

THE IMPACT OF RESTARTS IN ONLINE MACHINE SCHEDULING

DISSERTATION

zur Erlangung des Grades eines Doktors
der Naturwissenschaften

vorgelegt von

Aflatoun Amouzandeh, M.Sc. (Computer Science)

eingereicht bei der Naturwissenschaftlich-Technischen Fakultät
der Universität Siegen

Siegen 2025

Betreuer und erster Gutachter
Prof. Dr. Rob van Stee
Universität Siegen

Zweiter Gutachter
Prof. Dr. Markus Lohrey
Universität Siegen

Tag der mündlichen Prüfung
18. Februar 2026

Abstract

This dissertation investigates the impact of job preemptions and restarts in online machine scheduling, with a focus on three fundamental objectives. An online scheduling algorithm determines whether to process a job or not without knowing the future events. One important extension to the classical online scheduling framework is the use of *restarts*. Allowing restarts means that the processing of a job may be interrupted, but in this case the interrupted job loses all its previous progress and must be started again later, until it is completed without interruptions. This model also known as *preemption with restarts*. Although restarts can potentially be very beneficial in the context of online scheduling, there has been relatively little research on this topic up until now.

First, we study the problem of minimizing the total completion time on a single machine. We present a simple deterministic online algorithm that achieves a competitive ratio better than 1.4568, improving upon the previous best bound of $3/2$. The algorithm follows an increasing-size processing order and interrupts a running job only when doing so reduces the completion time of an arriving job by more than a factor of 1.4568 compared to processing it immediately.

Second, we extend the results to the minimization of total weighted completion time. For the case of equal processing times, we design a deterministic online algorithm that achieves a competitive ratio better than 1.325 based on carefully structured decision intervals. The algorithm uses at most three restarts for the entire schedule and redefines these decision intervals in certain situations to avoid further interruptions.

Finally, we consider the weighted makespan minimization problem. This problem is generalization of the classical makespan minimization problem. We present a deterministic online algorithm that achieves a competitive ratio better than 1.3098 in the case where all jobs have equal processing times.

Zusammenfassung

Diese Dissertation untersucht die Auswirkungen von Unterbrechungen und Neustarts von Jobs in Online-Scheduling und konzentriert sich dabei auf drei grundlegende Zielfunktionen. Ein Online-Scheduling-Algorithmus entscheidet, wann ein Job ausgeführt wird oder nicht, ohne die zukünftigen Ereignisse zu kennen. Eine wichtige Erweiterung des klassischen Online-Scheduling-Modells ist die Verwendung von Neustarts. Neustarts erlauben es, die Ausführung eines Jobs zu unterbrechen; in diesem Fall verliert der unterbrochene Job jedoch sämtlichen bislang erzielten Fortschritt und muss zu einem späteren Zeitpunkt vollständig neu begonnen werden, bis er ohne Unterbrechung abgeschlossen ist. Dieses Modell ist auch als Preemption mit Neustarts bekannt. Obwohl Neustarts im Kontext des Online-Schedulings potenziell sehr vorteilhaft sein können, wurde dieses Thema bislang nur relativ wenig untersucht.

Zunächst betrachten wir das Problem der Minimierung der Summe der Abschlusszeiten auf einer einzelnen Maschine. Wir präsentieren einen einfachen deterministischen Online-Algorithmus, der ein Kompetitivitätsverhältnis besser als 1.4568 erreicht und damit die bisher beste Schranke von $3/2$ verbessert. Der Algorithmus folgt einer Bearbeitungsreihenfolge nach zunehmender Größe und unterbricht einen laufenden Auftrag nur dann, wenn dadurch die Fertigstellungszeit eines eintreffenden Auftrags im Vergleich zur sofortigen Bearbeitung um mehr als den Faktor 1.4568 reduziert wird.

Im zweiten Teil erweitern wir die Resultate auf die Minimierung der gewichteten gesamten Abschlusszeit. Für den Fall gleicher Bearbeitungszeiten entwickeln wir einen deterministischen Online-Algorithmus, ein Kompetitivitätsverhältnis besser als 1.325 erreicht, basierend auf sorgfältig strukturierten Entscheidungsintervallen. Der Algorithmus verwendet höchstens drei Neustarts über den gesamten Ablauf hinweg und definiert diese Entscheidungsintervalle in bestimmten Situationen gezielt neu, um weitere Unterbrechungen zu vermeiden.

