all the large (small) jobs with size larger (no larger) than $\frac{1}{2}$. The large jobs and the small jobs are scheduled separately by our algorithm. We apply our optimal offline algorithm proposed for EEJS-Unit (introduced in Section 4.2) to schedule all the large jobs, since any two large jobs cannot run concurrently on the same machine. The small jobs are scheduled by the following steps. First, a demand chart is built to describe the total size of the active small jobs as a function of time (similar to the top diagram of Figure 4.2). Then, all the small jobs are placed inside the demand chart such that no three jobs overlap together in their placement. Next, the jobs are assigned to the machines by slicing the demand chart. Finally, the machine schedules are derived by a similar algorithm to that for solving the REEJS problem (introduced in Section 4.3.1). Figure 4.3 gives an example to illustrate the scheduling of small jobs. A formal description of the algorithm for scheduling small jobs is given in Algorithm 2.

The placement and slicing steps basically follow the Dual Coloring algorithm introduced in Section 3.4.2. In the placement step, each small job J is represented by a rectangle spanning its active interval $I(J)$ in the time dimension and having a height of its size $s(J)$ in the demand dimension (see Figure 4.3(b)). The placement proceeds from the top to the bottom of the demand chart and carefully controls the overlaps between different jobs to ensure that no area of the demand chart is covered by three or more jobs simultaneously.

In the slicing step, the demand chart is sliced into horizontal stripes of height $\frac{1}{2}$ each. Thanks to the overlap limit, all the small jobs placed completely within each stripe have a total size at most 1 and thus can be assigned to run on one machine at any time. On the other hand, since the size of each small job is bounded by $\frac{1}{2}$, a job can span at most 2 stripes in the placement. For each pair of neighboring stripes, due to the overlap limit, at most two small jobs can cross them concurrently at any time, so their total size is bounded by 1. Therefore, all the jobs placed across a pair of neighboring stripes can be assigned to run on one machine at any time.

The number of stripes produced by the slicing step is at most $\max_t \lceil \frac{s(\mathcal{J}^s, t)}{1/2} \rceil \leq 2 \cdot \max_t \lceil s(\mathcal{J}^s, t) \rceil$. Thus, according to the above analysis, all the small jobs $\mathcal{J}^s$ can be

Algorithm 2: Scheduling small jobs offline in EEJS-Arbitrary

Input: $\mathcal{J}^s = \{J \in \mathcal{J} : s(J) \leq \frac{1}{2}\}$.
Output: A schedule for all the small jobs.

- 1 —**Place the small jobs inside the demand chart**—
- 2 $Altitudes = \{s(\mathcal{J}^s, t) : t \in \bigcup_{J \in \mathcal{J}^s} I(J)\};$
- 3 **while** $Altitudes \neq \emptyset$ **do**
- 4 Determine $h = \max_{h' \in Altitudes} h'$;
- 5 Let $\mathcal{Red}$, $\mathcal{Blue}$ and $\mathcal{Uncolored}$ be the respective sets of all maximal red, blue and uncolored intervals at altitude h ;
- 6 **while** $\mathcal{Uncolored} \neq \emptyset$ **do**
- 7 Pick an uncolored interval $I_u \in \mathcal{Uncolored}$;
- 8 **if** $\exists J \in \mathcal{J}^s$ such that $I(J) \cap I_u \neq \emptyset$ and
- 9 $\forall I \in \mathcal{Uncolored} \cup \mathcal{Red} \setminus \{I_u\}, I(J) \cap I = \emptyset$ **then**
- 10 Place job J at altitude h and delete J from $\mathcal{J}^s$;
- 11 Color the rectangle $\{I(J) \cap I_u\} \times (h - s(J), h]$ red;
- 12 Delete I_u from $\mathcal{Uncolored}$;
- 13 Add $[I_u^-, I(J)^-)$ and $[I(J)^+, I_u^+)$ into $\mathcal{Uncolored}$ if they are not empty;
- 14 Add $h - s(J)$ into $Altitudes$ if $h > s(J)$ and $h - s(J) \notin Altitudes$;
- 15 **else**
- 16 Color the rectangle $I_u \times (0, h]$ blue;
- 17 Delete I_u from $\mathcal{Uncolored}$;
- 18 Delete h from $Altitudes$;
- 19 —**Slice the demand chart and schedule small jobs**—
- 20 Slice the demand chart into horizontal stripes of height $\frac{1}{2}$ each;
- 21 **for each stripe do**
- 22 Schedule all the small jobs placed completely within the stripe by applying the optimal REEJS algorithm;
- 23 **for every two neighboring stripes do**
- 24 Schedule all the small jobs placed across the two stripes by applying the optimal REEJS algorithm;

Figure 4.3: An example for scheduling small jobs in the offline setting

divided into at most $2 \cdot \max_t \lceil s(\mathcal{J}^s, t) \rceil + 2 \cdot \max_t \lceil s(\mathcal{J}^l, t) \rceil - 1 = 4 \cdot \max_t \lceil s(\mathcal{J}^s, t) \rceil - 1$ disjoint groups such that the jobs in each group have a total size bounded by 1 at any time. Therefore, we can apply the optimal REEJS algorithm in Section 4.3.1 to each group of jobs to produce an optimal REEJS schedule in just one iteration (see Figure 4.3(c)). The schedule constructed for each group of jobs contains a set of machines with non-overlapping “on” intervals. The final output of our algorithm includes the machines used in the schedules for all the groups of small jobs as well as the machines used by the optimal EEJS-Unit algorithm to schedule all the large jobs.

Now, we prove that our algorithm achieves an approximation ratio of 5. Let $OPT_{EEJS}(\mathcal{J})$ denote the overall energy consumption of the optimal schedule for processing all the jobs $\mathcal{J}$. It is obvious that the optimal schedule for processing a set of jobs cannot consume less energy than that for processing a subset of the jobs therein. Since $\mathcal{J}^l \subseteq \mathcal{J}$ and $\mathcal{J}^s \subseteq \mathcal{J}$, we have $OPT_{EEJS}(\mathcal{J}^l) \leq OPT_{EEJS}(\mathcal{J})$ and $OPT_{EEJS}(\mathcal{J}^s) \leq OPT_{EEJS}(\mathcal{J})$. Let $A(\mathcal{J})$ denote the overall energy consumption of the schedule produced by applying our algorithm. Since the large jobs and the small jobs are scheduled separately by our algorithm, we have $A(\mathcal{J}) = A(\mathcal{J}^l) + A(\mathcal{J}^s)$. We first bound $A(\mathcal{J}^l)$, the energy usage for processing the large jobs $\mathcal{J}^l$. Since no two large jobs can run simultaneously on the same machine, by applying the optimal EEJS-Unit algorithm, we have

$$A(\mathcal{J}^l) = OPT_{EEJS}(\mathcal{J}^l) \leq OPT_{EEJS}(\mathcal{J}).$$

Next, we bound the energy usage for processing the small jobs $\mathcal{J}^s$. If we apply the optimal REEJS algorithm in Section 4.3.1 to schedule all the small jobs $\mathcal{J}^s$, the machines used would consist of $\max_t \lceil s(\mathcal{J}^s, t) \rceil$ disjoint sets $\mathcal{M}_1^{REEJS}, \mathcal{M}_2^{REEJS}, \dots$ where each set $\mathcal{M}_k^{REEJS}$ denotes the machines of level k . Let $OPT_{REEJS}(\mathcal{J}^s)$ denote the energy consumption of the optimal REEJS schedule for $\mathcal{J}^s$, and $E(\mathcal{M}_k^{REEJS})$ denote the total energy consumption of the machines in $\mathcal{M}_k^{REEJS}$. Then, $OPT_{REEJS}(\mathcal{J}^s) = \sum_{k=1}^{\max_t \lceil s(\mathcal{J}^s, t) \rceil} E(\mathcal{M}_k^{REEJS})$. Note that the machines $\mathcal{M}_k^{REEJS}$ actually serve the job demands between altitudes $k - 1$ and k in the demand chart.

On the other hand, by applying our approximation algorithm, all the machines used to process the small jobs can be divided into at most $4 \cdot \max_t \lceil s(\mathcal{J}^s, t) \rceil - 1$ disjoint sets, among which there is one set for processing the jobs placed completely within each

stripe of the demand chart and one set for processing the jobs placed across each pair of neighboring stripes. By the construction of the demand chart, the set of machines for the jobs placed in the stripe from altitude $(h-1) \cdot \frac{1}{2}$ to $h \cdot \frac{1}{2}$ only needs to be “on” when the total size of active small jobs exceeds $(h-1) \cdot \frac{1}{2} \geq \lceil \frac{h}{2} \rceil - 1$. Thus, the “on” intervals of these machines must be completely contained in the “on” intervals of the machines $\mathcal{M}_{\lceil h/2 \rceil}^{REEJS}$. This is true before as well as after the “on” intervals are merged by the optimal REEJS algorithm. Similarly, the set of machines for the jobs placed across the stripes from altitude $(h-2) \cdot \frac{1}{2}$ to $(h-1) \cdot \frac{1}{2}$ and from $(h-1) \cdot \frac{1}{2}$ to $h \cdot \frac{1}{2}$ only needs to be “on” when the total size of active small jobs exceeds $(h-1) \cdot \frac{1}{2} \geq \lceil \frac{h}{2} \rceil - 1$. So, they must also have their “on” intervals fully contained in the “on” intervals of the machines $\mathcal{M}_{\lceil h/2 \rceil}^{REEJS}$. Therefore, the total energy usage of all the machines for small jobs by our algorithm is bounded by

$$\begin{aligned}
 & \sum_{h=1}^{2 \cdot \max_t \lceil s(\mathcal{J}^s, t) \rceil} E(\mathcal{M}_{\lceil h/2 \rceil}^{REEJS}) + \sum_{h=2}^{2 \cdot \max_t \lceil s(\mathcal{J}^s, t) \rceil} E(\mathcal{M}_{\lceil h/2 \rceil}^{REEJS}) \\
 &= 2 \cdot \sum_{k=1}^{\max_t \lceil s(\mathcal{J}^s, t) \rceil} E(\mathcal{M}_k^{REEJS}) + E(\mathcal{M}_1^{REEJS}) + 2 \cdot \sum_{k=2}^{\max_t \lceil s(\mathcal{J}^s, t) \rceil} E(\mathcal{M}_k^{REEJS}) \\
 &< 4 \cdot \sum_{k=1}^{\max_t \lceil s(\mathcal{J}^s, t) \rceil} E(\mathcal{M}_k^{REEJS}) \\
 &= 4 \cdot OPT_{REEJS}(\mathcal{J}^s).
 \end{aligned}$$

Recall that the REEJS problem is a relaxation of the EEJS-Arbitrary problem. It follows that

$$OPT_{REEJS}(\mathcal{J}^s) \leq OPT_{EEJS}(\mathcal{J}^s) \leq OPT_{EEJS}(\mathcal{J}).$$

As a result, $A(\mathcal{J}^s) \leq 4 \cdot OPT_{EEJS}(\mathcal{J})$ and $A(\mathcal{J}) = A(\mathcal{J}^l) + A(\mathcal{J}^s) \leq 5 \cdot OPT_{EEJS}(\mathcal{J})$.

Theorem 4.2. *For the Energy-Efficient Job Scheduling problem in the offline setting, the algorithm introduced above achieves an approximation ratio of 5.*

4.3.3 Non-Clairvoyant Online Setting

Next, we consider the online version of the EEJS-Arbitrary problem, where each job must be placed on a machine at its arrival without knowing any future job arrivals.

Figure 4.4: An example for scheduling small jobs by Modified First Fit

Again, there are two flavours for the online version: non-clairvoyant and clairvoyant. In the non-clairvoyant online setting, the processing length of each job is not revealed at its arrival and thus cannot be used for scheduling purposes. In the clairvoyant online setting, the processing length of each job is known at its arrival and can be used for scheduling purposes. We first study the EEJS-Arbitrary problem in the non-clairvoyant online setting. It has been shown in [55] that no deterministic online algorithm can achieve a competitive ratio less than μ for the MinUsageTime DBP problem in the non-clairvoyant online setting, where μ represents the max/min item duration (job processing length) ratio. This problem can be seen as a special case of our EEJS-Arbitrary problem by assuming the initial energy cost C to be 0 for powering a machine up. Thus, the lower bound μ also applies to the competitiveness achievable for EEJS-Arbitrary. In the following, we propose a $(\mu + 9)$ -competitive Modified First Fit algorithm for EEJS-Arbitrary in the non-clairvoyant online setting.

Similar to the 5-approximation algorithm in the offline setting, the Modified First Fit algorithm also schedules the large jobs $\mathcal{J}^l$ (of size larger than $\frac{1}{2}$) and the small jobs $\mathcal{J}^s$ (of

size at most $\frac{1}{2}$) separately. The large jobs are scheduled by applying the 2-competitive online algorithm designed for EEJS-Unit (introduced in Section 4.2), while the small jobs are scheduled based on the idea of the First-Fit strategy. Specifically, each machine used for processing small jobs is given a layer label. The machines given the label k are referred to as the layer- k machines. At any time, at most one machine of each layer can be in the “on” state. To schedule an incoming job J , we find the lowest-indexed layer k such that either (1) no layer- k machine is in the “on” state or (2) the “on” machine of layer k has enough capacity available to accommodate job J . In the former case, a new layer- k machine is powered up to run job J . In the latter case, the incoming job J is assigned to the “on” machine of layer k . Whenever an “on” machine for small jobs becomes idle, the machine waits for at most C time units for receiving new incoming jobs. If no incoming job is placed on it after C time units, the machine is powered down. Figure 4.4 shows an example of how Modified First Fit schedules small jobs.

Now, we prove that our Modified First Fit algorithm is $(\mu + 9)$ -competitive. Let $MFF(\mathcal{J})$ denote the overall energy consumption for processing all the jobs $\mathcal{J}$ by applying our Modified First Fit algorithm. Then, $MFF(\mathcal{J}) = MFF(\mathcal{J}^l) + MFF(\mathcal{J}^s)$. We first consider $MFF(\mathcal{J}^l)$. Since any two large jobs cannot run concurrently on the same machine, by applying the 2-competitive online algorithm designed for EEJS-Unit, we have

$$MFF(\mathcal{J}^l) \leq 2 \cdot OPT_{EEJS}(\mathcal{J}^l) \leq 2 \cdot OPT_{EEJS}(\mathcal{J}).$$

Next, we bound $MFF(\mathcal{J}^s)$. We split the energy consumption by all the machines used for $\mathcal{J}^s$ into two parts: the energy usage when the machines are busy (referred to as the busy energy usage) and the energy usage for powering up the machines and when they are idle (referred to as the auxiliary energy usage). We show that the busy energy usage of all the machines used for $\mathcal{J}^s$ is no more than

$$(\mu + 3) \cdot \int_{\bigcup_{J \in \mathcal{J}^s} I(J)} \lceil s(\mathcal{J}^s, t) \rceil \, dt.$$

Since the number of “on” machines for processing the small jobs $\mathcal{J}^s$ must be at least $\lceil s(\mathcal{J}^s, t) \rceil$ at any time t , $\int_{\bigcup_{J \in \mathcal{J}^s} I(J)} \lceil s(\mathcal{J}^s, t) \rceil \, dt$ is a lower bound on the “on” time and hence the energy consumption of the optimal EEJS schedule for $\mathcal{J}^s$. We also prove that

the auxiliary energy usage of all the machines used for $\mathcal{J}^s$ is no more than four times the energy consumption of the optimal REEJS schedule for $\mathcal{J}^s$, i.e., $OPT_{REEJS}(\mathcal{J}^s)$. Thus, $MFF(\mathcal{J}^s)$ is bounded by

$$\begin{aligned}
 & (\mu + 3) \cdot OPT_{EEJS}(\mathcal{J}^s) + 4 \cdot OPT_{REEJS}(\mathcal{J}^s) \\
 & \leq (\mu + 3) \cdot OPT_{EEJS}(\mathcal{J}^s) + 4 \cdot OPT_{EEJS}(\mathcal{J}^s) \\
 & = (\mu + 7) \cdot OPT_{EEJS}(\mathcal{J}^s) \\
 & \leq (\mu + 7) \cdot OPT_{EEJS}(\mathcal{J}).
 \end{aligned}$$

Combining $MFF(\mathcal{J}^l)$ and $MFF(\mathcal{J}^s)$, we can then conclude that the Modified First Fit algorithm is $(\mu + 9)$ -competitive.

Bounding the busy energy usage of machines used for $\mathcal{J}^s$: We apply the charging technique similar to that introduced in Section 3.2.2 to bound the total busy time of the machines used for small jobs. It is easy to bound the total busy time of the layer-1 machines $\mathcal{M}_1^{MFFs}$ by $\int_{\bigcup_{J \in \mathcal{J}^s} I(J)} 1 \, dt \leq \int_{\bigcup_{J \in \mathcal{J}^s} I(J)} \lceil s(\mathcal{J}^s, t) \rceil \, dt$, since at most one machine in $\mathcal{M}_1^{MFFs}$ is busy at any moment when there are small jobs active. In what follows, we focus on bounding the total busy time of the layer- k machines ($k \geq 2$). Let $\mathcal{J}_k^{MFFs}$ be the set of small jobs placed on the layer- k machines. By the Modified First-Fit algorithm, the busy time of the machines $\mathcal{M}_k^{MFFs}$ is given by $\text{len}(\bigcup_{J \in \mathcal{J}_k^{MFFs}} I(J))$. Let $\mathcal{J}_k$ be a maximal subset of $\mathcal{J}_k^{MFFs}$ by deleting any job $J \in \mathcal{J}_k^{MFFs}$ whose active interval is fully contained in another job's active interval. Then, by sorting the jobs in $\mathcal{J}_k$ in the order of their arrival times, i.e., $\mathcal{J}_k = \{J_{k,1}, J_{k,2}, \dots, J_{k,z_k}\}$ where $I(J_{k,1})^- < I(J_{k,2})^- < \dots < I(J_{k,z_k})^-$ and $z_k = |\mathcal{J}_k|$, we have $I(J_{k,1})^+ < I(J_{k,2})^+ < \dots < I(J_{k,z_k})^+$. Otherwise, if $I(J_{k,i})^+ \geq I(J_{k,j})^+$ for some $i < j$, the active interval of $J_{k,j}$ is inside the active interval of $J_{k,i}$, which contradicts the definition of $\mathcal{J}_k$. Besides, we also have $\bigcup_{J \in \mathcal{J}_k^{MFFs}} I(J) = \bigcup_{J \in \mathcal{J}_k} I(J)$ after reducing $\mathcal{J}_k^{MFFs}$ to $\mathcal{J}_k$.

We split $\bigcup_{J \in \mathcal{J}_k} I(J)$ into z_k disjoint intervals. Specifically, for each job $J_{k,i} \in \mathcal{J}_k$ ($1 \leq i \leq z_k - 1$), define

$$X(J_{k,i}) = [I(J_{k,i})^-, \min\{I(J_{k,i})^+, I(J_{k,i+1})^-\}).$$

For the job $J_{k,z_k} \in \mathcal{J}_k$, define

$$X(J_{k,z_k}) = I(J_{k,z_k}).$$

Then, the total busy time of the layer- k machines can be rewritten as

$$\begin{aligned} \text{len} \left(\bigcup_{J \in \mathcal{J}_k^{MFFs}} I(J) \right) &= \text{len} \left(\bigcup_{J \in \mathcal{J}_k} I(J) \right) \\ &= \sum_{J \in \mathcal{J}_k} \text{len}(X(J)). \end{aligned}$$

Now, we define

$$d_k = \sum_{J \in \mathcal{J}_k} \left(s(J) \cdot \text{len}(X(J)) \right).$$

It is easy to see that

$$\begin{aligned} d_k &\leq \sum_{J \in \mathcal{J}_k} \left(s(J) \cdot \text{len}(I(J)) \right) \\ &\leq \sum_{J \in \mathcal{J}_k^{MFFs}} \left(s(J) \cdot \text{len}(I(J)) \right). \end{aligned} \tag{4.1}$$

Recall that each job J in $\mathcal{J}_k$ is placed on a machine in $\mathcal{M}_k^{MFFs}$. According to the Modified First-Fit algorithm, when job J arrives at time $X(J)^- = I(J)^-$, there must exist a layer- $(k-1)$ machine $M \in \mathcal{M}_{k-1}^{MFFs}$ which is busy and does not have enough capacity available to accommodate job J . Otherwise, J would have been placed on a machine in $\mathcal{M}_{k-1}^{MFFs}$. Therefore, at time $X(J)^-$, the total size of the active jobs on machine M must be larger than $1 - s(J)$. We use $\mathcal{P}(J)$ to denote these jobs and we have

$$s(J) + \sum_{\hat{J} \in \mathcal{P}(J)} s(\hat{J}) > 1.$$

Define

$$d_k^* = \sum_{J \in \mathcal{J}_k} \sum_{\hat{J} \in \mathcal{P}(J)} \left(s(\hat{J}) \cdot \text{len}(X(J)) \right).$$

Then, the total busy time of the machines $\mathcal{M}_k^{MFFs}$ can be bounded by

$$\begin{aligned} \sum_{J \in \mathcal{J}_k} \text{len}(X(J)) &< \sum_{J \in \mathcal{J}_k} \left(\left(\sum_{\hat{J} \in \mathcal{P}(J)} s(\hat{J}) + s(J) \right) \cdot \text{len}(X(J)) \right) \\ &= d_k + d_k^*. \end{aligned} \tag{4.2}$$

We next show that d_k^* is bounded by $(\mu + 1) \cdot \sum_{J \in \mathcal{J}_{k-1}^{MFFs}} (s(J) \cdot \text{len}(I(J)))$. Define

$$\widehat{\mathcal{J}}_k = \bigcup_{J \in \mathcal{J}_k} \mathcal{P}(J).$$

Obviously, $\widehat{\mathcal{J}}_k \subseteq \mathcal{J}_{k-1}^{MFFs}$. For each job $\hat{J} \in \widehat{\mathcal{J}}_k$, let $\mathcal{P}^{-1}(\hat{J})$ denote the set of all the jobs J in $\mathcal{J}_k$ such that $\hat{J} \in \mathcal{P}(J)$. Note that the jobs in $\mathcal{P}^{-1}(\hat{J})$ may come from different machines in $\mathcal{M}_k^{MFFs}$. For each job $J \in \mathcal{P}^{-1}(\hat{J})$, since job $\hat{J}$ has already been placed on a machine in $\mathcal{M}_{k-1}^{MFFs}$ when J arrives, $\hat{J}$ cannot arrive later than J . Thus,

$$X(J)^- = I(J)^- \geq I(\hat{J})^-. \quad (4.3)$$

By definition, $I(\hat{J})$ covers time $X(J)^-$, i.e., $X(J)^- < I(\hat{J})^+$. Recall that the max/min job processing length ratio is μ , which suggests that

$$\text{len}(X(J)) \leq \text{len}(I(J)) \leq \mu \cdot \text{len}(I(\hat{J})).$$

Therefore,

$$\begin{aligned} X(J)^+ &= X(J)^- + \text{len}(X(J)) \\ &< I(\hat{J})^+ + \mu \cdot \text{len}(I(\hat{J})). \end{aligned} \quad (4.4)$$

Remark that $X(J)$ of all the jobs J in $\mathcal{P}^{-1}(\hat{J}) \subseteq \mathcal{J}_k$ are disjoint. It follows from (4.3) and (4.4) that

$$\begin{aligned} \sum_{J \in \mathcal{P}^{-1}(\hat{J})} \text{len}(X(J)) &\leq \max_{J \in \mathcal{P}^{-1}(\hat{J})} X(J)^+ - \min_{J \in \mathcal{P}^{-1}(\hat{J})} X(J)^- \\ &\leq I(\hat{J})^+ + \mu \cdot \text{len}(I(\hat{J})) - I(\hat{J})^- \\ &= (\mu + 1) \cdot \text{len}(I(\hat{J})). \end{aligned}$$

As a result, we have

$$\begin{aligned} d_k^* &= \sum_{J \in \mathcal{J}_k} \sum_{\hat{J} \in \mathcal{P}(J)} (s(\hat{J}) \cdot \text{len}(X(J))) \\ &= \sum_{\hat{J} \in \widehat{\mathcal{J}}_k} s(\hat{J}) \cdot \left(\sum_{J \in \mathcal{P}^{-1}(\hat{J})} \text{len}(X(J)) \right) \\ &\leq \sum_{\hat{J} \in \widehat{\mathcal{J}}_k} (s(\hat{J}) \cdot (\mu + 1) \cdot \text{len}(I(\hat{J}))) \\ &\leq (\mu + 1) \cdot \sum_{J \in \mathcal{J}_{k-1}^{MFFs}} (s(J) \cdot \text{len}(I(J))). \end{aligned}$$

By (4.1) and (4.2), we can bound the total busy time of the machines $\mathcal{M}_k^{MFFs}$ by

$$d_k + d_k^* \leq \sum_{J \in \mathcal{J}_k^{MFFs}} \left(s(J) \cdot \text{len}(I(J)) \right) + (\mu + 1) \cdot \sum_{J \in \mathcal{J}_{k-1}^{MFFs}} \left(s(J) \cdot \text{len}(I(J)) \right)$$

and thus bound the total busy time of the machines $\bigcup_{k=2}^m \mathcal{M}_k^{MFFs}$ by

$$\begin{aligned} & \sum_{k=2}^m \left(d_k + d_k^* \right) \\ & \leq \sum_{k=2}^m \sum_{J \in \mathcal{J}_k^{MFFs}} \left(s(J) \cdot \text{len}(I(J)) \right) + \sum_{k=2}^m \left((\mu + 1) \cdot \sum_{J \in \mathcal{J}_{k-1}^{MFFs}} \left(s(J) \cdot \text{len}(I(J)) \right) \right) \\ & \leq \sum_{J \in \mathcal{J}^s} \left(s(J) \cdot \text{len}(I(J)) \right) + (\mu + 1) \cdot \sum_{J \in \mathcal{J}^s} \left(s(J) \cdot \text{len}(I(J)) \right) \\ & = (\mu + 2) \cdot \sum_{J \in \mathcal{J}^s} \left(s(J) \cdot \text{len}(I(J)) \right) \\ & = (\mu + 2) \cdot \int_{\bigcup_{J \in \mathcal{J}^s} I(J)} s(\mathcal{J}^s, t) \, dt \\ & \leq (\mu + 2) \cdot \int_{\bigcup_{J \in \mathcal{J}^s} I(J)} \lceil s(\mathcal{J}^s, t) \rceil \, dt. \end{aligned}$$

Combining with the total busy time of the machines $\mathcal{M}_1^{MFFs}$, we can bound the busy energy usage for processing all the small jobs by

$$(\mu + 3) \cdot \int_{\bigcup_{J \in \mathcal{J}^s} I(J)} \lceil s(\mathcal{J}^s, t) \rceil \, dt.$$

Bounding the auxiliary energy usage of machines used for $\mathcal{J}^s$: We first analyze the auxiliary energy usage by the machines of a given layer k . Recall that at most one machine of each layer can be “on” at any time. Thus, the busy intervals of the machines $\mathcal{M}_k^{MFFs}$ must not overlap with each other. Let $I_{k,1}, I_{k,2}, \dots, I_{k,n_k}$ denote all the busy intervals of the machines $\mathcal{M}_k^{MFFs}$ in the chronological order. Then, an initial energy cost C must be incurred at time $I_{k,1}^-$ when the first machine of layer k is powered up. According to our Modified First Fit algorithm, for each $1 \leq i \leq n_k - 1$, if $I_{k,i+1}^- - I_{k,i}^+ \leq C$, then the two consecutive busy intervals $I_{k,i}$ and $I_{k,i+1}$ must be associated with the same machine of layer k and this machine must be idle during the interval $[I_{k,i}^+, I_{k,i+1}^-]$. Consequently, an energy usage of $I_{k,i+1}^- - I_{k,i}^+$ is incurred over the

idle interval. Otherwise, if $I_{k,i+1}^- - I_{k,i}^+ > C$, a machine of layer k must be powered down at time $I_{k,i}^+ + C$ and a new machine of layer k must be powered up at time $I_{k,i+1}^-$. This suggests that a total energy usage of $2C$ is incurred by these two machines during the interval $[I_{k,i}^+, I_{k,i+1}^-)$. Finally, note that the last machine of layer k must be idle during the interval $[I_{k,n_k}^+, I_{k,n_k}^+ + C)$ which incurs an energy usage of C . Therefore, the total auxiliary energy usage of the machines $\mathcal{M}_k^{MFFs}$ is at most $2 \cdot (C + \sum_{i=1}^{n_k-1} \min\{I_{k,i+1}^- - I_{k,i}^+, C\})$. In the following, we show that $C + \sum_{i=1}^{n_k-1} \min\{I_{k,i+1}^- - I_{k,i}^+, C\}$ is no more than the energy consumption by the machines of level $\lceil k/2 \rceil$ in the optimal REEJS schedule for $\mathcal{J}^s$. In this way, the auxiliary energy usage of all the machines for $\mathcal{J}^s$ by applying our Modified First Fit algorithm can be bounded by $4 \cdot OPT_{REEJS}(\mathcal{J}^s)$.

First, we show that if a machine of layer k is used by our Modified First Fit algorithm, then at least $\lceil \frac{k}{2} \rceil$ levels of machines are needed in the optimal REEJS schedule for $\mathcal{J}^s$, i.e., level $\lceil \frac{k}{2} \rceil$ exists. In fact, if a small job J is placed on a machine in $\mathcal{M}_k^{MFFs}$, then for each layer lower than k , there must exist a machine which is in the “on” state at J ’s arrival time $I(J)^-$ but does not have enough capacity available to accommodate J . This implies that the “on” machine of each layer lower than k must have more than $1 - s(J)$ capacity occupied at time $I(J)^-$. Consequently, the total size of the small jobs active at time $I(J)^-$ must be larger than $(k-1)(1-s(J)) + s(J) = k-1 - (k-2) \cdot s(J) \geq \frac{k}{2}$, where the inequality follows from the fact that $s(J) \leq \frac{1}{2}$. Therefore, in the optimal REEJS schedule for $\mathcal{J}^s$, at least $\lceil \frac{k}{2} \rceil$ machines need to be “on” to satisfy the job demands at time $I(J)^-$. As a result, there are at least $\lceil \frac{k}{2} \rceil$ levels of machines in the optimal REEJS schedule for $\mathcal{J}^s$.

Let $\mathcal{M}_{\lceil k/2 \rceil}^{REEJS}$ denote the machines of level $\lceil \frac{k}{2} \rceil$ in the optimal REEJS schedule for $\mathcal{J}^s$. The above analysis suggests that a machine in $\mathcal{M}_{\lceil k/2 \rceil}^{REEJS}$ must be in the “on” state at the arrival of every small job placed on a machine of layer k by our Modified First Fit algorithm. It is obvious that there is a small job arriving at the beginning of each busy interval $I_{k,i}$ of the machines in $\mathcal{M}_k^{MFFs}$. Thus, a machine in $\mathcal{M}_{\lceil k/2 \rceil}^{REEJS}$ must be “on” at times $I_{k,1}^-, I_{k,2}^-, \dots, I_{k,n_k}^-$. In order for a machine to be powered up at time $I_{k,1}^-$, an initial energy cost C must be incurred. For each interval $(I_{k,i}^-, I_{k,i+1}^-]$ (where $1 \leq i \leq n_k - 1$), if different machines in $\mathcal{M}_{\lceil k/2 \rceil}^{REEJS}$ are “on” at times $I_{k,i}^-$ and $I_{k,i+1}^-$ respectively, at least an initial energy cost C is incurred in $(I_{k,i}^-, I_{k,i+1}^-]$. Otherwise, suppose the same machine in

$\mathcal{M}_{\lceil k/2 \rceil}^{REEJS}$ is “on” at times $I_{k,i}^-$ and $I_{k,i+1}^-$. If $I_{k,i+1}^- - I_{k,i}^- \leq C$, the minimum possible energy consumption by the machine during $(I_{k,i}^-, I_{k,i+1}^-]$ is at least $I_{k,i+1}^- - I_{k,i}^-$, i.e., to keep the machine “on” throughout the interval, since powering the machine down and up would use more energy. If $I_{k,i+1}^- - I_{k,i}^- > C$, the minimum possible energy consumption by the machine during $(I_{k,i}^-, I_{k,i+1}^-]$ is at least an initial energy cost C , since it consumes more energy to keep the machine “on” throughout the interval. Therefore, the total energy consumed by the machines in $\mathcal{M}_{\lceil k/2 \rceil}^{REEJS}$ is at least $C + \sum_{i=1}^{n_k-1} \min\{I_{k,i+1}^- - I_{k,i}^-, C\} \geq C + \sum_{i=1}^{n_k-1} \min\{I_{k,i+1}^- - I_{k,i}^+, C\}$. As a result, the total auxiliary energy usage of the machines $\mathcal{M}_k^{MFFs}$ is at most twice the energy consumption of the machines $\mathcal{M}_{\lceil k/2 \rceil}^{REEJS}$. Thus, the auxiliary energy usage of all the machines for $\mathcal{J}^s$ by our Modified First Fit algorithm is bounded by $4 \cdot \sum_{k=1}^{\max_t \lceil s(\mathcal{J}^s, t) \rceil} E(\mathcal{M}_k^{REEJS}) = 4 \cdot OPT_{REEJS}(\mathcal{J}^s)$.

Theorem 4.3. *For the Energy-Efficient Job Scheduling problem in the non-clairvoyant online setting, the Modified First Fit algorithm achieves a competitive ratio of $\mu + 9$, where μ is the max/min job processing length ratio among all the jobs to schedule.*

Next, we prove that our Modified First Fit algorithm is $O(1)$ -competitive for a special case where each job to be scheduled has a processing length no more than C . Recall that the total energy usage for processing the large jobs is bounded by $2 \cdot OPT_{EEJS}(\mathcal{J}^l)$ and the auxiliary energy usage for processing the small jobs is bounded by $4 \cdot OPT_{REEJS}(\mathcal{J}^s)$. We thus only need to bound the total busy time (busy energy usage) of the machines used for $\mathcal{J}^s$ in this special case. In the following, we prove that the busy energy usage for processing $\mathcal{J}^s$ is bounded by $2 \cdot OPT_{REEJS}(\mathcal{J}^s)$.

Consider the machines $\mathcal{M}_k^{MFFs}$ of a layer k by our Modified First Fit algorithm. Let $\mathcal{J}_k^{MFFs}$ be the set of small jobs processed by the machines $\mathcal{M}_k^{MFFs}$. Then, all the busy intervals of the machines $\mathcal{M}_k^{MFFs}$ are given by the union of the active intervals of the jobs $\mathcal{J}_k^{MFFs}$, i.e., $\bigcup_{i=1}^{n_k} I_{k,i} = \bigcup_{J \in \mathcal{J}_k^{MFFs}} I(J)$. If each job has a processing length at most C , we have $\bigcup_{J \in \mathcal{J}_k^{MFFs}} I(J) \subseteq \bigcup_{J \in \mathcal{J}_k^{MFFs}} [I(J)^-, I(J)^- + C)$. As a result,

$$\bigcup_{i=1}^{n_k} I_{k,i} \subseteq \bigcup_{J \in \mathcal{J}_k^{MFFs}} [I(J)^-, I(J)^- + C). \quad (4.5)$$

Now, consider the machines $\mathcal{M}_{\lceil k/2 \rceil}^{REEJS}$ of level $\lceil \frac{k}{2} \rceil$ in the optimal REEJS schedule for $\mathcal{J}^s$. It has been shown by the former analysis that a machine in $\mathcal{M}_{\lceil k/2 \rceil}^{REEJS}$ must be in

the “on” state at the arrival of every small job placed on a machine of $\mathcal{M}_k^{MFFs}$ by our Modified First Fit algorithm. Thus, $\forall J \in \mathcal{J}_k^{MFFs}$, $I(J)^- \in \bigcup_{M \in \mathcal{M}_{\lceil k/2 \rceil}^{REEJS}} Q(M)$, where $Q(M)$ denotes the “on” interval of a machine M . As a result, if we extend the “on” interval of every machine in $\mathcal{M}_{\lceil k/2 \rceil}^{REEJS}$ by length C , all the intervals $[I(J)^-, I(J)^- + C)$ would be covered, i.e., $\forall J \in \mathcal{J}_k^{MFFs}$,

$$[I(J)^-, I(J)^- + C) \subseteq \bigcup_{M \in \mathcal{M}_{\lceil k/2 \rceil}^{REEJS}} [Q(M)^-, Q(M)^+ + C). \quad (4.6)$$

Combining (4.5) and (4.6), we have

$$\bigcup_{i=1}^{n_k} I_{k,i} \subseteq \bigcup_{M \in \mathcal{M}_{\lceil k/2 \rceil}^{REEJS}} [Q(M)^-, Q(M)^+ + C).$$

Note that all the busy intervals of the machines $\mathcal{M}_k^{MFFs}$ are disjoint. Therefore, the total busy time of the machines $\mathcal{M}_k^{MFFs}$ is bounded by

$$\begin{aligned} \sum_{i=1}^{n_k} \text{len}(I_{k,i}) &= \text{len} \left(\bigcup_{i=1}^{n_k} I_{k,i} \right) \\ &\leq \text{len} \left(\bigcup_{M \in \mathcal{M}_{\lceil k/2 \rceil}^{REEJS}} [Q(M)^-, Q(M)^+ + C) \right) \\ &\leq \sum_{M \in \mathcal{M}_{\lceil k/2 \rceil}^{REEJS}} (\text{len}(Q(M)) + C) \\ &= C \cdot |\mathcal{M}_{\lceil k/2 \rceil}^{REEJS}| + \sum_{M \in \mathcal{M}_{\lceil k/2 \rceil}^{REEJS}} \text{len}(Q(M)). \end{aligned}$$

Note that the right end of the above inequality is exactly the energy consumption of the machines $\mathcal{M}_{\lceil k/2 \rceil}^{REEJS}$ (after the simplification presented in Section 4.1.2). By adding up the above inequality for all the layers of machines, the total busy time of the machines used for $\mathcal{J}^s$ by our Modified First Fit algorithm is bounded by $2 \cdot \text{OPT}_{REEJS}(\mathcal{J}^s)$. Hence, we have the following result.

Theorem 4.4. *For the Energy-Efficient Job Scheduling problem in the non-clairvoyant online setting, the Modified First Fit algorithm is 8-competitive when each job has a processing length at most C .*

4.3.4 Clairvoyant Online Setting

Finally, we consider the EEJS-Arbitrary problem in the clairvoyant online setting, where the processing lengths of jobs are known at their arrivals. Following our work described in Chapter 3, Azar and Vainstein [12] recently proposed a $O(\sqrt{\log \mu})$ -competitive Hybrid algorithm for the MinUsageTime DBP problem in the clairvoyant online setting and established a matching lower bound of $\Omega(\sqrt{\log \mu})$ on the competitive ratio of any deterministic online algorithm. The lower bound of $\Omega(\sqrt{\log \mu})$ also applies to EEJS-Arbitrary, which generalizes MinUsageTime DBP. In this section, we introduce a $O(\sqrt{\log \mu})$ -competitive algorithm for EEJS-Arbitrary in the clairvoyant online setting based on our Modified First Fit algorithm proposed in Section 4.3.3 and the Hybrid algorithm by Azar and Vainstein [12].

The algorithm works by scheduling jobs shorter and longer than C separately. If an incoming job J has a processing length less than C , i.e., $\text{len}(I(J)) \leq C$, we apply our Modified First Fit algorithm to schedule the job. Otherwise, the Hybrid algorithm is applied to schedule the job. The Hybrid algorithm defines two types of bins (general bins and classify-by-duration bins) for placing items in MinUsageTime DBP. For both bin types, an open bin is closed once it becomes empty. In our context of job scheduling, this implies that whenever a machine used for processing the jobs by the Hybrid algorithm becomes idle, we power down the machine immediately. Thus, the machines used for processing jobs longer than C are busy throughout their “on” intervals.

Now, we show that the above algorithm is $O(\sqrt{\log \mu})$ -competitive. Let $\mathcal{J}^M$ and $\mathcal{J}^H$ denote the sets of jobs scheduled by the Modified First Fit algorithm and the Hybrid algorithm respectively. According to Theorem 4.4, the total energy usage of the machines used for processing $\mathcal{J}^M$ is bounded by $8 \cdot \text{OPT}_{EEJS}(\mathcal{J}^M)$. On the other hand, each job in $\mathcal{J}^H$ has a processing length longer than C . By applying the Hybrid algorithm, any machine M used for processing the jobs $\mathcal{J}^H$ must keep busy for at least C time units and thus have an “on” interval at least C long. Therefore, the energy usage of machine M is bounded by $C + \text{len}(Q(M)) \leq 2 \cdot \text{len}(Q(M))$, where $Q(M)$ denotes the “on” interval of M . This suggests that the total energy usage of the machines used for processing $\mathcal{J}^H$ is bounded by twice their total busy time. Based on the competitiveness of the Hybrid algorithm [12], the total busy time of the machines used for $\mathcal{J}^H$ is bounded by

$O(\sqrt{\log \mu}) \cdot OPT_{BUSY}(\mathcal{J}^H)$, where $OPT_{BUSY}(\mathcal{J}^H)$ is the minimum possible busy time of machines for processing the jobs $\mathcal{J}^H$. Obviously, $OPT_{BUSY}(\mathcal{J}^H) \leq OPT_{EEJS}(\mathcal{J}^H)$. It follows that the total busy time of the machines used for $\mathcal{J}^H$ is bounded by $O(\sqrt{\log \mu}) \cdot OPT_{EEJS}(\mathcal{J}^H)$. Thus, we can bound the total energy usage for processing all the jobs $\mathcal{J}$ by

$$8 \cdot OPT_{EEJS}(\mathcal{J}^M) + O(\sqrt{\log \mu}) \cdot OPT_{EEJS}(\mathcal{J}^H) \leq O(\sqrt{\log \mu}) \cdot OPT_{EEJS}(\mathcal{J}).$$

Theorem 4.5. *For the Energy-Efficient Job Scheduling problem in the clairvoyant online setting, the algorithm introduced above is $O(\sqrt{\log \mu})$ -competitive, where μ is the max/min job processing length ratio among all the jobs to schedule.*

4.4 Summary

In this chapter, we have studied the Energy-Efficient Job Scheduling problem, which can be seen as a generalization of the MinUsageTime DBP problem.

We first considered a special case of the problem where each job to be scheduled has a unit size equal to the machine capacity. We have proposed an optimal offline algorithm and a 2-competitive online algorithm to solve this special case, where 2 is proved to be the best achievable competitive ratio by any deterministic online algorithm.

Next, we studied the problem in the general case where each job can have an arbitrary size. We have established a non-trivial lower bound on the optimal solution. Based on this lower bound, we have developed a 5-approximation algorithm and a $O(\mu)$ -competitive Modified First Fit algorithm in the offline and non-clairvoyant online settings respectively. We have also shown that the Modified First Fit algorithm can be adopted to further construct a $O(\sqrt{\log \mu})$ -competitive online algorithm in the clairvoyant online setting.