Abschließend betrachten wir das Problem der Minimierung des gewichteten Makespans. Dieses Problem stellt eine Verallgemeinerung des klassischen Makespan-Minimierungsproblems dar. Wir präsentieren einen deterministischen Online-Algorithmus, der im Fall gleicher Bearbeitungszeiten einen Wettbewerbsfaktor besser als 1.3098 erreicht.

Acknowledgments

I would like to express my sincere gratitude to my supervisor, Prof. Dr. Rob van Stee, for his guidance and support throughout this research.

I would also like to acknowledge the Naturwissenschaftlich-Technische Fakultät of the University of Siegen for providing an excellent research environment and the necessary resources to conduct my work. Additionally, I am sincerely grateful for the financial support from the German Research Foundation (DFG), which made this work possible.

I am thankful to my colleagues, especially Sören Schmitt and Paul Hesse, for valuable discussions and collaboration. I am especially grateful for the valuable collaboration with Prof. Dr. Klaus Jansen, Lis Pirotton and Corinna Wambsganz, which greatly enriched this research.

I sincerely appreciate the support provided by Tina Heinz, and I am also thankful to Dr. Shafie Shokrani for guidance and discussions.

Finally, I wish to express my deepest thanks to my beloved family for their unlimited love and support and for always being there for me no matter the circumstances.

Contents

1	Introduction	1
1.1	Optimization under Uncertainty	2
1.2	Online Optimization Problems	3
1.3	Online Machine Scheduling Problem	5
1.3.1	Online Single Machine Scheduling	9
1.3.2	Related Results	13
1.4	Structure and Publications	14
2	Survey of Online Algorithms	17
2.1	Online Scheduling	17
2.2	Online Packing	19
2.3	Online Coloring	22
2.4	Online Matching	25
2.5	Allocations and Auctions	30
2.6	Advise and Predictions	33
2.7	Prophet Inequalities	36
2.8	Various Online Problems	38
3	Minimizing the Total Completion Time	45
3.1	Introduction	45
3.1.1	Our Results	47
3.2	Improved RSPT algorithm (IRSPT)	47
3.3	IRSPT-like algorithms	49
3.4	Lower Bound	50
3.5	Competitive analysis of IRSPT	52
4	Minimizing the Total Weighted Completion Time	67
4.1	Introduction	67
4.1.1	Our Results	68
4.2	Lower Bound	69
4.3	Algorithm LW-3R	70

4.4	Competitive analysis of LW-3R	72
5	Minimizing the Weighted Makespan	89
5.1	Introduction	89
5.1.1	Our Results	91
5.2	Lower Bound	91
5.3	Limited Largest Weight (LLW) algorithm	95
5.4	Competitive analysis of LLW	97
5.5	Offline Setting	104
6	Discussion and Open Problems	117
	References	119

CHAPTER 1

Introduction

Scheduling is a fundamental problem in computer science and operations research, concerned with allocating limited resources over time. Essentially, scheduling is about deciding which tasks should be executed according to time and resource availability constraints, in order to achieve specific objectives such as optimizing productivity, minimizing costs or maximizing throughput. The evolution of scheduling can be traced back to early civilizations and their efforts to manage time and resources efficiently, all the way to the modern computing era where scheduling problems have become more complex and dynamic.

The concept of scheduling took a more formalized shape during the Industrial Revolution (late 18th to early 19th century) with the advent of machinery and the need for efficient production, managing the flow of materials and labor became crucial for maximizing productivity. As factories grew larger and more complex, there was a need to assign tasks to workers and machines to meet the demands of mass production. During this period, Frederick Taylor introduced the principles of scientific management, including time and motion studies which aimed to optimizing worker efficiency. Later, in the 1910s, Henry Gantt developed the Gantt chart, a tool for visualizing and scheduling tasks. These innovations, along with the increasing importance of time and motion studies helped recognize scheduling problems as essential for maintaining efficient operations.

Operations Research evolved into a formalized scientific approach during World War II, when better decision-making in military logistics and training schedules became essential. During this time, optimization methods were formalized to improve resource allocation and task scheduling. As the com-

plexity of scheduling increased, especially with the growth of manufacturing and later in computer systems, the mathematical optimization began to play a significant role. Scheduling is an optimization problem, where the goal is to find the best solution for task allocation given certain constraints. Optimization techniques provided a framework to systematically solve these problems. With advancements in mathematical modeling and algorithms, the challenge of scheduling evolved into a discipline that could address more complex scenarios and constraints to achieve the most effective solutions.