Our results indicates that the competitiveness bounds for the problem are dependent on the max/min job processing length ratio only and do not depend on the ratio between the base and variable power usage rates or between the power usage rates and the energy overheads for powering machines up and down.

Chapter 5

The Flexible Job Scheduling Problem

In this chapter, we study a *Flexible Job Scheduling* problem defined as follows. We are given a set of jobs, each having an arrival time, a starting deadline and a processing length. Each job has to be started by the scheduler (algorithm) between its arrival and its starting deadline. Once started, the job runs for a period of its processing length without interruption. The target is to minimize the span of all the jobs — the time duration in which at least one job is running.

The Flexible Job Scheduling problem is motivated by generalizing the MinUsageTime DBP problem to model flexible jobs. Note that each job (item) has fixed arrival time and departure time in MinUsageTime DBP, such that it has to be processed in a rigid time window without interruption. However, in some real situations, not all the jobs have to be started immediately at their arrivals. In contrast, on arriving at time t , a job is generally given a laxity l such that it can be started at any time during the time interval $[t, t + l]$. Here, the laxity represents the urgent level of the job. An urgent job is given a small laxity or even zero laxity for its processing, while a trivial job may be given a large laxity and thus can be delayed for a long time. To address the laxity, we consider the following generalized MinUsageTime DBP problem: each item is specified by a size, an arrival time, a starting deadline and a processing length, such that it has to be packed into a bin between its arrival and its starting deadline and then stay in the bin for a period of its processing length without interruption. Still, the total size of the items placed in a bin can never exceed the bin capacity at any time. The target is to minimize the total bin usage time for packing all the items.

Note that this generalized problem can possibly save large amount of bin usage time for packing all the items. For example, consider that we are given n items to pack, with the i th item having an arrival time of $i - 1$, a starting deadline of n , a processing length of 1 and a size of $\frac{1}{n}$. If each item is packed immediately upon arrival, then any packing solution must incur a total bin usage time of n . Due to the laxity, we can actually delay the starting time of each item to n . In this way, all the items can be packed into one bin, which incurs a bin usage time of 1 only. This suggests that giving each item a laxity can significantly reduce the total bin usage time needed to pack items.

According to the analysis in Section 3.2.2 of Chapter 3, the total bin usage time is mainly influenced by the span of all the items and the total time-space demand of all the items. While the total time-space demand is fixed given a set of items, the span of all the items can be optimized by carefully determining the starting times of the items in the generalized MinUsageTime DBP problem. This is why we study the Flexible Job Scheduling problem, which can be seen as a special case of the generalized MinUsageTime DBP problem by assuming the bin to have an infinite capacity. Khandekar *et al.* [45] showed that the Flexible Job Scheduling problem can be solved in polynomial time via the dynamic programming technique in the offline setting. We thus mainly focus on the problem in the online setting in this chapter.

In Section 5.1, we first introduce the definitions and notations. In Section 5.2, we study the Flexible Job Scheduling problem in the non-clairvoyant online setting. In Section 5.3, we study the problem in the clairvoyant online setting. In Section 5.4, we discuss how the results can be used to generalize the MinUsageTime DBP problem.

5.1 Preliminaries

We define some key notations used in this chapter. Again, for any time interval I , we use I^- and I^+ to denote the left and right endpoints of I respectively. Let $\text{len}(I) = I^+ - I^-$ denote the length of interval I .

For any job J , let $a(J)$, $d(J)$ and $p(J)$ denote J 's arrival time, starting deadline (latest possible starting time)¹ and processing length respectively. When a job J arrives at time

¹We assume a starting deadline for each job rather than a completion deadline as one of the settings we study is non-clairvoyant where the processing length of each job is not known.

$a(J)$, its starting deadline $d(J)$ is known and the job can be started at any time during the time interval $[a(J), d(J)]$. The difference $d(J) - a(J)$ is known as the *laxity* of job J . Once started, job J has to run for a processing length $p(J)$ without interruption. A *scheduler* decides the starting times of the jobs. If a scheduler Λ starts a job J at time t , we refer to the time interval $[t, t + p(J))$ in which J is running as the *active interval* of job J and denote it by $I_\Lambda(J)$. Let $\mathcal{J}$ denote the set of jobs to schedule. The time duration in which at least one job in $\mathcal{J}$ is running is known as the *span* of $\mathcal{J}$. We denote the span of $\mathcal{J}$ induced by a scheduler Λ as $\text{span}_\Lambda(\mathcal{J}) = \text{len}(\bigcup_{J \in \mathcal{J}} I_\Lambda(J))$. The target is to schedule all the jobs to minimize the span. We denote the minimum possible span induced by an optimal scheduler as $\text{span}_{\min}(\mathcal{J})$.

5.2 Non-Clairvoyant Online Setting

In this section, we study the Flexible Job Scheduling problem in the non-clairvoyant online setting, where the processing length of each job is not known until it completes its execution. Thus, the information of processing length cannot be used for scheduling purposes. In Section 5.2.1, we show that no deterministic online scheduler can achieve a competitive ratio less than μ , where $\mu = \frac{\max_{J \in \mathcal{J}} p(J)}{\min_{J \in \mathcal{J}} p(J)}$ is the max/min job processing length ratio. In Section 5.2.2, we introduce two $O(\mu)$ -competitive schedulers: Batch and Batch+. We show that the Batch scheduler achieves a competitive ratio between 2μ and $2\mu + 1$, while the Batch+ scheduler achieves a tight competitive ratio of $\mu + 1$. We further consider a special case in which all the jobs have the same laxity in Section 5.2.3. We establish a lower bound of 2 on the competitive ratio of any deterministic online scheduler and show that the Batch scheduler achieves a tight competitive ratio of 3 in this case.

5.2.1 A Lower Bound on Competitiveness of Any Deterministic Online Algorithm

We construct an instance to establish the lower bound μ on the competitive ratio of any deterministic online scheduler. In our instance, each job has one of two processing lengths: 1 or μ , where $\mu > 1$ can be revealed to the scheduler in advance. In the non-clairvoyant online setting, an adversary has the freedom of assigning a processing length to a job

after it is started. In our instance, each job is assigned the processing length 1 time unit after it is started since the processing lengths of all jobs are at least 1. Therefore, if a job J is assigned a processing length of 1, it completes execution immediately. If the job is assigned a processing length of μ , it continues to run for another $\mu - 1$ time units before completing execution. Let $k \geq 1$ be an integer. We release jobs in up to $k + 1$ iterations. For each iteration i , let T_i denote the time to release all the jobs of this iteration. Among all the jobs released in iteration i , the number of concurrently running jobs induced by an online scheduler shall be referred to as the *concurrency* of iteration i . Note that each iteration has its own concurrency and the concurrency normally changes over time.

In the first iteration, we start by releasing 2^{2^k} jobs at time $T_1 = 0$. Let the j th job ($1 \leq j \leq 2^{2^k}$) have a laxity of α^j , where $\alpha > \mu + 1$ is a constant. As long as the concurrency of the first iteration does not exceed $\sqrt{2^{2^k}} = 2^{2^{k-1}}$ by applying the online scheduler, all the jobs are assigned a processing length of 1 when they are due for the length assignment. If the concurrency never exceeds $2^{2^{k-1}}$ until all the jobs complete execution, we stop releasing jobs and do not proceed with any further iteration. Otherwise, consider the time t_1 when the concurrency first exceeds $2^{2^{k-1}}$. Suppose there are m_1 ($m_1 \geq 2^{2^{k-1}} + 1$) jobs running at time t_1 . Let $J_1, J_2, \dots, J_{m_1}$ be all these jobs sorted in an increasing order of their laxities. Note that none of these jobs has been assigned the processing length, since all the jobs assigned the processing length so far were assigned a length of 1 and have completed execution. We shall assign a processing length of μ to job J_{m_1} and refer to it as the *earmarked job* of the first iteration. All the other jobs $J_1, J_2, \dots, J_{m_1-1}$ will be assigned the same processing length of 1. Apart from this, all the jobs started after time t_1 will also be assigned a processing length of 1. Then, we set T_2 , the job release time of the second iteration, to the completion time of the earmarked job J_{m_1} of the first iteration.

The above process is repeated iteratively with decreasing number of jobs released in each iteration. In each iteration i ($2 \leq i \leq k$), we release $2^{2^{k-i+1}}$ jobs at time T_i . Let the j th job ($1 \leq j \leq 2^{2^{k-i+1}}$) have a laxity of α^j . As long as the concurrency of iteration i does not exceed $\sqrt{2^{2^{k-i+1}}} = 2^{2^{k-i}}$ by applying the online scheduler, we assign a processing length of 1 to all the jobs. If the concurrency never exceeds $2^{2^{k-i}}$ until all the jobs released in iteration i complete execution, we do not proceed with any

 Figure 5.1: Jobs released in iterations i and $i + 1$

further iteration of job release. Otherwise, let t_i denote the time when the concurrency of iteration i first exceeds 2^{2k-i} . Suppose there are m_i ($m_i \geq 2^{2k-i} + 1$) jobs running at time t_i , and let $J_1, J_2, \dots, J_{m_i}$ be all these jobs sorted in an increasing order of their laxities. We shall assign a processing length of μ to job J_{m_i} and refer to it as the *earmarked job* of iteration i . All the other jobs $J_1, J_2, \dots, J_{m_i-1}$ will be assigned the same processing length of 1. In addition, we shall assign a processing length of 1 to all the jobs started after time t_i . Then, we set T_{i+1} , the job release time of iteration $i + 1$, to the completion time of the earmarked job J_{m_i} of iteration i . Figure 5.1 illustrates the above process. In the final iteration $k + 1$ (if needed), we release 2^{2k} jobs at time T_{k+1} and directly assign the same processing length of 1 to all these jobs.

We introduce two lemmas for analyzing the online scheduler and the optimal scheduler of the above instance.

Lemma 5.1. *For each iteration i ($1 \leq i \leq k$), if there is no earmarked job, the span induced by the online scheduler for all the jobs released in this iteration must be at least 2^{2k-i} .*

Proof: An iteration i does not have an earmarked job only if its concurrency never exceeds $2^{2^{k-i}}$. In this case, all the jobs released in iteration i have the same processing length of 1, so their total length is $2^{2^{k-i+1}}$. Thus, the span of these jobs is at least $2^{2^{k-i+1}}/2^{2^{k-i}} = 2^{2^{k-i}}$. $\square$

Lemma 5.2. *For each iteration i ($2 \leq i \leq k+1$), all the earmarked jobs of the first $i-1$ iterations can be started at the job release time T_i of iteration i .*

Proof: For each iteration j ($1 \leq j < i$), let t_j denote the time when the concurrency of iteration i first exceeds $2^{2^{k-j}}$. Among all the m_j ($m_j \geq 2^{2^{k-j}} + 1$) jobs running at t_j , let $\tilde{J}_j$ be the job with the smallest laxity and let $\hat{J}_j$ be the job with the largest laxity ($\hat{J}_j$ is the earmarked job of iteration j). Denote the laxity of job $\hat{J}_j$ by α^{L_j} . Since the jobs released in iteration j have exponentially increasing laxities, it is easy to infer that $L_j \geq m_j \geq 2^{2^{k-j}} + 1$ and the laxity of job $\tilde{J}_j$ is at most $\alpha^{L_j - m_j + 1}$. We next show that $T_i \leq T_j + \alpha^{L_j}$ for each $1 \leq j < i$, which suggests that the earmarked job of iteration j can be started at time T_i .

Note that in iteration j , the earmarked job $\hat{J}_j$ is running at time t_j and has a processing length of μ . Since T_{j+1} is exactly the completion time of job $\hat{J}_j$, we have $T_{j+1} \leq t_j + \mu$. On the other hand, job $\tilde{J}_j$ is also running at time t_j and has a processing length of 1. Thus, $\tilde{J}_j$ must be started no earlier than $t_j - 1$. Since the laxity of $\tilde{J}_j$ is at most $\alpha^{L_j - m_j + 1}$, $\tilde{J}_j$'s starting deadline is bounded by $T_j + \alpha^{L_j - m_j + 1}$. As a result, we have $t_j - 1 \leq T_j + \alpha^{L_j - m_j + 1}$. Therefore,

$$T_{j+1} \leq t_j + \mu \leq T_j + (\mu + 1) + \alpha^{L_j - m_j + 1}.$$

Since the above inequality holds for each $1 \leq j < i$. It follows that

$$T_i \leq T_j + (i - j) \cdot (\mu + 1) + \sum_{l=j}^{i-1} \alpha^{L_l - m_l + 1}.$$

Now, to prove $T_i \leq T_j + \alpha^{L_j}$, we only need to show $(i - j) \cdot (\mu + 1) + \sum_{l=j}^{i-1} \alpha^{L_l - m_l + 1} \leq \alpha^{L_j}$.

For each $j \leq l < i$, we have $L_l \geq m_l \geq 2^{2^{k-l}} + 1$. Since the total number of jobs released in iteration $l + 1$ is $2^{2^{k-l}}$, we also have $2^{2^{k-l}} \geq L_{l+1}$. Thus, $L_l \geq L_{l+1} + 1$ for

each $j \leq l < i$. Consequently, $L_j \geq L_l + (l - j)$. Note that $m_l \geq 3$. Therefore,

$$\begin{aligned}
 \sum_{l=j}^{i-1} \alpha^{L_l - m_l + 1} &= \left(\sum_{l=j+1}^{i-1} \alpha^{L_l - m_l + 1} \right) + \alpha^{L_j - m_j + 1} \\
 &< \left(\sum_{l=j+1}^{i-1} \alpha^{L_l - 2} \right) + \frac{\alpha^{L_j}}{\alpha^2} \\
 &\leq \left(\frac{\alpha^{L_j}}{\alpha^3} + \frac{\alpha^{L_j}}{\alpha^4} + \cdots + \frac{\alpha^{L_j}}{\alpha^{i-j+1}} \right) + \frac{\alpha^{L_j}}{\alpha^2} \\
 &< \left(\sum_{h=2}^{\infty} \frac{1}{\alpha^h} \right) \cdot \alpha^{L_j}.
 \end{aligned}$$

Since $\alpha > \mu + 1 \geq 2$ and $i - j \leq k$, we have $(i - j) \cdot (\mu + 1) < k \cdot \alpha$. It is easy to verify that when $\alpha > 2$, $k \cdot \alpha < \alpha^{2^k}$ holds for any $k \geq 1$. Since $L_j \geq m_j \geq 2^{2^{k-j}} + 1 \geq 2^{2^k} + 1 > 2^k + 1$, we have $\alpha^{2^k} < \frac{1}{\alpha} \cdot \alpha^{L_j}$. As a result, $(i - j) \cdot (\mu + 1) < \frac{1}{\alpha} \cdot \alpha^{L_j}$. Thus,

$$(i - j) \cdot (\mu + 1) + \sum_{l=j}^{i-1} \alpha^{L_l - m_l + 1} < \left(\sum_{h=1}^{\infty} \frac{1}{\alpha^h} \right) \cdot \alpha^{L_j} = \frac{1}{\alpha - 1} \cdot \alpha^{L_j} < \alpha^{L_j},$$

where the last inequality holds since $\alpha > 2$. Therefore, $T_i \leq T_j + \alpha^{L_j}$ for each $1 \leq j < i$ and the lemma is proven. $\square$

Now, we derive the ratio between the spans induced by the online scheduler and the optimal scheduler. In the first iteration, if there is no earmarked job, according to Lemma 5.1, the span induced by the online scheduler is at least $2^{2^{k-1}}$. In this case, the optimal schedule is to start all the jobs immediately once they are released (i.e., at time $T_1 = 0$) so that the induced span is 1. Thus, the ratio between the two spans is at least $2^{2^{k-1}}$. If there is an earmarked job in the first iteration, the second iteration follows.

If the job releasing process is terminated after i iterations ($2 \leq i \leq k$), each of the first $i - 1$ iterations has an earmarked job. Since the jobs of each iteration are released when the earmarked job of the previous iteration completes, the active intervals of these $i - 1$ earmarked jobs do not overlap. Moreover, by Lemma 5.1, the span induced by the online scheduler for the jobs released in iteration i is at least $2^{2^{k-i}}$. Thus, the online scheduler must induce a total span at least $(i - 1) \cdot \mu + 2^{2^{k-i}}$. A better schedule is to start all the jobs released in iteration i together with all earmarked jobs of the first $i - 1$ iterations at

time T_i (Lemma 5.2). All the remaining jobs released in the first $i - 1$ iterations (which have processing lengths of 1) can be started immediately once they are released. In this way, the total span is $\mu + (i - 1)$. Therefore, the ratio between the spans induced by the online scheduler and the optimal scheduler is at least $\frac{(i-1)\cdot\mu+2^{2k-i}}{\mu+(i-1)}$.

Finally, if the job releasing process proceeds to iteration $k + 1$, the span induced by the online scheduler must be at least $k \cdot \mu + 1$. A better schedule is to start all the jobs released in iteration $k + 1$ together with all the earmarked jobs of the previous iterations at time T_{k+1} (Lemma 5.2). All the remaining jobs are started immediately once they are released. This schedule induces a span at most $\mu + k$. So, the ratio between the two spans is at least $\frac{k\cdot\mu+1}{\mu+k}$.

In summary, the competitive ratio by applying any deterministic online scheduler is at least

$$\min \left\{ 2^{2k-1}, \min_{2 \leq i \leq k} \left\{ \frac{(i-1) \cdot \mu + 2^{2k-i}}{\mu + (i-1)} \right\}, \frac{k \cdot \mu + 1}{\mu + k} \right\}.$$

Since $\frac{(i-1)\cdot\mu+2^{2k-i}}{\mu+(i-1)} > \frac{2^{2k}}{\mu+k}$ and $\frac{k\cdot\mu+1}{\mu+k} > \frac{\mu}{\frac{\mu}{k}+1}$, the above competitive ratio can be made arbitrarily close to μ as k goes towards infinity. Thus, we can conclude that:

Theorem 5.1. *For the Flexible Job Scheduling problem in the non-clairvoyant online setting, no deterministic online scheduler can achieve a competitive ratio less than μ , where μ is the max/min job processing length ratio among all the jobs to schedule.*

5.2.2 Batch Scheduler and Batch+ Scheduler

It is easy to infer that an eager scheduler that starts every job immediately at its arrival cannot achieve any bounded competitive ratio even for any given μ , because it does not make use of any laxity to boost the concurrency of job execution. Similarly, a lazy scheduler that delays the start of each job till its starting deadline cannot achieve any bounded competitive ratio for any given μ either, since it does not take any advantage of the flexibility offered by the laxity. Without any knowledge of job processing length, an intuitive strategy to improve the concurrency of job execution is to start as many jobs as possible simultaneously. In the following, we first propose a *Batch* scheduler and show

that it can achieve a competitive ratio of $2\mu + 1$ for the Flexible Job Scheduling problem in the non-clairvoyant online setting.

The Batch scheduler determines the starting times of jobs in iterations. In each iteration, the Batch scheduler waits until a job J pending execution hits its starting deadline $d(J)$, i.e., J is the job with the earliest starting deadline among those that have arrived but not yet started execution. We refer to J as the *flag job* of the iteration. If several jobs share the same starting deadline, any one of them can be chosen as the flag job. At time $d(J)$, the Batch scheduler starts all the jobs pending execution and let them run concurrently. Then, the scheduler proceeds to the next iteration to wait for another job to hit its starting deadline and repeats this process.

Theorem 5.2. *For the Flexible Job Scheduling problem in the non-clairvoyant online setting, the Batch scheduler achieves a competitive ratio between 2μ and $2\mu + 1$, where μ is the max/min job processing length ratio among all the jobs to schedule.*

Proof: Let J_i denote the flag job of iteration i as designated by the Batch scheduler. By definition, the flag jobs $J_1, J_2, J_3, \dots$ have increasing starting deadlines. To establish a lower bound on the span induced by an optimal scheduler, we choose a subset of the flag jobs, such that their active intervals cannot overlap with each other by any scheduler. First, we choose J_1 . Then, we find the lowest-indexed flag job J_h that satisfies $d(J_h) \geq d(J_1) + p(J_1)$ and choose the flag job J_{h+1} if J_{h+1} exists. This selection process is repeated. After a flag job J_i is chosen, we find the lowest-indexed flag job J_j that satisfies $d(J_j) \geq d(J_i) + p(J_i)$ and then choose the flag job J_{j+1} if J_{j+1} exists. By the definition of Batch, J_{j+1} arrives later than $d(J_i)$ and thus, $a(J_{j+1}) > d(J_j) \geq d(J_i) + p(J_i)$. Note that $d(J_i) + p(J_i)$ is the latest possible completion time of job J_i . This implies that the active intervals of J_{j+1} and J_i cannot overlap with each other by any scheduler. Thus, the span induced by an optimal scheduler is at least $\sum_{J \in \mathcal{F}} p(J)$, where $\mathcal{F}$ is the subset of flag jobs chosen.

Next, we derive an upper bound on the span induced by the Batch scheduler. Consider any two flag jobs J_i and J_{i+1} chosen in succession. Since J_j is the lowest-indexed flag job satisfying $d(J_j) \geq d(J_i) + p(J_i)$, for each flag job J_g ($i \leq g < j$), it holds that $d(J_g) < d(J_i) + p(J_i)$. This means that all the jobs started in iterations i to $j - 1$ by

Figure 5.2: An example schedule induced by the Batch scheduler

the Batch scheduler are started before time $d(J_i) + p(J_i)$. Thus, all these jobs must complete execution by time $d(J_i) + p(J_i) + \mu \cdot p(J_i)$, since the max/min job processing length ratio is μ . Therefore, the span induced by the Batch scheduler for iterations i to $j - 1$ is bounded by $d(J_i) + p(J_i) + \mu \cdot p(J_i) - d(J_i) = (\mu + 1) \cdot p(J_i)$. Similarly, for all the jobs started in iteration j , since they are started simultaneously, their induced span is bounded by the maximum job processing length, which is in turn capped by $\mu \cdot p(J_i)$. Consequently, the span induced by the Batch scheduler for iterations i to j is bounded by $(\mu + 1) \cdot p(J_i) + \mu \cdot p(J_i) = (2\mu + 1) \cdot p(J_i)$. Putting all the iterations together, the total span induced by the Batch scheduler is bounded by $(2\mu + 1) \cdot \sum_{J \in \mathcal{F}} p(J)$, where $\mathcal{F}$ is the subset of flag jobs chosen. Therefore, the competitive ratio of the Batch scheduler is bounded by $2\mu + 1$.

Now, we construct an instance to show that the competitive ratio of the Batch scheduler is no less than 2μ . Consider three groups of jobs as shown in Figure 5.2. The first group contains m short jobs, each with a laxity of 0 and a processing length of 1. The i th short job in the first group arrives at time $2(i - 1) \cdot \mu$. The second group also contains m short jobs, each with a laxity of $\mu - \varepsilon$ and a processing length of 1, where $\varepsilon > 0$ can be arbitrarily small. The i th short job in the second group arrives at time $2(i - 1) \cdot \mu + \varepsilon$. The third group contains $2m$ long jobs, all of which have the same starting deadline of $2m \cdot \mu$ and a processing length of μ . The i th long job arrives at time $(i - 1) \cdot \mu$. By applying the Batch scheduler, the i th short job in the first group and the $(2i - 1)$ -th long job in the third group are started together at time $2(i - 1) \cdot \mu$ in

the same iteration. The i th short job in the second group and the $(2i)$ -th long job in the third group are started together at time $(2i - 1)\mu$ in the same iteration. Therefore, the total span induced is $2m \cdot \mu$. A better schedule is to start all the long jobs in the third group at their (same) starting deadlines and start all the short jobs in the first two groups immediately at their arrivals. This yields a span of $m \cdot (1 + \varepsilon) + \mu$. Therefore, the ratio between the spans induced by the Batch scheduler and the optimal scheduler is at least $\frac{2m \cdot \mu}{m \cdot (1 + \varepsilon) + \mu}$, which can be made arbitrarily close to 2μ as m goes towards infinity. $\square$

To improve the competitiveness, we can make the Batch scheduler more aggressive. Next, we propose a tight $(\mu + 1)$ -competitive *Batch+* scheduler. Same as the Batch scheduler, Batch+ also determines the starting times of jobs in iterations. In each iteration, the Batch+ scheduler determines a flag job (the job with the earliest starting deadline) and starts all the pending jobs together with the flag job. What is different from Batch is that, during the active interval of the flag job, the Batch+ scheduler further starts all the newly arriving jobs immediately at their arrivals. Only when the flag job completes execution, the Batch+ scheduler starts the next iteration to buffer incoming jobs and search for a new flag job.

Theorem 5.3. *For the Flexible Job Scheduling problem in the non-clairvoyant online setting, the Batch+ scheduler achieves a tight competitive ratio of $\mu + 1$, where μ is the max/min job processing length ratio among all the jobs to schedule.*

Proof: We first show that the Batch+ scheduler is $(\mu + 1)$ -competitive. Let J_i denote the flag job of iteration i as designated by the Batch+ scheduler. By applying Batch+, all the jobs started in iteration i are started no later than the completion time of J_i , i.e., $d(J_i) + p(J_i)$. Since the max/min job processing length ratio is μ , the span of the jobs started in iteration i is at most $(\mu + 1) \cdot p(J_i)$. Therefore, the total span induced by the Batch+ scheduler is at most $(\mu + 1) \cdot \sum_{i \geq 1} p(J_i)$. Note that the flag job J_{i+1} of iteration $i + 1$ must arrive later than $d(J_i) + p(J_i)$. Thus, the active intervals of J_i and J_{i+1} cannot overlap with each other by any scheduler. Therefore, the span induced by an optimal scheduler is at least $\sum_{i \geq 1} p(J_i)$. As a result, the competitive ratio of the Batch+ scheduler is bounded by $\mu + 1$.

Figure 5.3: An example schedule induced by the Batch+ scheduler

We construct an instance to show that the competitive ratio of $\mu + 1$ is tight. Consider two groups of jobs as shown in Figure 5.3. One group contains m short jobs, each with a laxity of 0 and a processing length of 1. The i th short job arrives at time $(i - 1) \cdot (\mu + 1)$. The other group contains m long jobs, all of which have the same starting deadline of $m \cdot (\mu + 1)$ and a processing length of μ . The i th long job arrives at time $(i - 1) \cdot (\mu + 1) + (1 - \varepsilon)$, where $\varepsilon > 0$ can be arbitrarily small. By applying the Batch+ scheduler, the i th short job and the i th long job are started immediately at their arrivals in the same iteration, and thus a total span of $m \cdot (\mu + 1 - \varepsilon)$ is induced. An optimal schedule is to start all long jobs at their (same) starting deadlines and start all the short jobs immediately at their arrivals. This yields a span of $m + \mu$. Therefore, the ratio between the spans induced by the Batch+ scheduler and the optimal scheduler is $\frac{m \cdot (\mu + 1 - \varepsilon)}{m + \mu}$, which can be made arbitrarily close to $\mu + 1$ as m goes towards infinity. $\square$

5.2.3 A Uniform Laxity Case

In this section, we consider a special case in which all the jobs have the same laxity l for scheduling. We show that no deterministic online scheduler can achieve a competitive ratio less than 2, and the Batch scheduler achieves a tight competitive ratio of 3 in this case.

Theorem 5.4. *For the Flexible Job Scheduling problem in the non-clairvoyant online setting, no deterministic online scheduler can achieve a competitive ratio less than 2, if all the jobs have the same laxity for scheduling.*

Proof: We construct an instance to prove this result. Let l be the laxity of all the jobs. Let job J_1 arrive at time 0 and have a processing length of ε , where $\varepsilon > 0$ can be arbitrarily small. Suppose J_1 is started at time t_1 by an online scheduler. We then release another job J_2 at time t_1 immediately. If J_2 is not started before time $t_1 + \varepsilon$ so that the active intervals of J_1 and J_2 do not overlap, we assign J_2 a processing length of ε . In this way, the total span of J_1 and J_2 is 2ε , while the optimal schedule for J_1 and J_2 is to start them together at time t_1 to induce a span of ε . Consequently, the ratio between the two spans is 2. Now we consider the case that J_2 is started earlier than time $t_1 + \varepsilon$ so that the active intervals of job J_1 and J_2 overlap. We then assign J_2 a processing length of $\frac{l}{2}$ and let a third job J_3 with a processing length of $\frac{l}{2}$ arrive at time $t_1 + l$. It is easy to see that the active intervals of job J_2 and J_3 cannot overlap in this case and thus, the span induced by the online scheduler is no less than l . In an optimal schedule, J_2 and J_3 can be started together at time $t_1 + l$ to induce a span of $\varepsilon + \frac{l}{2}$. Therefore, the ratio between the two spans is no less than $\frac{l}{\varepsilon + l/2}$ which can be made arbitrarily close to 2 as ε approaches 0. Hence, the theorem is proven. $\square$

Now, we show that the Batch scheduler achieves a tight competitive ratio of 3 in the non-clairvoyant online setting, if all the jobs have the same laxity l . Given a set of jobs $\mathcal{J}$, we use $I_B(J)$ and $I^*(J)$ to denote the active interval of each job J induced by the Batch scheduler and an optimal scheduler respectively. Note that by applying the Batch scheduler, several jobs may be started at the same time in a batch. We thus first reduce $\mathcal{J}$ to a subset $\mathcal{J}'$ without decreasing the span: for each batch, among all the jobs started together in the batch, we select the job with the longest processing length and add it into $\mathcal{J}'$. After the reduction, the span induced by the Batch scheduler can be written as $\text{len}(\bigcup_{J \in \mathcal{J}'} I_B(J))$.

Next, we further divide $\mathcal{J}'$ into two subsets $\mathcal{J}'_1$ and $\mathcal{J}'_2$, such that $\mathcal{J}'_1$ contains all the jobs $J \in \mathcal{J}'$ satisfying $I_B(J) \subseteq \bigcup_{J \in \mathcal{J}} I^*(J)$ and $\mathcal{J}'_2$ contains all the remaining jobs in $\mathcal{J}'$. By definition, the span induced by the Batch scheduler on $\mathcal{J}'_1$ is bounded by the optimal solution. What is left is to bound $\text{len}\left(\left(\bigcup_{J \in \mathcal{J}'_2} I_B(J)\right) \setminus \left(\bigcup_{J \in \mathcal{J}} I^*(J)\right)\right)$ by 2 times the optimal solution.

Without loss of generality, we assume that the optimal schedule on $\mathcal{J}$ consists of multiple contiguous time intervals $I_1^*, I_2^*, \dots, I_m^*$, i.e., $\bigcup_{J \in \mathcal{J}} I^*(J) = \bigcup_{i=1}^m I_i^*$ with $I_i^* \cap I_j^* = \emptyset$

for any $i \neq j$. We use $\mathcal{J}_i^*$ to denote all the jobs with their active intervals induced by the optimal schedule fully contained in I_i^* , i.e., $\mathcal{J}_i^* = \{J \in \mathcal{J} : I^*(J) \subseteq I_i^*\}$. Now we prove that for each $i \in \{1, \dots, m\}$,

$$\text{len} \left(\bigcup_{J \in \mathcal{J}_2' \cap \mathcal{J}_i^*} (I_B(J) \setminus I_i^*) \right) \leq 2 \cdot \text{len}(I_i^*).$$

Note that each job $J \in \mathcal{J}_2' \cap \mathcal{J}_i^*$ satisfies $I_B(J) \setminus I_i^* \neq \emptyset$. Thus, J must either satisfy $I_B(J)^- < I_i^{*-}$ or $I_B(J)^+ > I_i^{*+}$. Consequently, we can rewrite $\text{len} \left(\bigcup_{J \in \mathcal{J}_2' \cap \mathcal{J}_i^*} (I_B(J) \setminus I_i^*) \right)$ as

$$\text{len} \left(\bigcup_{J \in \mathcal{J}_2' \cap \mathcal{J}_i^* : I_B(J)^- < I_i^{*-}} (I_B(J) \setminus I_i^*) \right) + \text{len} \left(\bigcup_{J \in \mathcal{J}_2' \cap \mathcal{J}_i^* : I_B(J)^+ > I_i^{*+}} (I_B(J) \setminus I_i^*) \right).$$

In the following text, we show that $\text{len} \left(\bigcup_{J \in \mathcal{J}_2' \cap \mathcal{J}_i^* : I_B(J)^- < I_i^{*-}} (I_B(J) \setminus I_i^*) \right) \leq \text{len}(I_i^*)$ and $\text{len} \left(\bigcup_{J \in \mathcal{J}_2' \cap \mathcal{J}_i^* : I_B(J)^+ > I_i^{*+}} (I_B(J) \setminus I_i^*) \right) \leq \text{len}(I_i^*)$.

To prove $\text{len} \left(\bigcup_{J \in \mathcal{J}_2' \cap \mathcal{J}_i^* : I_B(J)^- < I_i^{*-}} (I_B(J) \setminus I_i^*) \right) \leq \text{len}(I_i^*)$, we show that at most one job $\tilde{J} \in \mathcal{J}_2' \cap \mathcal{J}_i^*$ can satisfy $I_B(\tilde{J})^- < I_i^{*-}$. Suppose two different jobs $\tilde{J}_1$ and $\tilde{J}_2$ are all started earlier than time I_i^{*-} by the Batch scheduler. Without loss of generality, suppose $\tilde{J}_1$ is started earlier than $\tilde{J}_2$. Then, we have $I_B(\tilde{J}_1)^- < I_B(\tilde{J}_2)^- < I_i^{*-}$. By the definition of the Batch scheduler, each job started in the batch in which $\tilde{J}_2$ is scheduled must arrive later than time $I_B(\tilde{J}_1)^-$. Besides, there must exist a flag job in that batch hitting its starting deadline. This suggests that $I_B(\tilde{J}_2)^- - I_B(\tilde{J}_1)^- > l$. Since job $\tilde{J}_1$ can be started at time $I_B(\tilde{J}_1)^-$, its arrival time must satisfy $a(\tilde{J}_1) \leq I_B(\tilde{J}_1)^- < I_B(\tilde{J}_2)^- - l < I_i^{*-} - l$. On the other hand, since $\tilde{J}_1 \in \mathcal{J}_i^*$, $\tilde{J}_1$ can be scheduled with its active interval fully inside I_i^* . This suggests that $\tilde{J}_1$'s arrival time must be no earlier than time $I_i^{*-} - l$, which leads to a contradiction. Consequently, if there indeed exists a job $\tilde{J} \in \mathcal{J}_2' \cap \mathcal{J}_i^*$ satisfying $I_B(\tilde{J})^- < I_i^{*-}$, we can bound $\text{len} \left(\bigcup_{J \in \mathcal{J}_2' \cap \mathcal{J}_i^* : I_B(J)^- < I_i^{*-}} (I_B(J) \setminus I_i^*) \right)$ by $\text{len}(I_B(\tilde{J})) = p(\tilde{J}) \leq \text{len}(I_i^*)$.

Next we show that $\text{len} \left(\bigcup_{J \in \mathcal{J}_2' \cap \mathcal{J}_i^* : I_B(J)^+ > I_i^{*+}} (I_B(J) \setminus I_i^*) \right)$ can be bounded by $\text{len}(I_i^*)$.

Note that

$$\begin{aligned} & \text{len} \left(\bigcup_{J \in \mathcal{J}'_2 \cap \mathcal{J}_i^* : I_B(J)^+ > I_i^{*+}} (I_B(J) \setminus I_i^*) \right) \\ &= \text{len} \left(\bigcup_{J \in \mathcal{J}'_2 \cap \mathcal{J}_i^* : I_i^{*+} \in I_B(J)} [I_i^{*+}, I_B(J)^+] \right) + \text{len} \left(\bigcup_{J \in \mathcal{J}'_2 \cap \mathcal{J}_i^* : I_B(J)^- \geq I_i^{*+}} I_B(J) \right). \end{aligned}$$

We check if there exists at least one job $J \in \mathcal{J}'_2 \cap \mathcal{J}_i^*$ satisfying $I_i^{*+} \in I_B(J)$. If no such job exists, then

$$\text{len} \left(\bigcup_{J \in \mathcal{J}'_2 \cap \mathcal{J}_i^* : I_i^{*+} \in I_B(J)} [I_i^{*+}, I_B(J)^+] \right) = 0$$

and

$$\text{len} \left(\bigcup_{J \in \mathcal{J}'_2 \cap \mathcal{J}_i^* : I_B(J)^+ > I_i^{*+}} (I_B(J) \setminus I_i^*) \right) = \text{len} \left(\bigcup_{J \in \mathcal{J}'_2 \cap \mathcal{J}_i^* : I_B(J)^- \geq I_i^{*+}} I_B(J) \right).$$

Similar to the method for showing that there exists at most one job $\tilde{J} \in \mathcal{J}'_2 \cap \mathcal{J}_i^*$ satisfying $I_B(\tilde{J})^- < I_i^{*-}$, we can also show that there exists at most one job $\vec{J} \in \mathcal{J}'_2 \cap \mathcal{J}_i^*$ satisfying $I_B(\vec{J})^- \geq I_i^{*+}$. Consequently, we have

$$\begin{aligned} \text{len} \left(\bigcup_{J \in \mathcal{J}'_2 \cap \mathcal{J}_i^* : I_B(J)^- \geq I_i^{*+}} I_B(J) \right) &= \text{len}(I_B(\vec{J})) \\ &= p(\vec{J}) \\ &\leq \text{len}(I_i^*) \end{aligned}$$

and thus,

$$\begin{aligned} & \text{len} \left(\bigcup_{J \in \mathcal{J}'_2 \cap \mathcal{J}_i^*} (I_B(J) \setminus I_i^*) \right) \\ &= \text{len} \left(\bigcup_{J \in \mathcal{J}'_2 \cap \mathcal{J}_i^* : I_B(J)^- < I_i^{*-}} (I_B(J) \setminus I_i^*) \right) + \text{len} \left(\bigcup_{J \in \mathcal{J}'_2 \cap \mathcal{J}_i^* : I_B(J)^+ > I_i^{*+}} (I_B(J) \setminus I_i^*) \right) \\ &\leq \text{len}(I_i^*) + \text{len}(I_i^*) \\ &= 2 \cdot \text{len}(I_i^*). \end{aligned}$$

In the case that there exists at least one job $J \in \mathcal{J}'_2 \cap \mathcal{J}_i^*$ satisfying $I_i^{*+} \in I_B(J)$, we choose $\hat{J}$ from $\{J \in \mathcal{J}'_2 \cap \mathcal{J}_i^* : I_i^{*+} \in I_B(J)\}$ such that

$$I_B(\hat{J})^+ = \max_{J \in \mathcal{J}'_2 \cap \mathcal{J}_i^* : I_i^{*+} \in I_B(J)} I_B(J)^+.$$

We then have

$$\text{len} \left(\bigcup_{J \in \mathcal{J}'_2 \cap \mathcal{J}_i^* : I_i^{*+} \in I_B(J)} [I_i^{*+}, I_B(J)^+] \right) = \text{len} \left([I_i^{*+}, I_B(\hat{\mathcal{J}})^+] \right).$$

If no job in $\mathcal{J}'_2 \cap \mathcal{J}_i^*$ is started at or later than time I_i^{*+} by the Batch scheduler, i.e., $\text{len} \left(\bigcup_{J \in \mathcal{J}'_2 \cap \mathcal{J}_i^* : I_B(J)^- \geq I_i^{*+}} I_B(J) \right) = 0$, then

$$\begin{aligned} \text{len} \left(\bigcup_{J \in \mathcal{J}'_2 \cap \mathcal{J}_i^* : I_B(J)^+ > I_i^{*+}} (I_B(J) \setminus I_i^*) \right) &= \text{len} \left([I_i^{*+}, I_B(\hat{\mathcal{J}})^+] \right) \\ &\leq \text{len}(I_B(\hat{\mathcal{J}})) \\ &= p(\hat{\mathcal{J}}) \\ &\leq \text{len}(I_i^*) \end{aligned}$$

and again

$$\text{len} \left(\bigcup_{J \in \mathcal{J}'_2 \cap \mathcal{J}_i^*} (I_B(J) \setminus I_i^*) \right) \leq 2 \cdot \text{len}(I_i^*).$$

Otherwise, there exists a job $\vec{J} \in \mathcal{J}'_2 \cap \mathcal{J}_i^*$ started at or later than time I_i^{*+} by the Batch scheduler, i.e., $I_B(\vec{J})^- \geq I_i^{*+}$, and we have

$$\begin{aligned} \text{len} \left(\bigcup_{J \in \mathcal{J}'_2 \cap \mathcal{J}_i^* : I_B(J)^- \geq I_i^{*+}} I_B(J) \right) &= \text{len}(I_B(\vec{J})) \\ &= p(\vec{J}). \end{aligned}$$

By definition, job $\vec{J}$ must arrive later than time $I_B(\hat{\mathcal{J}})^-$. Besides, according to the optimal schedule, $\vec{J}$ can be scheduled with its active interval fully contained in I_i^* . Thus,

$$\begin{aligned} p(\vec{J}) &= \text{len}(I^*(\vec{J})) \\ &= I^*(\vec{J})^+ - I^*(\vec{J})^- \\ &< I_i^{*+} - I_B(\hat{\mathcal{J}})^- \\ &= \text{len} \left([I_B(\hat{\mathcal{J}})^-, I_i^{*+}) \right). \end{aligned}$$

As a result,

$$\begin{aligned}
 & \text{len} \left(\bigcup_{J \in \mathcal{J}'_2 \cap \mathcal{J}_i^* : I_B(J)^+ > I_i^{*+}} (I_B(J) \setminus I_i^*) \right) \\
 & \leq \text{len} \left([I_i^{*+}, I_B(\widehat{J})^+] \right) + p(\vec{J}) \\
 & \leq \text{len} \left([I_i^{*+}, I_B(\widehat{J})^+] \right) + \text{len} \left([I_B(\widehat{J})^-, I_i^{*+}] \right) \\
 & = \text{len}(I_B(\widehat{J})) \\
 & \leq \text{len}(I_i^*)
 \end{aligned}$$

and

$$\text{len} \left(\bigcup_{J \in \mathcal{J}'_2 \cap \mathcal{J}_i^*} (I_B(J) \setminus I_i^*) \right) \leq 2 \cdot \text{len}(I_i^*).$$

In summary, we can bound the span induced by the Batch scheduler by

$$\begin{aligned}
 \text{len} \left(\bigcup_{J \in \mathcal{J}} I_B(J) \right) &= \text{len} \left(\bigcup_{J \in \mathcal{J}'} I_B(J) \right) \\
 &\leq \text{len} \left(\left(\bigcup_{J \in \mathcal{J}'} I_B(J) \right) \setminus \left(\bigcup_{J \in \mathcal{J}} I^*(J) \right) \right) + \text{len} \left(\bigcup_{J \in \mathcal{J}} I^*(J) \right) \\
 &= \text{len} \left(\left(\bigcup_{J \in \mathcal{J}'_2} I_B(J) \right) \setminus \left(\bigcup_{J \in \mathcal{J}} I^*(J) \right) \right) + \text{len} \left(\bigcup_{J \in \mathcal{J}} I^*(J) \right) \\
 &\leq \sum_{i=1}^m \text{len} \left(\bigcup_{J \in \mathcal{J}'_2 \cap \mathcal{J}_i^*} (I_B(J) \setminus I_i^*) \right) + \text{len} \left(\bigcup_{J \in \mathcal{J}} I^*(J) \right) \\
 &\leq \sum_{i=1}^m 2 \cdot \text{len}(I_i^*) + \text{len} \left(\bigcup_{J \in \mathcal{J}} I^*(J) \right) \\
 &= 2 \cdot \text{len} \left(\bigcup_{J \in \mathcal{J}} I^*(J) \right) + \text{len} \left(\bigcup_{J \in \mathcal{J}} I^*(J) \right) \\
 &= 3 \cdot \text{len} \left(\bigcup_{J \in \mathcal{J}} I^*(J) \right).
 \end{aligned}$$

Hence, the Batch scheduler is 3-competitive in the non-clairvoyant online setting, if all the jobs have the same laxity.