Optimization problem An optimization problem P is defined by a set of inputs $\mathcal{I}$ and a cost function C . For each input $I \in \mathcal{I}$, there exists a set of feasible solutions $F(I)$. Each solution $J \in F(I)$ is associated with a positive real number $C(I, J)$, representing its objective function value.

Optimization problems are generally categorized into two types: maximization and minimization problems. This dissertation primarily focuses on minimization problems, where the goal is to minimize the cost associated with a solution.

Given an input I , an algorithm ALG produces a solution $\text{ALG}[I] \in F(I)$, with the cost of this solution denoted as $\text{ALG}(I) = C(I, \text{ALG}[I])$. An optimal algorithm OPT finds the best solution for any input, ensuring that for all feasible inputs I :

$$\text{OPT}(I) = \min_{J \in F(I)} C(I, J).$$

1.1 Optimization under Uncertainty

Optimization under uncertainty is a widely studied field with a wide range of applications. The key challenge in this field is making decisions or finding optimal solutions when the inputs to the problem are not fully known or are subject to variation. Uncertainty can be modeled in several ways and different approaches are used depending on the nature of the problem and the type of uncertainty involved.

One of the most common approaches is to make stochastic assumptions about the input. In this case, the uncertain parameters are modeled using

probability distributions. For example, the processing time of jobs might be considered as a random variable with a known distribution, such as a normal or exponential distribution, rather than a fixed value.

When the parameters are only known approximately, this situation is modeled by robust optimization. In robust optimization, the goal is to find solutions that perform well even under a range of possible variations of the uncertain parameters, ensuring that the solution is not overly sensitive to any specific estimate. This approach is particularly useful in scenarios where the uncertainty is too large or complex to model probabilistically and more general assumptions about the range of uncertainty must be made.

A survey on this topic was published by Bertsimas et al. [32], which provides a comprehensive overview of various methods and approaches used in robust optimization, along with their applications and challenges.

1.2 Online Optimization Problems

In many optimization problems, it is typically assumed that all relevant information is available in advance. However, in real-world scenarios this assumption often does not hold.

As an example, consider a busy office with only one printer serving all employees. This printer can handle only one print job at a time. Typically, printers process jobs in the order they arrive. This approach can result in delays, particularly when small or urgent jobs are queued behind larger ones.

Employee A sends a large document to print. The printer immediately begins working on this job. Shortly after, Employee B submits a small document that needs to be printed quickly. Because the printer is occupied, Employee B's request must wait patiently. Just as the printer finishes Employee A's lengthy job, Employee C sends another print request for a medium-sized but urgent document.

At this point, the printer's controller faces a decision. Should it print Employee B's document next to reduce waiting time, or to prioritize the urgent job submitted later? If the controller follows the simple first-come,

first-served rule, Employee C's urgent request will be delayed further, potentially causing problems for critical tasks.

An online problem can also be classified as *semi-online* when some partial information about future inputs is available in advance. This additional knowledge can help make better decisions, even though the full input sequence is not known. For instance, in an online scheduling problem, the jobs might be known to be ordered by their processing times, the processing times might fall within a restricted range, or even the minimum size job might be known ahead of time.

Online algorithms Optimization algorithms can be classified into offline and online algorithms. Offline algorithms have access to the entire input before computing a solution. Many offline optimization problems are NP-hard, meaning that finding the exact optimal solution is computationally infeasible. In contrast, online algorithms operate in environments where the input is revealed sequentially, requiring the algorithm to make irrevocable decisions at each step without knowledge of future inputs. Since they lack complete information, online algorithms must make decisions based only on the data available at the time.

Competitive analysis Competitive analysis is a worst-case performance measure used to evaluate online algorithms. It compares the performance of an online algorithm to an optimal offline algorithm that has complete knowledge of future inputs. This approach provides a worst-case guarantee on how well an online algorithm performs relative to the best possible strategy and it is a fundamental tool in the study of online algorithms.

The competitive ratio $\mathcal{R}$ of an online algorithm ALG is defined as

$$\mathcal{R} = \sup_I \frac{ALG(I)}{OPT(I)}.$$

This means that if an algorithm has a competitive ratio of $\mathcal{R}$, we have the guarantee that no matter how bad the input (from the point of view of the algorithm), the algorithm will never have cost more than $\mathcal{R}$ times the optimal cost. Note that the focus here is entirely on the lack of knowledge

about the input; the online algorithm may use as much computation as it needs to make its decisions.