An instance can be constructed to show that the competitive ratio 3 is tight. Let a job with a processing length of ε be released at time 0 and three jobs with the same

processing length of l be released at times $l - \varepsilon$, $l + \varepsilon$ and $2l + 2\varepsilon$ respectively. By applying the Batch scheduler, the first job and the second job is started together at time l in the first batch; the third job is started at time $2l + \varepsilon$ in the second batch; the fourth job is started at time $3l + 2\varepsilon$. Thus, the span induced by the Batch scheduler is $3l$. In an optimal schedule, the first job is started at time 0; the second job is started at time $2l - \varepsilon$; the third job is started at time $2l + \varepsilon$; the fourth job is started at time $2l + 2\varepsilon$. In this way, the span is $l + 4\varepsilon$. It is easy to see than the ratio between the two spans can be made arbitrarily close to 3 as ε approaches 0.

Theorem 5.5. *For the Flexible Job Scheduling problem in the non-clairvoyant online setting, the Batch scheduler achieves a tight competitive ratio of 3, if all the jobs have the same laxity for scheduling.*

5.3 Clairvoyant Online Setting

In this section, we study the Flexible Job Scheduling problem in the clairvoyant online setting, where the processing length of each job is known at its arrival so that this information can be used by the scheduler. In Section 5.3.1, we establish a lower bound 2 on the competitive ratio of any deterministic online scheduler. In Section 5.3.2, we apply a classify-by-duration strategy to the Batch+ scheduler and show that the resultant algorithm achieves a competitive ratio of $7 + 2\sqrt{6}$. In Section 5.3.3, we propose another online scheduler called Profit which achieves an improved competitive ratio of $4 + 2\sqrt{2}$.

5.3.1 A Lower Bound on Competitiveness of Any Deterministic Online Algorithm

We construct an instance to establish a lower bound 2 on the competitive ratio of any deterministic online scheduler. Let k be an integer. In our instance, we release jobs in up to k iterations and let each job have a processing length of either $\frac{1}{k^2}$ or 1. A job with a processing length of $\frac{1}{k^2}$ is given zero laxity and is thus an interval job, while a job with a processing length of 1 is given non-zero laxity and is called a long job. In each iteration i ($1 \leq i < k$), a long job is released at time $2(i-1)$ and is given a laxity of $2(k+1) - 2(i-1)$. We also release interval jobs during the time interval $[2(i-1), 2i-1)$ one after another

(i.e., the first interval job is released at time $2(i-1)$, the second interval job is released at time $2(i-1) + \frac{1}{k^2}$, and so on) if the long job is not started. Consequently, the number of interval jobs released in one iteration is at most k^2 . If the long job is not started during the time interval $[2(i-1), 2i-1)$, we terminate the job releasing process after iteration i and do not proceed with any further iteration. Otherwise, we immediately stop releasing interval jobs in iteration i once the long job is started during $[2(i-1), 2i-1)$ and let an iteration $i+1$ follow. In the final iteration k (if needed), a long job with a laxity of 4 is released at time $2(k-1)$ and no interval job is released in this iteration.

Now, we examine the ratio between the spans induced by any deterministic online scheduler and the optimal scheduler. In the case that the job releasing process terminates after i iterations where $1 \leq i < k$, we must have k^2 interval jobs released in iteration i , and the long job is started after these k^2 interval jobs are completed. Thus, the span induced by the online scheduler for all the jobs released in iteration i is exactly 2. Let x_j denote the total length of the interval jobs released in iteration j ($1 \leq j < i$). According to the description above, the long job released in iteration j must be started no earlier than time $2(j-1) + x_j - \frac{1}{k^2}$. Thus, the span induced by the online scheduler must be at least $\sum_{j=1}^{i-1} (x_j - \frac{1}{k^2} + 1) + 2$. On the other hand, the optimal schedule is to start all the long jobs together at time $2(i-1)$, which induces a span of $\sum_{j=1}^{i-1} x_j + 1$. Thus, the ratio between the two spans is at least

$$\begin{aligned}
 \frac{\sum_{j=1}^{i-1} (x_j - \frac{1}{k^2} + 1) + 2}{\sum_{j=1}^{i-1} x_j + 1} &= \frac{\sum_{j=1}^{i-1} x_j + (i+1)}{\sum_{j=1}^{i-1} x_j + 1} - \frac{\frac{i-1}{k^2}}{\sum_{j=1}^{i-1} x_j + 1} \\
 &> \frac{\sum_{j=1}^{i-1} x_j + (i+1)}{\sum_{j=1}^{i-1} x_j + 1} - \frac{\frac{1}{k}}{\sum_{j=1}^{i-1} x_j + 1} \\
 &\geq \frac{\sum_{j=1}^{i-1} x_j + (i+1)}{\sum_{j=1}^{i-1} x_j + 1} - \frac{1}{k} \\
 &\geq \frac{(i-1) + (i+1)}{(i-1) + 1} - \frac{1}{k} \\
 &= 2 - \frac{1}{k},
 \end{aligned}$$

where the last inequality is due to the fact that $x_j \leq 1$ for each $1 \leq j < i$. In the case that the job releasing process terminates after k iterations, the span induced by the

online scheduler is at least $\sum_{j=1}^{k-1} (x_j - \frac{1}{k^2} + 1) + 1$. The optimal schedule is to start all the long jobs together at time $2(k-1)$, which induces a span of $\sum_{j=1}^{k-1} x_j + 1$. Consequently, the ratio between the two spans is

$$\begin{aligned}
 \frac{\sum_{j=1}^{k-1} (x_j - \frac{1}{k^2} + 1) + 1}{\sum_{j=1}^{k-1} x_j + 1} &= \frac{\sum_{j=1}^{k-1} x_j + k}{\sum_{j=1}^{k-1} x_j + 1} - \frac{\frac{k-1}{k^2}}{\sum_{j=1}^{k-1} x_j + 1} \\
 &> \frac{\sum_{j=1}^{k-1} x_j + k}{\sum_{j=1}^{k-1} x_j + 1} - \frac{1}{k} \\
 &\geq \frac{(k-1) + k}{(k-1) + 1} - \frac{1}{k} \\
 &= 2 - \frac{2}{k},
 \end{aligned}$$

where the last inequality is due to the fact that $x_j \leq 1$. In summary, the ratio between the spans induced by the online scheduler and the optimal scheduler is no less than $2 - \frac{2}{k}$, which can be made arbitrarily close to 2 as k goes towards infinity. Hence, we have the following result.

Theorem 5.6. *For the Flexible Job Scheduling problem in the clairvoyant online setting, no deterministic online scheduler can achieve a competitive ratio less than 2.*

5.3.2 A Classify-by-Duration Strategy

As shown in Section 5.2.1, in the non-clairvoyant online setting, the competitiveness of an online scheduler is limited by the max/min job processing length ratio μ . In the clairvoyant online setting, with the knowledge of job processing length, a natural strategy to break this limit is to classify jobs into categories according to their lengths and schedule each category separately. In this way, the max/min job processing length ratio of each job category is reduced compared to the whole job set. In this section, we show that applying such a classify-by-duration strategy to the Batch+ scheduler can achieve an $O(1)$ competitive ratio.

We first give a formal description of the *Classify-by-Duration Batch+* (CDB) scheduler. Let α be the max/min job processing length ratio of each job category. Given a base job processing length b , we classify jobs into categories such that each category includes all the jobs with processing length between $b\alpha^{i-1}$ and $b\alpha^i$ for an integer i . Whenever a

new job arrives, it is put into a category according to its processing length. For each job category, we apply the Batch+ scheduler separately to decide the starting times of jobs. If the max/min job processing length ratio for the entire set of jobs to schedule is μ , there can be up to $\lceil \log_\alpha \mu \rceil + 1$ non-empty job categories produced by the classification. Let $\mathcal{J}_1, \mathcal{J}_2, \dots, \mathcal{J}_{\lceil \log_\alpha \mu \rceil + 1}$ denote these job categories. Let $\mathcal{F}_i$ be the set of flag jobs designated by the Batch+ scheduler in the job category $\mathcal{J}_i$. Let $\mathcal{F} = \bigcup_{i=1}^{\lceil \log_\alpha \mu \rceil + 1} \mathcal{F}_i$ be all the flag jobs designated by Classify-by-Duration Batch+ in the entire job set.

Lemma 5.3. *The span induced by the Classify-by-Duration Batch+ scheduler for a set of jobs $\mathcal{J}$ is bounded by $(\alpha + 1)$ times the span of all the flag jobs $\mathcal{F} \subseteq \mathcal{J}$ in the schedule.*

Proof: By applying the Batch+ scheduler, the active interval of each flag job J is $[d(J), d(J) + p(J)]$. Thus, the span of all the flag jobs is $\bigcup_{J \in \mathcal{F}} [d(J), d(J) + p(J)]$. Such a span may not be a single contiguous time interval. It may be composed of multiple contiguous time intervals that are apart from each other. Let $I_1, I_2, \dots, I_m$ ($m \geq 1$) be these contiguous intervals. Then, we can divide all the flag jobs $\mathcal{F}$ into m groups, such that the span of each job group $\mathcal{H}_i$ is I_i , i.e., $\bigcup_{J \in \mathcal{H}_i} [d(J), d(J) + p(J)] = [\min_{J \in \mathcal{H}_i} d(J), \max_{J \in \mathcal{H}_i} (d(J) + p(J))] = I_i$. The span of all the flag jobs can be written as $\sum_{i=1}^m \text{len}(I_i)$.

By the argument for the Batch+ scheduler in the proof of Theorem 5.3, the active intervals of all the jobs started in the iteration defined by flag job J must lie inside the time interval $[d(J), d(J) + (\alpha + 1) \cdot p(J)]$. Thus, the span of all the jobs $\mathcal{J}$ is bounded by $\text{len} \left(\bigcup_{J \in \mathcal{F}} [d(J), d(J) + (\alpha + 1) \cdot p(J)] \right)$. For each flag job $J \in \mathcal{H}_i$, we have $d(J) \geq \min_{J \in \mathcal{H}_i} d(J) = I_i^-$ and

$$\begin{aligned} d(J) + (\alpha + 1) \cdot p(J) &= (\alpha + 1) \cdot (d(J) + p(J)) - \alpha \cdot d(J) \\ &\leq (\alpha + 1) \cdot \max_{J \in \mathcal{H}_i} (d(J) + p(J)) - \alpha \cdot \min_{J \in \mathcal{H}_i} d(J) \\ &= \min_{J \in \mathcal{H}_i} d(J) + (\alpha + 1) \cdot \left(\max_{J \in \mathcal{H}_i} (d(J) + p(J)) - \min_{J \in \mathcal{H}_i} d(J) \right) \\ &= I_i^- + (\alpha + 1) \cdot \text{len}(I_i). \end{aligned}$$

Hence, $[d(J), d(J) + (\alpha + 1) \cdot p(J)] \subseteq [I_i^-, I_i^- + (\alpha + 1) \cdot \text{len}(I_i)]$. This suggests that the span of all the jobs J is bounded by $\text{len} \left(\bigcup_{i=1}^m [I_i^-, I_i^- + (\alpha + 1) \cdot \text{len}(I_i)] \right)$. Since

$\bigcup_{i=1}^m [I_i^-, I_i^- + (\alpha + 1) \cdot \text{len}(I_i))$ has a total length capped by $(\alpha + 1) \cdot \sum_{i=1}^m \text{len}(I_i)$, the lemma is proven. $\square$

With Lemma 5.3, we only need to analyze the span for the flag jobs $\mathcal{F}$. Recall that $\mathcal{F}$ can be classified into the categories $\mathcal{F}_1, \mathcal{F}_2, \dots, \mathcal{F}_m$. As argued in the proof of Theorem 5.3, each flag job designated by the Batch+ scheduler must arrive later than the latest possible completion time of the previous flag job. Thus, the active intervals of any two flag jobs in the same category cannot overlap with each other by any scheduler. For simplicity of presentation, we shall refer to the ratio between the spans induced by the Classify-by-Duration Batch+ scheduler and an optimal scheduler as the CDB-to-optimal span ratio. In the following, we study the CDB-to-optimal span ratio for any set of jobs satisfying the above property.

Lemma 5.4. *If a set of jobs $\mathcal{J}$ can be classified into a collection of categories $\mathcal{J}_1, \mathcal{J}_2, \dots, \mathcal{J}_m$ such that*

- (a) *the max/min job processing length ratio of each category is bounded by α ,*
- (b) *the active intervals of any two jobs in the same category cannot overlap by any scheduler,*

then the CDB-to-optimal span ratio of $\mathcal{J}$ is bounded by $3 + \frac{1}{\alpha-1}$.

Proof: We prove it by an induction on the number of job categories. If $\mathcal{J}$ contains only one job category, then according to property (b), the active intervals of any two jobs in $\mathcal{J}$ cannot overlap by any scheduler. Thus, the span induced by any scheduler is exactly $\sum_{J \in \mathcal{J}} p(J)$. Therefore, the CDB-to-optimal span ratio of $\mathcal{J}$ is 1, which is bounded by $3 + \frac{1}{\alpha-1}$.

Suppose that the CDB-to-optimal span ratio is bounded by $3 + \frac{1}{\alpha-1}$ for all job sets containing no more than $(n - 1)$ categories. We now show that for any job set $\mathcal{J}$ that contains n categories, its CDB-to-optimal span ratio is also bounded by $3 + \frac{1}{\alpha-1}$. Consider an optimal schedule on $\mathcal{J}$. Let $I^*(J)$ denote the active interval of each job $J \in \mathcal{J}$ in the optimal schedule. Then, $\text{len}\left(\bigcup_{J \in \mathcal{J}} I^*(J)\right) = \text{span}_{\min}(\mathcal{J})$. Now, we derive a lower bound on $\text{span}_{\min}(\mathcal{J})$. Assume that $\mathcal{J}$ can be classified into n categories $\mathcal{J}_1, \mathcal{J}_2, \dots, \mathcal{J}_n$, where

$\mathcal{J}_n$ includes jobs with the longest processing lengths. Let $J_{n,1}, J_{n,2}, \dots, J_{n,q}$ be all the jobs in $\mathcal{J}_n$ sorted in an increasing order of their arrivals. Note that the span $\bigcup_{J \in \mathcal{J}} I^*(J)$ induced by the optimal schedule may consist of multiple contiguous time intervals. For each job $J_{n,j}$ ($1 \leq j \leq q$), let $I_S(J_{n,j})$ denote the contiguous interval that $J_{n,j}$'s active interval $I^*(J_{n,j})$ falls in. Then, we define an interval I_j for $J_{n,j}$ as

$$I_j = [\max\{I_S(J_{n,j})^-, I^*(J_{n,j-1})^+\}, I^*(J_{n,j})^+],$$

i.e., I_j is left delimited by the starting point of $I_S(J_{n,j})$ and the ending point of $J_{n,j-1}$'s active interval, and it is right delimited by the ending point of $J_{n,j}$'s active interval. By property (b) of $\mathcal{J}_n$, we can infer that $I^*(J_{n,j}) \subseteq I_j$ and all the I_j 's do not overlap with each other. For each I_j , define $X(J_{n,j})$ as the set of all the jobs $J \in \mathcal{J}$ satisfying $I^*(J) \subseteq I_j$ or $I^*(J)^- < I_j^+ < I^*(J)^+$, i.e., these jobs have their active intervals either fully contained in I_j or crossing the ending point of I_j . Obviously, $J_{n,j} \in X(J_{n,j})$. Let $\mathcal{J}_{overlap} = \bigcup_{j=1}^q X(J_{n,j})$ and $\mathcal{J}_{misc} = \mathcal{J} \setminus \mathcal{J}_{overlap}$. It is easy to see that the active interval of any job in $\mathcal{J}_{misc}$ does not overlap with any I_j , i.e., $(\bigcup_{J \in \mathcal{J}_{misc}} I^*(J)) \cap (\bigcup_{j=1}^q I_j) = \emptyset$. Figure 5.4 illustrates the above definitions with an example of jobs classified into three categories: $\mathcal{J}_1 = \{J_{1,1}, J_{1,2}, J_{1,3}, J_{1,4}\}$, $\mathcal{J}_2 = \{J_{2,1}, J_{2,2}, J_{2,3}, J_{2,4}, J_{2,5}\}$, $\mathcal{J}_3 = \{J_{3,1}, J_{3,2}\}$.

We can thus establish a lower bound on $\text{span}_{min}(\mathcal{J})$ as

$$\begin{aligned} \text{span}_{min}(\mathcal{J}) &= \text{len}\left(\bigcup_{J \in \mathcal{J}} I^*(J)\right) \\ &\geq \text{len}\left(\bigcup_{J \in \mathcal{J}_{misc}} I^*(J)\right) + \sum_{j=1}^q \text{len}(I_j) \\ &\geq \text{span}_{min}(\mathcal{J}_{misc}) + \sum_{j=1}^q \text{len}(I_j), \end{aligned} \tag{5.1}$$

where $\text{span}_{min}(\mathcal{J}_{misc})$ is the span induced by an optimal scheduler on $\mathcal{J}_{misc}$.

Next, we prove that the span induced by Classify-by-Duration Batch+ for $\mathcal{J}$ has the following upper bound:

$$\text{span}_{CDB}(\mathcal{J}) \leq \left(3 + \frac{1}{\alpha - 1}\right) \cdot \left(\text{span}_{min}(\mathcal{J}_{misc}) + \sum_{j=1}^q \text{len}(I_j)\right). \tag{5.2}$$

 Figure 5.4: Definition of I_j

Due to property (b), by applying Classify-by-Duration Batch+ on $\mathcal{J}$, each job in $\mathcal{J}$ is started at its starting deadline. Thus,

$$\begin{aligned}
 \text{span}_{CDB}(\mathcal{J}) &= \text{len} \left(\bigcup_{J \in \mathcal{J}} [d(J), d(J) + p(J)] \right) \\
 &\leq \text{len} \left(\bigcup_{J \in \mathcal{J}_{misc}} [d(J), d(J) + p(J)] \right) \\
 &\quad + \text{len} \left(\bigcup_{J \in \mathcal{J}_{overlap}} [d(J), d(J) + p(J)] \right) \\
 &= \text{span}_{CDB}(\mathcal{J}_{misc}) + \text{span}_{CDB}(\mathcal{J}_{overlap}).
 \end{aligned}$$

It is obvious that the jobs $\mathcal{J}_{misc}$ come from at most $(n - 1)$ categories $\mathcal{J}_1, \mathcal{J}_2, \dots, \mathcal{J}_{n-1}$. According to the induction hypothesis, the CDB-to-optimal span ratio of $\mathcal{J}_{misc}$ is bounded by $3 + \frac{1}{\alpha-1}$, i.e., $\text{span}_{CDB}(\mathcal{J}_{misc}) \leq (3 + \frac{1}{\alpha-1}) \cdot \text{span}_{min}(\mathcal{J}_{misc})$. What remains is to show that $\text{span}_{CDB}(\mathcal{J}_{overlap}) \leq (3 + \frac{1}{\alpha-1}) \cdot \sum_{j=1}^q \text{len}(I_j)$.

In each job set $X(J_{n,j})$ defined earlier, all the jobs other than $J_{n,j}$ fall in categories $\mathcal{J}_1, \mathcal{J}_2, \dots, \mathcal{J}_{n-1}$. Recall that the active intervals of all the jobs in each category $\mathcal{J}_i$ cannot overlap by any scheduler. We can thus infer that $X(J_{n,j})$ contains at most one job $\tilde{J}_i$ from each category $\mathcal{J}_i$ ($1 \leq i \leq n - 1$) which satisfies $I^*(J_{n,j})^+ < d(\tilde{J}_i) + p(\tilde{J}_i)$. Assume on the contrary that there are two jobs $\tilde{J}_i, \tilde{J}'_i \in X(J_{n,j}) \cap \mathcal{J}_i$ satisfying this condition. By the argument in the proof of Theorem 5.3, either $d(\tilde{J}_i) + p(\tilde{J}_i) \leq a(\tilde{J}'_i)$ or $d(\tilde{J}'_i) + p(\tilde{J}'_i) \leq a(\tilde{J}_i)$ holds. This implies that one of these two jobs must arrive after time $I^*(J_{n,j})^+$. On the other hand, by the definition of $X(J_{n,j})$, these two jobs have their active intervals overlapping with I_j in the optimal schedule. So, they must both arrive before $I_j^+ = I^*(J_{n,j})^+$, which leads to a contradiction. Hence, there is at most

one job $\tilde{J}_i \in X(J_{n,j}) \cap \mathcal{J}_i$ satisfying $I^*(J_{n,j})^+ < d(\tilde{J}_i) + p(\tilde{J}_i)$. All the remaining jobs $J \in X(J_{n,j}) \cap \mathcal{J}_i \setminus \{\tilde{J}_i\}$ must have their completion times $d(J) + p(J) \leq I^*(J_{n,j})^+ = I_j^+$. By the definition of $X(J_{n,j})$, J 's active interval $I^*(J)$ in the optimal schedule must be fully contained in I_j . Since $I^*(J)^- \leq d(J)$, J 's starting deadline $d(J)$ must also be in I_j . As a result, J 's active interval $[d(J), d(J) + p(J))$ induced by Classify-by-Duration Batch+ must be fully contained in I_j . Therefore,

$$\begin{aligned}
 \text{len} \left(\bigcup_{J \in X(J_{n,j})} [d(J), d(J) + p(J)) \right) &= \text{len} \left(\bigcup_{i=1}^{n-1} \left(\bigcup_{J \in X(J_{n,j}) \cap \mathcal{J}_i} [d(J), d(J) + p(J)) \right) \right) \\
 &\quad + \text{len} \left([d(J_{n,j}), d(J_{n,j}) + p(J_{n,j})) \right) \\
 &\leq \text{len} \left(\bigcup_{i=1}^{n-1} \left(\bigcup_{J \in X(J_{n,j}) \cap \mathcal{J}_i \setminus \{\tilde{J}_i\}} [d(J), d(J) + p(J)) \right) \right) \\
 &\quad + \sum_{i=1}^{n-1} \text{len} \left([d(\tilde{J}_i), d(\tilde{J}_i) + p(\tilde{J}_i)) \right) + p(J_{n,j}) \\
 &\leq \text{len}(I_j) + \left(\sum_{i=1}^{n-1} p(\tilde{J}_i) \right) + p(J_{n,j}).
 \end{aligned}$$

Since $\tilde{J}_i \in \mathcal{J}_i$ and $J_{n,j} \in \mathcal{J}_n$, we have $p(\tilde{J}_i) \leq \frac{1}{\alpha^{n-1-i}} \cdot p(J_{n,j})$. Note that $I^*(J_{n,j}) \subseteq I_j$, so $p(J_{n,j}) \leq \text{len}(I_j)$. It follows that $p(\tilde{J}_i) \leq \frac{1}{\alpha^{n-1-i}} \cdot \text{len}(I_j)$. Thus,

$$\begin{aligned}
 \text{len} \left(\bigcup_{J \in X(J_{n,j})} [d(J), d(J) + p(J)) \right) &\leq \text{len}(I_j) + \left(\sum_{i=1}^{n-1} \frac{1}{\alpha^{n-1-i}} \cdot \text{len}(I_j) \right) + \text{len}(I_j) \\
 &< \text{len}(I_j) + \frac{1}{1 - \frac{1}{\alpha}} \cdot \text{len}(I_j) + \text{len}(I_j) \\
 &= \left(3 + \frac{1}{\alpha - 1} \right) \cdot \text{len}(I_j).
 \end{aligned}$$

Therefore,

$$\begin{aligned}
 \text{span}_{CDB}(\mathcal{J}_{\text{overlap}}) &= \text{len} \left(\bigcup_{J \in \mathcal{J}_{\text{overlap}}} [d(J), d(J) + p(J)] \right) \\
 &= \text{len} \left(\bigcup_{j=1}^q \left(\bigcup_{J \in X(J_{n,j})} [d(J), d(J) + p(J)] \right) \right) \\
 &\leq \sum_{j=1}^q \text{len} \left(\bigcup_{J \in X(J_{n,j})} [d(J), d(J) + p(J)] \right) \\
 &< \left(3 + \frac{1}{\alpha - 1} \right) \cdot \sum_{i=1}^q \text{len}(I_j).
 \end{aligned}$$

Hence, inequality (5.2) is proven. Based on (5.1) and (5.2), the CDB-to-optimal span ratio of the job set $\mathcal{J}$ with n categories is bounded by $3 + \frac{1}{\alpha-1}$. By induction, the lemma is proven. $\square$

By Lemma 5.4, the CDB-to-optimal span ratio for the set of flag jobs $\mathcal{F}$ is bounded by $3 + \frac{1}{\alpha-1}$, i.e., $\frac{\text{span}_{CDB}(\mathcal{F})}{\text{span}_{\min}(\mathcal{F})} \leq 3 + \frac{1}{\alpha-1}$. It is obvious that the minimum possible span of any job set $\mathcal{J}$ is no less than that of its flag jobs $\mathcal{F}$ designated by Classify-by-Duration Batch+, i.e., $\text{span}_{\min}(\mathcal{J}) \geq \text{span}_{\min}(\mathcal{F})$. Thus, by Lemma 5.3, we have

$$\frac{\text{span}_{CDB}(\mathcal{J})}{\text{span}_{\min}(\mathcal{J})} \leq \frac{(\alpha + 1) \cdot \text{span}_{CDB}(\mathcal{F})}{\text{span}_{\min}(\mathcal{F})} \leq 3\alpha + 4 + \frac{2}{\alpha - 1}.$$

It is easy to derive that $(3\alpha + 4 + \frac{2}{\alpha-1})$ has a minimum value of $7 + 2\sqrt{6} \approx 11.9$ when α is set to $1 + \sqrt{\frac{2}{3}}$.

Theorem 5.7. *For the Flexible Job Scheduling problem in the clairvoyant online setting, the Classify-by-Duration Batch+ scheduler achieves a competitive ratio of $3\alpha + 4 + \frac{2}{\alpha-1}$, where α is the max/min job processing length ratio for each job category. When setting $\alpha = 1 + \sqrt{\frac{2}{3}}$, this scheduler is $(7 + 2\sqrt{6})$ -competitive.*

5.3.3 Profit Scheduler

Although the Classify-by-Duration Batch+ scheduler can achieve a constant competitive ratio, it schedules different categories of jobs independently, which may lose some opportunities to optimize the span. Moreover, Classify-by-Duration Batch+ does not make

use of any runtime state in scheduling. In fact, with the knowledge of job processing length, we can derive how much the active interval of a new job can overlap with those of currently running jobs. This information would help decide whether starting a new job can make effective use of the existing span. Motivated by this observation, we propose a more competitive *Profit* scheduler that considers all the jobs as a whole.

The Profit scheduler determines the starting times of jobs in iterations. In each iteration, the Profit scheduler waits until a job J_f pending execution hits its starting deadline $d(J_f)$, i.e., J_f is the job with the earliest starting deadline among all pending jobs. Such a job J_f is designated as the *flag job* of this iteration. If several jobs share the same starting deadline, the one with the longest processing length is chosen as the flag job. The Profit scheduler starts the flag job J_f at time $d(J_f)$. For each job J that has arrived before time $d(J_f)$ but not yet started execution, if $p(J) \leq k \cdot p(J_f)$ (where $k > 1$ is an algorithm parameter), the Profit scheduler also starts job J at time $d(J_f)$ in this iteration. This guarantees that at least $\frac{1}{k}$ of J 's active interval overlaps with J_f 's active interval in the span and such a job J is said to be *profitable* to job J_f . Apart from this, if a job J arrives during the active interval of the flag job J_f and satisfies $p(J) \leq k \cdot (d(J_f) + p(J_f) - a(J))$, the Profit scheduler starts job J immediately at its arrival $a(J)$ in this iteration. Since the flag job J_f completes execution at time $d(J_f) + p(J_f)$, the above condition also ensures that at least $\frac{1}{k}$ of J 's active interval overlaps with J_f 's active interval in the span. Such a job J is also said to be *profitable* to job J_f . Meanwhile, among all the pending jobs not profitable to job J_f , the Profit scheduler watches for any job to hit its starting deadline. Once a job hits its starting deadline, a new iteration is initiated and the above job starting process is repeated. Note that by applying the Profit scheduler, the flag jobs of different iterations may be running concurrently. As a result, it is possible that a new incoming job J is profitable to several running flag jobs when it arrives. In this case, the scheduler starts job J immediately and we can attribute J to any iteration of these flag jobs.

In the following text, we analyze the competitive ratio of the Profit scheduler. The analysis framework bears some similarity to that for Classify-by-Processing-Length Batch+, but is more sophisticated than the latter. Let $\mathcal{F}$ be all the flag jobs designated by the Profit scheduler in the entire job set $\mathcal{J}$. The following lemma describes the relation between the spans induced by the Profit scheduler for $\mathcal{J}$ and $\mathcal{F}$.

Lemma 5.5. *The span induced by the Profit scheduler for a set of jobs $\mathcal{J}$ is bounded by k times the span of all the flag jobs $\mathcal{F} \subseteq \mathcal{J}$ in the schedule.*

Proof: This lemma can be proved in a similar way to Lemma 5.3. First, we show that each job J started in an iteration with a flag job J_f must have its active interval fall inside the time interval $[d(J_f), d(J_f) + k \cdot p(J_f)]$. If J arrives before time $d(J_f)$, then $p(J) \leq k \cdot p(J_f)$ and J is started at time $d(J_f)$. Then, the completion time of J satisfies $d(J_f) + p(J) \leq d(J_f) + k \cdot p(J_f)$. If J arrives during the active interval of the flag job J_f , then $p(J) \leq k \cdot (d(J_f) + p(J_f) - a(J))$ and J is started at its arrival time $a(J)$. Hence, the completion time of job J satisfies

$$\begin{aligned}
 a(J) + p(J) &\leq a(J) + k \cdot (d(J_f) + p(J_f) - a(J)) \\
 &= a(J) + k \cdot p(J_f) - k \cdot (a(J) - d(J_f)) \\
 &< a(J) + k \cdot p(J_f) - (a(J) - d(J_f)) \\
 &= d(J_f) + k \cdot p(J_f).
 \end{aligned}$$

Therefore, all the jobs started in an iteration with a flag job J_f must have their active intervals fall inside $[d(J_f), d(J_f) + k \cdot p(J_f)]$. As a result, the span of all the jobs $\mathcal{J}$ is bounded by $\text{len} \left(\bigcup_{J \in \mathcal{F}} [d(J), d(J) + k \cdot p(J)] \right)$.

By applying the Profit scheduler, the active interval of each flag job J_f is $[d(J_f), d(J_f) + p(J_f)]$. Thus, the span of all the flag jobs designated by the Profit scheduler is $\bigcup_{J \in \mathcal{F}} [d(J), d(J) + p(J)]$. Such a span may consist of multiple contiguous time intervals which are apart from each other. Let $I_1, I_2, \dots, I_m$ ($m \geq 1$) be these contiguous intervals. Then, the span of all the flag jobs can be written as $\sum_{i=1}^m \text{len}(I_i)$. We can divide all the flag jobs $\mathcal{F}$ into m groups, such that the span of each job group $\mathcal{H}_i$ is I_i , i.e., $\bigcup_{J \in \mathcal{H}_i} [d(J), d(J) + p(J)] = [\min_{J \in \mathcal{H}_i} d(J), \max_{J \in \mathcal{H}_i} (d(J) + p(J))] = I_i$. For each flag job $J \in \mathcal{H}_i$, we have $d(J) \geq \min_{J \in \mathcal{H}_i} d(J) = I_i^-$ and

$$\begin{aligned}
 d(J) + k \cdot p(J) &= k \cdot (d(J) + p(J)) - (k-1) \cdot d(J) \\
 &\leq k \cdot \max_{J \in \mathcal{H}_i} (d(J) + p(J)) - (k-1) \cdot \min_{J \in \mathcal{H}_i} d(J) \\
 &= \min_{J \in \mathcal{H}_i} d(J) + k \cdot \left(\max_{J \in \mathcal{H}_i} (d(J) + p(J)) - \min_{J \in \mathcal{H}_i} d(J) \right) \\
 &= I_i^- + k \cdot \text{len}(I_i).
 \end{aligned}$$

Hence, $[d(J), d(J) + k \cdot p(J)] \subseteq [I_i^-, I_i^- + k \cdot \text{len}(I_i)]$. This suggests that the span of all the jobs $\mathcal{J}$ is bounded by $\text{len}\left(\bigcup_{i=1}^m [I_i^-, I_i^- + k \cdot \text{len}(I_i)]\right)$. Since $\bigcup_{i=1}^m [I_i^-, I_i^- + k \cdot \text{len}(I_i)]$ has a total length capped by $k \cdot \sum_{i=1}^m \text{len}(I_i)$, the lemma is proven. $\square$

With Lemma 5.5, we only need to analyze the span for the flag jobs $\mathcal{F}$. We start by proving a useful lemma for the flag jobs $\mathcal{F}$.

Lemma 5.6. *For any two flag jobs designated by the Profit scheduler, the one with an earlier starting deadline must be completed before the other one.*

Proof: Let J_1 and J_2 be the flag jobs designated by the Profit scheduler in two different iterations. Note that $d(J_1) \neq d(J_2)$. Otherwise, at their (same) starting deadlines, the job with a longer processing length would be chosen as the flag job by the Profit scheduler. Then, the other job is profitable to the flag job and will be started in the same iteration so that it would not become a flag job itself.

Without loss of generality, suppose J_1 has an earlier starting deadline than J_2 , i.e., $d(J_1) < d(J_2)$. If $a(J_2) \geq d(J_1) + p(J_1)$, then $d(J_2) + p(J_2) > a(J_2) \geq d(J_1) + p(J_1)$. If $a(J_2) < d(J_1) + p(J_1)$, since J_2 is also a flag job and is started at its starting deadline $d(J_2) > d(J_1)$, J_2 must not be profitable to J_1 . If J_2 arrives before time $d(J_1)$, we have $p(J_2) > k \cdot p(J_1)$ and thus $d(J_2) + p(J_2) > d(J_1) + k \cdot p(J_1) > d(J_1) + p(J_1)$. If J_2 arrives during J_1 's active interval $[d(J_1), d(J_1) + p(J_1)]$, we have $p(J_2) > k \cdot (d(J_1) + p(J_1) - a(J_2))$ and it follows that

$$\begin{aligned} d(J_2) + p(J_2) &> a(J_2) + k \cdot (d(J_1) + p(J_1) - a(J_2)) \\ &> a(J_2) + (d(J_1) + p(J_1) - a(J_2)) \\ &= d(J_1) + p(J_1). \end{aligned}$$

Therefore, in any case, J_1 is completed before J_2 . $\square$

For simplicity of presentation, we shall refer to the ratio between the spans induced by the Profit scheduler and an optimal scheduler as the Profit-to-optimal span ratio. To study the Profit-to-optimal span ratio, we construct a directed graph $G(\mathcal{F}, E)$ for the set of flag jobs $\mathcal{F}$ as follows. For each flag job $J \in \mathcal{F}$, let $X(J)$ be the set of all the flag

the schedule induced by the Profit scheduler

$$X(J_1) = \{J_2, J_{root}\} \quad X(J_2) = \{J_{root}\} \quad X(J_3) = \{J_4, J_{root}\}$$

$$X(J_4) = \{J_{root}\} \quad X(J_5) = \{J_6, J_{root}\} \quad X(J_6) = \{J_{root}\}$$

$$X(J_7) = \{J_{root}\} \quad X(J_{root}) \text{ is an empty set}$$

Figure 5.5: Constructing a graph for flag jobs

jobs $J' \in \mathcal{F}$ satisfying $a(J') < d(J) + p(J)$ and $d(J) < d(J')$, i.e., J' arrives before J completes and J' is started after J . Thus, J' must not be profitable to J . If $X(J) \neq \emptyset$, then among all the jobs in $X(J)$, we choose the job $\tilde{J}$ with the earliest starting deadline and create an directed edge from $\tilde{J}$ to J in the graph $G(\mathcal{F}, E)$. Figure 5.5 shows an example of graph construction. It can be shown that the graph constructed in this way is a collection of trees (i.e., a forest).

Lemma 5.7. *The directed graph $G(\mathcal{F}, E)$ is a collection of trees.*

Proof: By definition, for each job $J \in \mathcal{F}$ with $X(J) \neq \emptyset$, there is only one job $J' \in X(J)$ which has an edge pointing to J . Moreover, no circle exists in $G(\mathcal{F}, E)$. Otherwise, suppose a set of edges $(J_1, J_2), (J_2, J_3), \dots, (J_{m-1}, J_m), (J_m, J_1) \in E$ form a circle. According to the definition of an edge $(J', J) \in E$, we have $d(J) < d(J')$. Consequently, $d(J_1) < d(J_m) < d(J_{m-1}) < \dots < d(J_3) < d(J_2) < d(J_1)$, which leads to a contradiction. Hence, the lemma is proven. $\square$

Based on Lemma 5.7, for each edge $(J', J) \in E$, we refer to job J' as the *parent* of job J , and refer to J as a *child* of J' . By definition, if job J' is the parent of job J , J' is the job with the earliest starting deadline in $X(J)$. If $X(J) = \emptyset$ for a job J , we refer to J as a *root job*. If a job J does not have any child, we call it a *leaf job*.

We next show that to study the Profit-to-optimal span ratio for all the flag jobs $\mathcal{F}$, it is equivalent to study the Profit-to-optimal span ratio for any tree in the graph $G(\mathcal{F}, E)$.

Lemma 5.8. *Given a job $J \in \mathcal{F}$, for each job $J' \in X(J)$, there exists a path from J' to J in the graph $G(\mathcal{F}, E)$.*

Proof: Suppose job J is in a tree T of $G(\mathcal{F}, E)$. Let $J_1, J_2, \dots, J_q, J$ be the sequence of nodes along the path from the root job of tree T to J , i.e., J_1 is the root job of T and is the parent of J_2 , J_2 is the parent of J_3 , $\dots$, and J_q is the parent of J . By the definition of edges, we have $d(J) < d(J_q) < d(J_{q-1}) < \dots < d(J_1)$. We prove that $X(J) \subseteq \{J_1, J_2, \dots, J_q\}$ by contradiction.

Assume on the contrary that there exists a job $J' \in X(J)$ such that $J' \notin \{J_1, J_2, \dots, J_q\}$. Since J_q is the parent of J , J_q has the earliest starting deadline among all the jobs in $X(J)$. Thus, $d(J') > d(J_q)$.

We first consider the case that $d(J_i) < d(J') < d(J_{i-1})$ for some i . Since $J' \in X(J)$, we have $a(J') < d(J) + p(J)$. It follows from $d(J) < d(J_i)$ and Lemma 5.6 that $d(J) + p(J) < d(J_i) + p(J_i)$. As a result, $a(J') < d(J_i) + p(J_i)$ and thus $J' \in X(J_i)$. The existence of the edge (J_{i-1}, J_i) suggests that job J_{i-1} has the earliest starting deadline among all the jobs in $X(J_i)$. Therefore, $d(J_{i-1}) < d(J')$, which leads to a contradiction.

Next, we consider the case that $d(J_1) < d(J')$. Similarly, since $a(J') < d(J) + p(J) < d(J_1) + p(J_1)$, we have $J' \in X(J_1)$, but this contradicts the fact that $X(J_1) = \emptyset$ since J_1 is a root job.

Thus, it is proven that $X(J) \subseteq \{J_1, J_2, \dots, J_q\}$, which implies that there exists a path from each job $J' \in X(J)$ to job J . $\square$

Lemma 5.9. *If there is no path between two jobs J and J' in the graph $G(\mathcal{F}, E)$, the active intervals of J and J' can never overlap with each other by any scheduler.*

Proof: Without loss of generality, suppose $d(J) < d(J')$. If $a(J') < d(J) + p(J)$, then $J' \in X(J)$. According to Lemma 5.8, there exists a path from J' to J in the graph $G(\mathcal{F}, E)$, which leads to a contradiction. Therefore, we have $a(J') \geq d(J) + p(J)$, i.e., job J' arrives no earlier than the latest possible completion time of job J . Thus, their active intervals cannot overlap with each other by any scheduler. $\square$

Lemma 5.10. *The Profit-to-optimal span ratio for the flag jobs $\mathcal{F}$ is bounded by a constant C if the Profit-to-optimal span ratio for the flag jobs in any tree of the graph $G(\mathcal{F}, E)$ is bounded by C .*

Proof: According to Lemma 5.7, we can decompose the directed graph $G(\mathcal{F}, E)$ into a collection of trees $T_1(\mathcal{F}_1, E_1), T_2(\mathcal{F}_2, E_2), \dots, T_m(\mathcal{F}_m, E_m)$, where m is an integer and $\mathcal{F}_i$ is the set of flag jobs in tree T_i . For any two jobs $J \in \mathcal{F}_i$ and $J' \in \mathcal{F}_j$ ($i \neq j$), there is no path between them in $G(\mathcal{F}, E)$. By Lemma 5.9, the active intervals of J and J' cannot overlap with each other by any scheduler. This suggests that an optimal schedule for the job set $\mathcal{F}_i \cup \mathcal{F}_j$ can be obtained by simply combining the optimal schedules for $\mathcal{F}_i$ and $\mathcal{F}_j$, and thus $\text{span}_{\min}(\mathcal{F}_i \cup \mathcal{F}_j) = \text{span}_{\min}(\mathcal{F}_i) + \text{span}_{\min}(\mathcal{F}_j)$. Consequently, $\text{span}_{\min}(\mathcal{F}) = \sum_{i=1}^m \text{span}_{\min}(\mathcal{F}_i)$.

Note that by applying the Profit scheduler, all the flag jobs are started at their starting deadlines. Thus,

$$\begin{aligned}
 \text{span}_{\text{Profit}}(\mathcal{F}) &= \text{len} \left(\bigcup_{J \in \mathcal{F}} [d(J), d(J) + p(J)] \right) \\
 &\leq \sum_{i=1}^m \text{len} \left(\bigcup_{J \in \mathcal{F}_i} [d(J), d(J) + p(J)] \right) \\
 &= \sum_{i=1}^m \text{span}_{\text{Profit}}(\mathcal{F}_i).
 \end{aligned}$$

If the Profit-to-optimal span ratio for any $\mathcal{F}_i$ is bounded by C , i.e., $\text{span}_{\text{Profit}}(\mathcal{F}_i) \leq C \cdot \text{span}_{\min}(\mathcal{F}_i)$, it follows that

$$\text{span}_{\text{Profit}}(\mathcal{F}) \leq \sum_{i=1}^m (C \cdot \text{span}_{\min}(\mathcal{F}_i)) = C \cdot \text{span}_{\min}(\mathcal{F}).$$

Hence, the lemma is proven. $\square$

We now focus on the Profit-to-optimal span ratio for a tree of flag jobs. In the following, we prove an upper bound of $2 + \frac{1}{k} + \frac{1}{k-1}$ on the Profit-to-optimal span ratio for any tree by an induction on the *height* of the tree — the number of edges along the longest path between the root job and the leaf jobs in the tree.