The concept behind the competitive ratio can be viewed as the analysis of a two-player zero-sum game, where the online algorithm competes against the optimal offline algorithm, which plays the role of an adversary by selecting the worst-case input sequence. Consider again our previous example, a system with a single printer serving multiple users, where only one job can be processed at a time. The printer does not have prior knowledge of future job arrivals and their processing times or their priorities. Now, imagine an idealized printer that has complete knowledge of all job arrivals, processing times and priorities in advance. This printer can schedule the jobs optimally to minimize total weighted completion time and ensure that higher-priority jobs are appropriately prioritized.

The competitive ratio measures the performance of the real-time printer relative to this optimal offline scheduler. For instance, if the real printer requires 50 time units to complete all jobs while the optimal scheduler could complete them in 30 time units, the competitive ratio is $50/30 \approx 1.67$. This indicates that the online scheduler performs approximately 67% worse than the optimal offline solution.

To overcome the unrealistic power of the adversary, some models allow randomization in the online algorithm. The adversary knows the random process used by the online algorithm but not the outcome of that process. Such an oblivious adversary must commit to a sequence before the online algorithm begins.

1.3 Online Machine Scheduling Problem

The machine scheduling problem is an important challenge in the development of optimization algorithms and computational methods. The objective is to assign a set of tasks to a set of machines in a way that optimizes certain criteria, such as minimizing the total completion time, reducing waiting periods or balancing the workload among machines. This problem has wide-ranging applications, for example in manufacturing where products must be processed in a specific order, in computer systems where

tasks are distributed across processors, and in logistics where deliveries must be scheduled efficiently.

In offline machine scheduling scenarios, all tasks and resources are known in advance, allowing for careful planning and optimization. However, in real-world applications, information about tasks often arrives sequentially and decisions must be made without knowledge of future tasks. This brings us to the area of online machine scheduling. The online scheduling is particularly relevant in modern multi-core processors and cloud computing environments, where tasks must be efficiently distributed across CPUs to maximize performance. For example, in a CPU scheduler, processes arrive unpredictably, requiring real-time decisions about which task should be executed next. If a high-priority process arrives after the system has already committed resources to a lower-priority task, inefficiencies may arise. Similarly, in cloud computing, virtual machines must be assigned dynamically to available servers, ensuring that computational resources are utilized efficiently while maintaining quality-of-service guarantees.

In online machine scheduling, the arrival of jobs can be categorized into two main types: over time and over a list. Jobs arriving over time means that each job has a specific release time, after which it can be processed (as discussed in [151]). On the other hand, jobs arriving over a list means that a new job only arrives once the current job has been scheduled on the machine (as discussed in [7]). In this work, we focus solely on the scenario where all jobs arrive over time.

One of the earliest online algorithms was developed in the 1960s considered load balancing [82], although they were not widely studied until the early 1990s. Over the years, researchers have explored numerous objectives, each designed to address different practical needs and constraints. A collection of objective functions that will be discussed is as follows:

- **Makespan ($C_{\max}$):** The time required to complete all tasks. The objective is to ensure that the time at which all machines are finished is minimized.
- **Weighted Makespan ($WC_{\max}$):** The maximum weighted completion time of the jobs.

- **Total Completion Time ($\sum C_j$):** The sum of completion times for all tasks. This objective is important in environments where the timely completion of each task contributes to overall system performance. Note that minimizing $\sum C_j$ is equivalent to minimizing average completion time.
- **Weighted Completion Time ($\sum w_j C_j$):** Each task has a weight representing its importance. The aim is to minimize the weighted sum of completion times, balancing the need to prioritize more critical tasks while maintaining overall efficiency.
- **Total Flow Time ($\sum F_j$):** The sum of the time periods from the release of each task to its completion. This is significant in optimizing the turnaround time for tasks in a system, ensuring quicker processing and throughput.
- **Total Weighted Flow Time ($\sum w_j F_j$):** Each task has a weight representing its importance. The aim is to minimize the weighted sum of flow times.

Minimizing the maximum weighted flow time is equivalent to minimizing the weighted makespan when all jobs are available at time 0. The same relation exists between $\sum F_j$ and $\sum C_j$. Moreover, since the average completion time is simply a scaled version of the total completion time, the bounds for total completion time also apply to the average completion time. In certain cases, minimizing the total completion time can also help reduce the overall makespan, thereby linking these two performance metrics.