First, consider a tree with a height of 0, i.e., there is only one job J in the tree. Then, the span induced by any scheduler is exactly $p(J)$. Thus, the Profit-to-optimal span ratio for such a tree is 1, which is bounded by $2 + \frac{1}{k} + \frac{1}{k-1}$.

Suppose that the Profit-to-optimal span ratio is bounded by $2 + \frac{1}{k} + \frac{1}{k-1}$ for any tree with a height no more than $(n-1)$. According to Lemma 5.10, this upper bound of the span ratio also holds for a directed graph consisting of a collection of trees, each with a height no more than $(n-1)$. We now show that this upper bound also holds for any tree T with a height of n .

Let $\mathcal{J}_T$ be the set of flag jobs in T . Consider an optimal schedule on $\mathcal{J}_T$ in which the active interval of each job $J \in \mathcal{J}_T$ is $I^*(J)$, i.e., $\text{span}_{\min}(\mathcal{J}_T) = \text{len} \left(\bigcup_{J \in \mathcal{J}_T} I^*(J) \right)$. We start by determining a lower bound on the span induced by this optimal schedule. Note that $\bigcup_{J \in \mathcal{J}_T} I^*(J)$ may consist of multiple contiguous time intervals. Let J_{root} be the root

 Figure 5.6: An optimal schedule on tree T

job of T and let I_S denote the contiguous interval that J_{root} 's active interval $I^*(J_{root})$ falls in. As illustrated in Figure 5.6, we define an interval $I_{middle} = [I_S^-, I^*(J_{root})^+)$, i.e., I_{middle} is left delimited by the starting point of I_S and it is right delimited by the ending point of J_{root} 's active interval. It is obvious that $I^*(J_{root}) \subseteq I_{middle}$ and thus $p(J_{root}) \leq \text{len}(I_{middle})$. Then, we define $\mathcal{J}_{middle}$ as the set of all the jobs $J \in \mathcal{J}_T$ satisfying $I^*(J) \subseteq I_{middle}$ or $I^*(J)^- < I_{middle}^+ < I^*(J)^+$, i.e., these jobs have their active intervals either fully contained in I_{middle} or crossing the ending point of I_{middle} . Obviously, $J_{root} \in \mathcal{J}_{middle}$. Let $\mathcal{J}_{left}$ be the set of all the jobs $J \in \mathcal{J}_T$ satisfying $I^*(J)^+ < I_{middle}^-$. Let $\mathcal{J}_{right}$ be the set of all the jobs other than $\mathcal{J}_{left}$ and $\mathcal{J}_{middle}$, i.e., $\mathcal{J}_{right} = \mathcal{J}_T \setminus (\mathcal{J}_{left} \cup \mathcal{J}_{middle})$. Apparently, each job $J \in \mathcal{J}_{right}$ satisfies $I^*(J)^- \geq I_{middle}^+ = I^*(J_{root})^+$. We can therefore establish a lower bound on the span induced by the optimal schedule as

$$\begin{aligned} \text{span}_{\min}(\mathcal{J}_T) &\geq \text{len}\left(\bigcup_{J \in \mathcal{J}_{left}} I^*(J)\right) + \text{len}(I_{middle}) + \text{len}\left(\bigcup_{J \in \mathcal{J}_{right}} I^*(J)\right) \\ &\geq \text{span}_{\min}(\mathcal{J}_{left}) + \text{len}(I_{middle}) + \text{span}_{\min}(\mathcal{J}_{right}), \end{aligned} \quad (5.3)$$

where $\text{span}_{\min}(\mathcal{J}_{left})$ and $\text{span}_{\min}(\mathcal{J}_{right})$ are the spans induced by the optimal schedulers on $\mathcal{J}_{left}$ and $\mathcal{J}_{right}$ respectively.

Next, we prove that the span induced by the Profit scheduler for $\mathcal{J}_T$ has the following upper bound:

$$\begin{aligned} \text{span}_{\text{Profit}}(\mathcal{J}_T) &\leq \left(2 + \frac{1}{k} + \frac{1}{k-1}\right) \cdot (\text{span}_{\min}(\mathcal{J}_{left}) + \text{len}(I_{middle}) + \text{span}_{\min}(\mathcal{J}_{right})). \end{aligned} \quad (5.4)$$

Note that all the jobs in $\mathcal{J}_T$ are flag jobs. Thus, by applying the Profit scheduler, each job in $\mathcal{J}_T$ is started at its starting deadline. It follows that

$$\begin{aligned}
 \text{span}_{\text{Profit}}(\mathcal{J}_T) &= \text{len} \left(\bigcup_{J \in \mathcal{J}_T} [d(J), d(J) + p(J)] \right) \\
 &\leq \text{len} \left(\bigcup_{J \in \mathcal{J}_{\text{left}}} [d(J), d(J) + p(J)] \right) + \text{len} \left(\bigcup_{J \in \mathcal{J}_{\text{middle}}} [d(J), d(J) + p(J)] \right) \\
 &\quad + \text{len} \left(\bigcup_{J \in \mathcal{J}_{\text{right}}} [d(J), d(J) + p(J)] \right) \\
 &= \text{span}_{\text{Profit}}(\mathcal{J}_{\text{left}}) + \text{span}_{\text{Profit}}(\mathcal{J}_{\text{middle}}) + \text{span}_{\text{Profit}}(\mathcal{J}_{\text{right}}).
 \end{aligned}$$

It can be shown that by applying the graph construction technique for the job set $\mathcal{J}_{\text{left}}$, $\mathcal{J}_{\text{left}}$ must form a collection of trees, each with a height no more than $(n - 1)$. Let $G'(\mathcal{J}_{\text{left}}, E_{\text{left}})$ be the graph constructed for $\mathcal{J}_{\text{left}}$. For each edge $(J, J') \in E_{\text{left}}$, by definition, we have $J \in X(J')$. It follows from Lemma 5.8 that there exists a path from J to J' in the tree T of jobs $\mathcal{J}_T$. Thus, every path in G' remains and cannot become shorter in T . Note that $J_{\text{root}} \notin \mathcal{J}_{\text{left}}$ is a root job that has a path to every node in T . Therefore, the longest path in T must be longer than that in G' . This implies that each tree in G' has a height at most $(n - 1)$. According to the induction hypothesis and Lemma 5.10, the Profit-to-optimal span ratio of $\mathcal{J}_{\text{left}}$ is bounded by $2 + \frac{1}{k} + \frac{1}{k-1}$, i.e., $\text{span}_{\text{Profit}}(\mathcal{J}_{\text{left}}) \leq (2 + \frac{1}{k} + \frac{1}{k-1}) \cdot \text{span}_{\min}(\mathcal{J}_{\text{left}})$. Similarly, we have $\text{span}_{\text{Profit}}(\mathcal{J}_{\text{right}}) \leq (2 + \frac{1}{k} + \frac{1}{k-1}) \cdot \text{span}_{\min}(\mathcal{J}_{\text{right}})$. What remains is to show that $\text{span}_{\text{Profit}}(\mathcal{J}_{\text{middle}}) \leq (2 + \frac{1}{k} + \frac{1}{k-1}) \cdot \text{len}(I_{\text{middle}})$.

Recall that each job in $\mathcal{J}_{\text{middle}}$ is started at its starting deadline by the Profit scheduler. This suggests that the starting time of each job $J \in \mathcal{J}_{\text{middle}}$ determined by the Profit scheduler must be no earlier than $I^*(J)^-$. Let $J_1, J_2, \dots, J_m$ be all the jobs in $\mathcal{J}_{\text{middle}} \setminus \{J_{\text{root}}\}$ sorted in an increasing order of their starting deadlines. Since J_{root} is the root job, by the definition of edges, J_{root} must have a later starting deadline than all the other jobs in $\mathcal{J}_{\text{middle}}$. As a result, we have $d(J_1) < d(J_2) < \dots < d(J_m) < d(J_{\text{root}})$. Let J_i be the highest-indexed job satisfying $d(J_i) \leq I^*(J_{\text{root}})^+$, if there exists at least one such job. For notational convenience, if such a job does not exist (i.e., $I^*(J_{\text{root}})^+ < d(J_1)$), we define $i = 0$.

We first show that the span induced by the Profit scheduler for jobs $\{J_1, J_2, \dots, J_i\}$ is at most $\text{len}(I_{\text{middle}}) + \frac{1}{k} \cdot p(J_{\text{root}})$. According to Lemma 5.6, the jobs $J_1, J_2, \dots, J_i$ must

be completed no later than time $d(J_i) + p(J_i)$. If $d(J_i) + p(J_i) \leq I^*(J_{root})^+$, all these jobs must have their active intervals fully contained in I_{middle} . Consequently, the span induced by the Profit scheduler for these jobs is at most $\text{len}(I_{middle})$. If $d(J_i) + p(J_i) > I^*(J_{root})^+$, the span induced by the Profit scheduler for jobs $\{J_1, J_2, \dots, J_i\}$ is at most

$$\text{len}\left([I_{middle}^-, d(J_i) + p(J_i))\right) = \text{len}(I_{middle}) + \text{len}\left([I^*(J_{root})^+, d(J_i) + p(J_i))\right). \quad (5.5)$$

Since $I^*(J_{root})^+ \in [d(J_i), d(J_i) + p(J_i))$, J_{root} must arrive before J_i completes in the Profit schedule. This implies that J_{root} is not profitable to J_i . If J_{root} arrives earlier than time $d(J_i)$, we have $p(J_{root}) > k \cdot p(J_i)$. It follows from $d(J_i) \leq I^*(J_{root})^+$ that

$$\begin{aligned} p(J_{root}) &> k \cdot (d(J_i) + p(J_i) - I^*(J_{root})^+) \\ &= k \cdot \text{len}\left([I^*(J_{root})^+, d(J_i) + p(J_i))\right). \end{aligned}$$

If J_{root} arrives during J_i 's active interval $[d(J_i), d(J_i) + p(J_i))$, we have $p(J_{root}) > k \cdot (d(J_i) + p(J_i) - a(J_{root}))$. Note that $a(J_{root}) \leq I^*(J_{root})^- < I^*(J_{root})^+$. Hence, we have

$$\begin{aligned} p(J_{root}) &> k \cdot (d(J_i) + p(J_i) - I^*(J_{root})^+) \\ &= k \cdot \text{len}\left([I^*(J_{root})^+, d(J_i) + p(J_i))\right). \end{aligned}$$

Therefore, by inequality (5.5), the span for the jobs $\{J_1, J_2, \dots, J_i\}$ is always bounded by $\text{len}(I_{middle}) + \frac{1}{k} \cdot p(J_{root})$.

We next show that if $i < m$, $p(J_{j+1}) > k \cdot p(J_j)$ holds for each $i + 1 \leq j \leq m - 1$, and $p(J_{root}) > k \cdot p(J_m)$. By the definition of $\mathcal{J}_{middle}$, we have $I^*(J_{j+1})^- < I^*(J_{root})^+$. It follows that $a(J_{j+1}) \leq I^*(J_{j+1})^- < I^*(J_{root})^+ < d(J_{i+1}) \leq d(J_j)$. Thus, when job J_j is started at its starting deadline $d(J_j)$, J_{j+1} must have arrived and is not profitable to J_j , which suggests that $p(J_{j+1}) > k \cdot p(J_j)$. Similarly, since $a(J_{root}) \leq I^*(J_{root})^- < I^*(J_{root})^+ < d(J_{i+1}) \leq d(J_m)$, J_{root} is not profitable to J_m . Hence, $p(J_{root}) > k \cdot p(J_m)$. Together, these relations suggest that $p(J_j) < \frac{1}{k^{m+1-j}} \cdot p(J_{root})$ for each $i + 1 \leq j \leq m$.

Therefore, the span induced by the Profit scheduler for $\mathcal{J}_{middle}$ satisfies

$$\begin{aligned}
 & \text{span}_{Profit}(\mathcal{J}_{middle}) \\
 & \leq \text{len} \left(\bigcup_{j=1}^i [d(J_j), d(J_j) + p(J_j)] \right) + \text{len} \left(\bigcup_{j=i+1}^m [d(J_j), d(J_j) + p(J_j)] \right) \\
 & \quad + \text{len} \left([d(J_{root}), d(J_{root}) + p(J_{root})] \right) \\
 & \leq \text{len}(I_{middle}) + \frac{1}{k} \cdot p(J_{root}) + \sum_{j=i+1}^m p(J_j) + p(J_{root}) \\
 & < \text{len}(I_{middle}) + \left(1 + \frac{1}{k}\right) \cdot p(J_{root}) + \left(\sum_{h=1}^{\infty} \frac{1}{k^h}\right) \cdot p(J_{root}) \\
 & = \text{len}(I_{middle}) + \left(1 + \frac{1}{k} + \frac{1}{k-1}\right) \cdot p(J_{root}) \\
 & \leq \left(2 + \frac{1}{k} + \frac{1}{k-1}\right) \cdot \text{len}(I_{middle}).
 \end{aligned}$$

Hence, inequality (5.4) is proven. According to inequality (5.3), the Profit-to-optimal span ratio for the flag jobs $\mathcal{J}_T$ in a tree T with a height of n is bounded by $2 + \frac{1}{k} + \frac{1}{k-1}$. By induction, we can conclude that the Profit-to-optimal span ratio for any tree of flag jobs is bounded by $2 + \frac{1}{k} + \frac{1}{k-1}$.

By Lemma 5.10, the Profit-to-optimal span ratio for all the flag jobs $\mathcal{F}$ is bounded by $2 + \frac{1}{k} + \frac{1}{k-1}$. It is apparent that the minimum possible span of any job set $\mathcal{J}$ is no less than that of its flag jobs $\mathcal{F}$ designated by the Profit scheduler, i.e., $\text{span}_{min}(\mathcal{J}) \geq \text{span}_{min}(\mathcal{F})$. Thus, by Lemma 5.5, we have

$$\frac{\text{span}_{Profit}(\mathcal{J})}{\text{span}_{min}(\mathcal{J})} \leq \frac{k \cdot \text{span}_{Profit}(\mathcal{F})}{\text{span}_{min}(\mathcal{F})} \leq 2k + 1 + \frac{k}{k-1} = 2k + 2 + \frac{1}{k-1}.$$

It is easy to derive that $2k + 2 + \frac{1}{k-1}$ has a minimum value of $4 + 2\sqrt{2} \approx 6.9$ when k is set to $1 + \frac{\sqrt{2}}{2}$.

Theorem 5.8. *For the Flexible Job Scheduling problem in the clairvoyant online setting, the Profit scheduler achieves a competitive ratio of $2k + 2 + \frac{1}{k-1}$. When setting $k = 1 + \frac{\sqrt{2}}{2}$, this scheduler is $(2\sqrt{2} + 4)$ -competitive.*

5.4 The Generalized MinUsageTime DBP Problem

In this section, we study a generalization version of MinUsageTime DBP where each item is given a laxity for its starting. Specifically, let each item r be specified by an arrival time $a(r)$, a starting deadline $d(r)$, a processing length $p(r)$ and a size $s(r)$, such that r has to be started between $a(r)$ and $d(r)$. Once started, item r must be placed into a bin and stay in the bin for a processing length $p(r)$ without interruption. The total size of the items placed in a bin can never exceed the bin capacity at any time. The target of the generalized MinUsageTime DBP problem remains to minimize the total usage time of all the bins used. In the following text, we study the online version of generalized MinUsageTime DBP by applying the schedulers proposed in the previous sections and the online packing algorithms proposed in Chapter 3 together. We provide the competitive analysis of each algorithm for the problem in the non-clairvoyant online and clairvoyant online settings separately.

For the generalized MinUsageTime DBP problem, two lower bounds on the optimal solution can be easily established given a set of items $\mathcal{R}$:

Proposition 5.1. $OPT_{GDBP}(\mathcal{R}) \geq \sum_{r \in \mathcal{R}} (s(r) \cdot p(r))$.

Proposition 5.2. $OPT_{GDBP}(\mathcal{R}) \geq \text{span}_{\min}(\mathcal{R})$.

We first consider the generalized MinUsageTime DBP problem in the non-clairvoyant online setting. Given an item set $\mathcal{R}$, we apply the Batch+ scheduler to determine the starting times of the items and the First Fit packing algorithm to pack the items into bins. We next show that this strategy achieves a competitive ratio between $2\mu + 1$ and $2\mu + 3$, where μ represents the max/min item processing length ratio.

From (3.8) in Chapter 3, we know that given an instance $\mathcal{R}'$ of the MinUsageTime DBP problem, the total bin usage time by applying the First Fit packing algorithm is bounded by

$$(\mu + 2) \cdot \sum_{r \in \mathcal{R}'} (s(r) \cdot \text{len}(I(r))) + \text{span}(\mathcal{R}').$$

Let $BF(\mathcal{R})$ denote the total bin usage time by applying the Batch+ scheduler and First Fit packing for an instance $\mathcal{R}$. Then, by the result of Theorem 5.3, we have

$$\begin{aligned}
 BF(\mathcal{R}) &\leq (\mu + 2) \cdot \sum_{r \in \mathcal{R}} (s(r) \cdot p(r)) + \text{span}_{\text{Batch}+}(\mathcal{R}) \\
 &\leq (\mu + 2) \cdot \sum_{r \in \mathcal{R}} (s(r) \cdot p(r)) + (\mu + 1) \cdot \text{span}_{\min}(\mathcal{R}) \\
 &\leq (2\mu + 3) \cdot \text{OPT}_{GDBP}(\mathcal{R}),
 \end{aligned}$$

where the last inequality follows from the bounds given in Propositions 5.1 and 5.2.

Now, we construct an instance to show that the competitive ratio of applying the Batch+ scheduler and First Fit packing together is no less than $2\mu + 1$. Consider two groups of items. One group contains m short items, each with a size $1 - \frac{1}{2m}$, a laxity of 0 and a processing length of 1. For each $1 \leq i \leq m$, the i th short item arrives at time $t = (i - 1)(\mu + 1)$. The other group contains $2m$ long items, each with a size $\frac{1}{2m}$, a starting deadline of $m(\mu + 1)$ and a processing length of μ . For each $1 \leq i \leq m$, the $(2i - 1)$ th and the $(2i)$ th long items arrive at the same time $t = (i - 1)(\mu + 1) + (1 - \varepsilon)$, where $\varepsilon > 0$ can be arbitrarily small.

By applying Batch+ and First Fit packing, the i th short item and the $(2i - 1)$ th and the $(2i)$ th long items are started in the i th iteration. The i th short item and the $(2i - 1)$ th long item are packed into the same bin and the $(2i)$ th long item is packed into another bin. Thus, a total bin usage time of $m(2\mu + 1 - \varepsilon)$ is induced. A better solution is to let all the long items be started at their same starting deadline and packed into one bin. Let each short item be started when it arrives and packed into a separate bin. This strategy yields a total bin usage time of $m + \mu$. Therefore, the competitive ratio is at least $\frac{m(2\mu+1-\varepsilon)}{m+\mu}$, which can be made arbitrarily close to $2\mu + 1$ as m goes towards infinity.

For the generalized MinUsageTime DBP problem in the clairvoyant online setting, we can apply the classify-by-duration strategy to classify all the items $\mathcal{R}$ into categories upon their arrivals. For each item category $\mathcal{R}_i$, we apply the Profit scheduler to determine the starting times of the items and the First Fit packing algorithm to pack the items. Let α be the max/min item processing length ratio for each item category. Then, we can

bound the total bin usage time of this strategy by

$$\begin{aligned}
 CPF(\mathcal{R}) &\leq \sum_{i=1}^{\lceil \log_\alpha \mu \rceil + 1} \left((\alpha + 2) \cdot \sum_{r \in \mathcal{R}_i} (s(r) \cdot p(r)) + \text{span}_{Profit}(\mathcal{R}_i) \right) \\
 &\leq \sum_{i=1}^{\lceil \log_\alpha \mu \rceil + 1} \left((\alpha + 2) \cdot \sum_{r \in \mathcal{R}_i} (s(r) \cdot p(r)) + (2\sqrt{2} + 4) \cdot \text{span}_{min}(\mathcal{R}_i) \right) \\
 &= (\alpha + 2) \cdot \sum_{i=1}^{\lceil \log_\alpha \mu \rceil + 1} \left(\sum_{r \in \mathcal{R}_i} (s(r) \cdot p(r)) \right) + (2\sqrt{2} + 4) \cdot \sum_{i=1}^{\lceil \log_\alpha \mu \rceil + 1} \text{span}_{min}(\mathcal{R}_i) \\
 &\leq (\alpha + 2) \cdot \sum_{r \in \mathcal{R}} (s(r) \cdot p(r)) + (2\sqrt{2} + 4) \cdot \sum_{i=1}^{\lceil \log_\alpha \mu \rceil + 1} \text{span}_{min}(\mathcal{R}) \\
 &\leq (\alpha + 2) \cdot OPT_{GDBP}(\mathcal{R}) + (2\sqrt{2} + 4) \cdot (\lceil \log_\alpha \mu \rceil + 1) \cdot OPT_{GDBP}(\mathcal{R}) \\
 &= \left(\alpha + (2\sqrt{2} + 4) \cdot \lceil \log_\alpha \mu \rceil + 2\sqrt{2} + 6 \right) \cdot OPT_{GDBP}(\mathcal{R}),
 \end{aligned}$$

where the last inequality follows from the bounds given in Propositions 5.1 and 5.2.

5.5 Summary

In this chapter, we have studied the Flexible Job Scheduling problem in the online settings. In the non-clairvoyant online setting, we have established a lower bound of μ on the competitive ratio of any deterministic online scheduler, where μ is the max/min job processing length ratio among all the jobs to schedule. We have proposed two $O(\mu)$ -competitive online schedulers: Batch and Batch+. We have further considered a special case in which all the jobs have the same laxity. We have shown that no deterministic online scheduler can achieve a competitive ratio less than 2, and the Batch scheduler can achieve a competitive ratio of 3 in this case. In the clairvoyant online setting, we have established a lower bound of 2 on the competitive ratio of any deterministic online scheduler. We have proposed two $O(1)$ -competitive schedulers: Classify-by-Duration Batch+ and Profit. Finally, we have shown that the schedulers proposed for the Flexible Job Scheduling problem can be used to generalize the MinUsageTime DBP problem to model flexible jobs.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis, we have studied three combinatorial optimization problems related to jobs scheduling to minimize server usage time, which are the MinUsageTime DBP problem, the Energy-Efficient Job Scheduling problem and the Flexible Job Scheduling problem. These problems are motivated by issues arising from cloud computing and energy-efficient computing. We summarize the major contributions of the thesis in Figure 6.1.

6.2 Future Directions

The research described in this thesis has attracted some follow-up work. Azar and Vainstein [12] recently established a lower bound of $\Omega(\sqrt{\log \mu})$ on the competitive ratio of any deterministic online packing algorithm for the MinUsageTime DBP problem in the clairvoyant online setting and gave a matching $O(\sqrt{\log \mu})$ -competitive Hybrid algorithm to close the gap. Concurrent and independent to our work, Koehler and Khuller [49] proposed a 5-competitive Doubler scheduler for the Flexible Job Scheduling problem in the clairvoyant online setting.

Two results of the MinUsageTime DBP problem and the Flexible Job Scheduling problem can still be possibly improved. As introduced in Section 3.4.2, we know that for the MinUsageTime DBP problem, the approximation factor of 4 is mostly due to the overlapping between items placed inside the demand chart by applying the 2-allocation technique. An open question is whether there exist some better strategies to apply the

	MinUsageTime DBP	Energy-Efficient Job Scheduling	Flexible Job Scheduling
non-clairvoyant online setting	Lower Bound: μ (for any deterministic online algorithm, Li et al. [55]) Upper Bound: $\mu + 3$ (the First Fit packing algorithm, Section 3.2.2)	Lower Bound: μ (for any deterministic online algorithm, Li et al. [55]) Upper Bound: $\mu + 9$ (the Modified First Fit algorithm, Section 4.3.3)	Lower Bound: μ (for any deterministic online algorithm, Section 5.2.1) Upper Bound: $\mu + 1$ (the Batch+ scheduler, Section 5.2.2)
clairvoyant online setting	Lower Bound: $\Omega(\sqrt{\frac{\log \mu}{\log \log \mu}})$ (for any deterministic online algorithm, Section 3.3.1) Upper Bound: $O(\log \mu)$ (the Classify-by-Duration First Fit packing algorithm, Section 3.3.2)	Lower Bound: $\Omega(\sqrt{\log \mu})$ (for any deterministic online algorithm, Azar and Vainstein, [12]) Upper Bound: $O(\sqrt{\log \mu})$ (Section 4.3.4)	Lower Bound: 2 (for any deterministic online algorithm, Section 5.3.1) Upper Bound: $4 + 2\sqrt{2}$ (the Profit scheduler, Section 5.3.3)
offline setting	Upper Bound: 4 (the Dual Coloring algorithm, Section 3.4.2)	Upper Bound: 5 (Section 4.3.2)	Polynomial-time solvable via dynamic programming (Khandekar et al. [45])

Figure 6.1: A summary of the competitiveness and approximation bounds

2-allocation technique and thus achieve a better approximation factor. This question deserves to be carefully thought for further studying the MinUsageTime DBP problem. Besides, for the Flexible Job Scheduling problem in the clairvoyant online setting, we have known that the competitiveness of this problem has a lower bound of 2. The best known upper bound is 5 [49]. Closing the gap is an interesting issue for further study.

Besides, in this thesis, we have focused on studying the three problems with deterministic algorithms. For each problem, we have established the lower bound on the competitive ratio of any deterministic online algorithm and we propose deterministic online and approximation algorithms. On the other hand, for many combinatorial problems, randomized algorithms are also commonly used and they may provide better solutions. In our context of job scheduling, randomization can be performed on the machine to assign when there are multiple machines available to process a job. Studying the three problems by means of randomization is an interesting direction for further work. Moreover, the bounds we have derived represent the worst case performance of the algorithms relative to the optimum across all the possible inputs to the problems. Conducting simulation evaluation for particular workload inputs is another valuable direction for further study.

We further suggest the following two optimization problems to be addressed in the future work.

1. **Throughput Maximization Problem with Bounded Usage Time** Given a set of interval jobs $\mathcal{J}$ and a budget of machine usage time T , each job in $\mathcal{J}$ is associated with a size, an arrival time, a departure time and a weight. A subset of jobs $\mathcal{J}'$ is to be selected from $\mathcal{J}$ and scheduled onto machines such that the total usage time of the machines used does not exceed T . The target is to maximize the total weight of the selected jobs.
2. **A Skyline Variant of the MinUsageTime DBP Problem** A set of jobs $\mathcal{J}$ are to be scheduled onto the machines. Each job $J \in \mathcal{J}$ is specified by a size $s(J)$, an arrival time and a departure time. Once a job arrives, it must be placed onto a machine immediately and run for the period from its arrival to its departure without interruption. The total size of the jobs placed on any machine cannot exceed the machine capacity at any time. Besides, a function $h : \mathcal{J} \rightarrow \mathbb{R}$ has to be decided such that $[h(J_i), h(J_i) + s(J_i)) \cap [h(J_j), h(J_j) + s(J_j)) = \emptyset$ for any two overlapping jobs $J_i, J_j \in \mathcal{J}$ placed onto the same machine. Same as the MinUsageTime DBP problem, the target is to minimize the total usage time of the machines used.

The first optimization problem is a natural dual problem of the MinUsageTime DBP problem and can be used to model the following issue: given a budget on renting the cloud servers for processing jobs, how to select a subset of most valuable jobs and schedule them onto the cloud servers, such that the total cost does not exceed the budget? Unfortunately, no results on the general case of this problem are available so far. The second optimization problem is a skyline variant of the MinUsageTime DBP problem that views the computational resources on a machine as a spectrum. Each job must occupy a continuous “range” of computational resources for execution. It is easy to see that this resource model stems from the Dynamic Storage Allocation problem [38]. We shall consider this problem in the future.

Publications

1. Xueyan Tang, Yusen Li, **Runtian Ren**, Wentong Cai, “On First Fit Bin Packing for Online Cloud Server Allocation”, In *Proceedings of the 30th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 323–332, 2016.
2. **Runtian Ren**, Xueyan Tang, “Clairvoyant Dynamic Bin Packing for Job Scheduling with Minimum Server Usage Time”, In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pp. 227–237, 2016.
3. **Runtian Ren**, Xueyan Tang, Yusen Li, Wentong Cai, “Competitiveness of Dynamic Bin Packing for Online Cloud Server Allocation”, In *IEEE/ACM Transactions on Networking*, vol. 25, no. 3, pp. 1324–1331, 2017.
4. **Runtian Ren**, Xueyan Tang, “Online Flexible Job Scheduling for Minimum Span”, In *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pp. 55–66, 2017.

References

- [1] “Amazon EC2 pricing,” <http://aws.amazon.com/ec2/pricing/>.
- [2] “Gaikai,” <http://www.gaikai.com/>.
- [3] “Onlive,” <http://www.onlive.com/>.
- [4] S. Albers, “On energy conservation in data centers,” in *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2017, pp. 35–44.
- [5] M. Alicherry and R. Bhatia, “Line system design and a generalized coloring problem,” in *Proceedings of the 11th Annual European Symposium on Algorithms (ESA)*, 2003, pp. 19–30.
- [6] E. M. Arkin and E. B. Silverberg, “Scheduling jobs with fixed start and end times,” *Discrete Applied Mathematics*, vol. 18, no. 1, pp. 1–8, 1987.
- [7] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica *et al.*, “Above the clouds: A berkeley view of cloud computing,” Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Tech. Rep., 2009.
- [8] J. Augustine, S. Irani, and C. Swamy, “Optimal power-down strategies,” in *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2004, pp. 530–539.
- [9] B. Awerbuch, Y. Azar, and S. Plotkin, “Throughput-competitive on-line routing,” in *Proceedings of the 34th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, 1993, pp. 32–40.

REFERENCES

- [10] Y. Azar, N. Ben-Aroya, N. R. Devanur, and N. Jain, “Cloud scheduling with setup cost,” in *Proceedings of the 25th annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2013, pp. 298–304.
- [11] Y. Azar and S. Cohen, “An improved algorithm for online machine minimization,” *Operations Research Letters*, vol. 46, no. 1, pp. 128–133, 2018.
- [12] Y. Azar and D. Vainstein, “Tight bounds for clairvoyant dynamic bin packing,” in *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2017, pp. 77–86.
- [13] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. S. Naor, and B. Schieber, “A unified approach to approximating resource allocation and scheduling,” *Journal of the ACM*, vol. 48, no. 5, pp. 735–744, Sep. 2001.
- [14] A. Bar-Noy, R. Canetti, S. Kutten, Y. Mansour, and B. Schieber, “Bandwidth allocation with preemption,” in *Proceedings of the 27th Annual ACM Symposium on Theory of Computing (STOC)*, 1995, pp. 616–625.
- [15] A. Bar-Noy, S. Guha, J. Naor, and B. Schieber, “Approximating the throughput of multiple machines in real-time scheduling,” *SIAM Journal on Computing*, vol. 31, no. 2, pp. 331–352, 2001.
- [16] L. A. Barroso and U. Hölzle, “The case for energy-proportional computing,” *Computer*, vol. 40, no. 12, pp. 33–37, 2007.
- [17] S. Baruah, G. Koren, D. Mao, B. Mishra, A. Raghunathan, L. Rosier, D. Shasha, and F. Wang, “On the competitiveness of on-line real-time task scheduling,” in *Proceedings of the 12th IEEE Real-Time Systems Symposium (RTSS)*, 1991, pp. 106–115.
- [18] P. Berman and B. DasGupta, “Improvements in throughput maximization for real-time scheduling,” in *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC)*, 2000, pp. 680–687.

REFERENCES

- [19] R. Bhatia, J. Chuzhoy, A. Freund, and J. S. Naor, “Algorithmic aspects of bandwidth trading,” *ACM Transactions on Algorithms*, vol. 3, no. 1, p. 10, 2007.
- [20] A. Borodin and R. El-Yaniv, *Online Computation and Competitive Analysis*. Cambridge University Press, 2005.
- [21] K. I. Bouzina and H. Emmons, “Interval scheduling on identical machines,” *Journal of Global Optimization*, vol. 9, no. 3-4, pp. 379–393, 1996.
- [22] A. L. Buchsbaum, H. Karloff, C. Kenyon, N. Reingold, and M. Thorup, “Opt versus load in dynamic storage allocation,” *SIAM Journal on Computing*, vol. 33, no. 3, pp. 632–646, 2004.
- [23] G. Calinescu, A. Chakrabarti, H. Karloff, and Y. Rabani, “Improved approximation algorithms for resource allocation,” in *Proceedings of the 9th International Conference on Integer Programming and Combinatorial Optimization (IPCO)*, 2002, pp. 401–414.
- [24] J. W.-T. Chan, T.-W. Lam, and P. W. Wong, “Dynamic bin packing of unit fractions items,” *Theoretical Computer Science*, vol. 409, no. 3, pp. 521–529, 2008.
- [25] J. Chang, S. Khuller, and K. Mukherjee, “Lp rounding and combinatorial algorithms for minimizing active and busy time,” *Journal of Scheduling*, vol. 20, no. 6, pp. 657–680, 2017.
- [26] B. Chen, R. Hassin, and M. Tzur, “Allocation of bandwidth and storage,” *IIE Transactions*, vol. 34, no. 5, pp. 501–507, 2002.
- [27] L. Chen, N. Megow, and K. Schewior, “An $o(m^2 \log m)$ -competitive algorithm for online machine minimization,” in *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2016, pp. 155–163.
- [28] J. Chuzhoy, S. Guha, S. Khanna, and J. S. Naor, “Machine minimization for scheduling jobs with interval constraints,” in *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2004, pp. 81–90.

REFERENCES

- [29] J. Chuzhoy and J. S. Naor, “New hardness results for congestion minimization and machine scheduling,” *Journal of the ACM*, vol. 53, no. 5, pp. 707–721, 2006.
- [30] J. Chuzhoy, R. Ostrovsky, and Y. Rabani, “Approximation algorithms for the job interval selection problem and related scheduling problems,” *Mathematics of Operations Research*, vol. 31, no. 4, pp. 730–738, 2006.
- [31] E. G. Coffman, Jr, M. R. Garey, and D. S. Johnson, “Dynamic bin packing,” *SIAM Journal on Computing*, vol. 12, no. 2, pp. 227–258, 1983.
- [32] N. Devanur, K. Makarychev, D. Panigrahi, and G. Yaroslavtsev, “Online algorithms for machine minimization,” *arXiv preprint arXiv:1403.0486*, 2014.
- [33] U. Faigle and W. M. Nawijn, “Note on scheduling intervals on-line,” *Discrete Applied Mathematics*, vol. 58, no. 1, pp. 13–17, 1995.
- [34] U. Feige, “A threshold of $\ln n$ for approximating set cover,” *Journal of the ACM*, vol. 45, no. 4, pp. 634–652, 1998.
- [35] M. Flammini, G. Monaco, L. Moscardelli, H. Shachnai, M. Shalom, T. Tamir, and S. Zaks, “Minimizing total busy time in parallel scheduling with application to optical networks,” *Theoretical Computer Science*, vol. 411, no. 40-42, pp. 3553–3562, 2010.
- [36] J. A. Garay, I. S. Gopal, S. Kutten, Y. Mansour, and M. Yung, “Efficient on-line call control algorithms,” *Journal of Algorithms*, vol. 23, no. 1, pp. 180–194, 1997.
- [37] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*. W.H. Freeman & Company, 1979.
- [38] J. Gergov, “Approximation algorithms for dynamic storage allocation,” in *Proceedings of the 4th European Symposium on Algorithms (ESA)*, 1996, pp. 52–61.
- [39] —, “Algorithms for compile-time memory optimization,” in *Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 1999, pp. 907–908.

REFERENCES

- [40] W. Horn, “Some simple scheduling algorithms,” *Naval Research Logistics*, vol. 21, no. 1, pp. 177–185, 1974.
- [41] C.-Y. Huang, K.-T. Chen, D.-Y. Chen, H.-J. Hsu, and C.-H. Hsu, “GamingAnywhere: the first open source cloud gaming system,” *ACM Transactions on Multimedia Computing, Communications, and Applications*, vol. 10, no. 1, p. 10, 2014.
- [42] S. Irani, S. Shukla, and R. Gupta, “Online strategies for dynamic power management in systems with multiple power-saving states,” *ACM Transactions on Embedded Computing Systems*, vol. 2, no. 3, pp. 325–346, 2003.
- [43] Z. Ivkovic and E. L. Lloyd, “Fully dynamic algorithms for bin packing: Being (mostly) myopic helps,” *SIAM Journal on Computing*, vol. 28, no. 2, pp. 574–611, 1998.
- [44] A. R. Karlin, M. S. Manasse, L. A. McGeoch, and S. Owicki, “Competitive randomized algorithms for nonuniform problems,” *Algorithmica*, vol. 11, no. 6, pp. 542–571, 1994.
- [45] R. Khandekar, B. Schieber, H. Shachnai, and T. Tamir, “Real-time scheduling to minimize machine busy times,” *Journal of Scheduling*, vol. 18, no. 6, pp. 561–573, 2015.
- [46] S. Khuller, J. Li, and B. Saha, “Energy efficient scheduling via partial shutdown,” in *Proceedings of the 21st annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2010, pp. 1360–1372.
- [47] H. Kierstead, “The linearity of first-fit coloring of interval graphs,” *SIAM Journal on Discrete Mathematics*, vol. 1, no. 4, pp. 526–530, 1988.
- [48] —, “A polynomial time approximation algorithm for dynamic storage allocation,” *Discrete Mathematics*, vol. 88, no. 2-3, pp. 231–237, 1991.
- [49] F. Koehler and S. Khuller, “Busy time scheduling on a bounded number of machines,” in *Proceedings of the 15th Algorithms and Data Structures Symposium (WADS)*, 2017, pp. 521–532.

- [50] V. Kumar and A. Rudra, “Approximation algorithms for wavelength assignment,” in *Proceedings of the 25th International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, 2005, pp. 152–163.
- [51] 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*, vol. 26, no. 1, pp. 125–133, 1990.
- [52] E. L. Lawler, J. K. Lenstra, A. H. Rinnooy Kan, and D. B. Shmoys, “Sequencing and scheduling: Algorithms and complexity,” *Handbooks in Operations Research and Management Science*, vol. 4, pp. 445–522, 1993.
- [53] J. Li and S. Khuller, “Generalized machine activation problems,” in *Proceedings of the 22nd annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2011, pp. 80–94.
- [54] Y. Li, Y. Deng, R. Seet, X. Tang, and W. Cai, “MASTER: Multi-platform application streaming toolkits for elastic resources,” in *Proceedings of the 23rd ACM International Conference on Multimedia (MM)*, 2015, pp. 805–806.
- [55] Y. Li, X. Tang, and W. Cai, “On dynamic bin packing for resource allocation in the cloud,” in *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2014, pp. 2–11.
- [56] —, “Dynamic bin packing for on-demand cloud resource allocation,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 1, pp. 157–170, 2016.
- [57] R. J. Lipton and A. Tomkins, “Online interval scheduling,” in *Proceedings of the 5th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 1994, pp. 302–311.
- [58] M. G. Luby, J. Naor, and A. Orda, “Tight bounds for dynamic storage allocation,” *SIAM Journal on Discrete Mathematics*, vol. 9, no. 1, pp. 155–166, 1996.
- [59] G. B. Mertzios, M. Shalom, A. Voloshin, P. W. Wong, and S. Zaks, “Optimizing busy time on parallel machines,” *Theoretical Computer Science*, vol. 562, pp. 524–541, 2015.

REFERENCES

- [60] J. S. Naor, A. Orda, and Y. Petruschka, “Dynamic storage allocation with known durations,” *Discrete Applied Mathematics*, vol. 100, no. 3, pp. 203–213, 2000.
- [61] C. A. Phillips, C. Stein, E. Torng, and J. Wein, “Optimal time-critical scheduling via resource augmentation,” in *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC)*, 1997, pp. 140–149.
- [62] C. A. Phillips, R. Uma, and J. Wein, “Off-line admission control for general scheduling problems,” *Journal of Scheduling*, vol. 3, no. 6, pp. 365–381, 2000.
- [63] P. Raghavan and C. D. Tompson, “Randomized rounding: a technique for provably good algorithms and algorithmic proofs,” *Combinatorica*, vol. 7, no. 4, pp. 365–374, 1987.
- [64] J. Robson, “An estimate of the store size necessary for dynamic storage allocation,” *Journal of the ACM*, vol. 18, no. 3, pp. 416–423, 1971.
- [65] —, “Bounds for some functions concerning dynamic storage allocation,” *Journal of the ACM*, vol. 21, no. 3, pp. 491–499, 1974.
- [66] —, “Worst case fragmentation of first fit and best fit storage allocation strategies,” *The Computer Journal*, vol. 20, no. 3, pp. 242–244, 1977.
- [67] P. E. Ross, “Cloud computing’s killer app: Gaming,” *IEEE Spectrum*, vol. 46, no. 3, pp. 14–14, 2009.
- [68] B. Saha, “Renting a cloud,” in *Proceedings of the 33rd International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, 2013, p. 437.
- [69] S. S. Seiden, “Randomized online interval scheduling,” *Operations Research Letters*, vol. 22, no. 4, pp. 171–177, 1998.
- [70] M. Shalom, A. Voloshin, P. W. Wong, F. C. Yung, and S. Zaks, “Online optimization of busy time on parallel machines,” *Theoretical Computer Science*, vol. 560, pp. 190–206, 2014.

REFERENCES

- [71] R. Shea, J. Liu, E.-H. Ngai, and Y. Cui, “Cloud gaming: Architecture and performance,” *IEEE Network*, vol. 27, no. 4, pp. 16–21, 2013.
- [72] M. Slusarek, “A coloring algorithm for interval graphs,” in *Proceedings of the 14th International Symposium on Mathematical Foundations of Computer Science (MFCS)*. Springer, 1989, pp. 471–480.
- [73] W. Van Heddeghem, S. Lambert, B. Lannoo, D. Colle, M. Pickavet, and P. Demeester, “Trends in worldwide ict electricity consumption from 2007 to 2012,” *Computer Communications*, vol. 50, pp. 64–76, 2014.
- [74] V. V. Vazirani, *Approximation Algorithms*. Springer Science & Business Media, 2013.
- [75] D. P. Williamson and D. B. Shmoys, *The Design of Approximation Algorithms*. Cambridge university press, 2011.
- [76] P. Winkler and L. Zhang, “Wavelength assignment and generalized interval graph coloring,” in *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2003, pp. 830–831.