In scheduling theory, problems are commonly described using a three-field notation $\alpha|\beta|\gamma$ introduced by Graham et al. [82]. The first field (α) specifies the machine environment, such as a single machine or multiple parallel machines. The second field (β) describes any specific characteristics or constraints of the problem, such as whether tasks have deadlines. The third field (γ) denotes the objective function that needs to be optimized, such as minimizing the makespan, total completion time, or maximum lateness. In this thesis, we use this notation to categorize and discuss various machine scheduling problems.

We now formally define the general form of three problems considered in this work. A detailed description of each problem will be provided in the subsequent chapters.

Remark: For clarity and simplicity, jobs are occasionally referred to by their indices $1, 2, \dots, n$ instead of using the full notation $J_1, J_2, \dots, J_n$. These notations are equivalent throughout the text.

Definition 1.1. Online Machine Scheduling for Minimizing the Total Completion Time

- **Input.** A list $M = (M_1, M_2, \dots, M_m)$ of $m(\geq 1)$ machines, a sequence $J = J_1, \dots, J_n$ of $n(>> m)$ jobs are given, where each J_j is characterized by its processing time p_j and release time (arrival time) r_j , where $1 \leq j \leq n$.
- **Objective.** Minimization of $\sum_{j=1}^n C_j$.

Definition 1.2. Online Machine Scheduling for Minimizing the Total Weighted Completion Time

- **Input.** A list $M = (M_1, M_2, \dots, M_m)$ of $m(\geq 1)$ machines, a sequence $J = J_1, \dots, J_n$ of $n(>> m)$ jobs are given, where each J_j is characterized by its processing time p_j , weight w_j and release time (arrival time) r_j , where $1 \leq j \leq n$.
- **Objective.** Minimization of $\sum_{j=1}^n w_j C_j$.

Definition 1.3. Online Machine Scheduling for Minimizing the Weighted Makespan

- **Input.** A list $M = (M_1, M_2, \dots, M_m)$ of $m(\geq 1)$ machines, a sequence $J = J_1, \dots, J_n$ of $n(>> m)$ jobs are given, where each J_j is characterized by its processing time p_j , weight w_j and release time (arrival time) r_j , where $1 \leq j \leq n$.
- **Objective.** Minimization of $WC_{\max}$

The table 1.1 summarizes several various online and semi-online algorithms for scheduling problems and presents the competitive ratios of the best-known online algorithms for each of these problems.

Table 1.1: Summary of several various deterministic online and semi-online algorithms for machine scheduling problems and the lower bound on the competitive ratios of these problems.

Problem	Algorithm	Competitive Ratio	Lower Bound	References
$1 \mid r_j, \text{online} \mid \sum C_j$	DSPT	2	2	[135, 95]
$1 \mid r_j, \text{online, pmtn} \mid \sum C_j$	SRPT	1	1	[117]
$1 \mid r_j, \text{online, restart} \mid \sum C_j$	RSPT	1.5	1.2108	[154]
$1 \mid r_j, \text{online, restart} \mid \sum C_j$	IRSPT	1.4568	1.2108	Chapter 3
$1 \mid r_j, \text{online, } p_j = a_j t \mid \sum C_j$	D-SGR	$1 + a_{\max}$	$1 + a_{\max}$	[123]
$P \mid r_j, \text{online, pmtn} \mid \sum C_j$	SRPT	5/4	1.047	[124]
$1 \mid r_j, \text{online, pmtn} \mid \sum w_j C_j$	WRSPT	1.2259	1.1038	[102]
$1 \mid r_j, \text{online} \mid \sum w_j C_j$	D-SWPT	2	2	[19]
$P \mid r_j, \text{online} \mid \sum w_j C_j$	Online(ε)	1.791	1.309	[143, 124]
$1 \mid r_j, \text{online, } \frac{p_{\max}}{p_{\min}} \leq \gamma \mid \sum w_j C_j$	β D-SWPT	$1 + 1/\beta$	$1 + 1/\beta$	[149]
$1 \mid r_j, \text{online, } p_j = p, \text{restart} \mid \sum w_j C_j$	LW-3R	1.3247	1.09725	Chapter 4
$1 \mid r_j, \text{online, pmtn} \mid WC_{\max}$	LW rule	1	1	[44]
$1 \mid r_j, \text{online} \mid WC_{\max}$	DLW	2	2	[44, 119]
$P \mid r_j, \text{online, } p_j = p \mid WC_{\max}$	DLW	1.618	1.618	[119]
$1 \mid r_j, \text{online, 1-restart, } p_j = p \mid WC_{\max}$	$\mathcal{A}$	1.4656	1.4656	[121]
$1 \mid r_j, \text{online, } p_j = p, \text{restart} \mid WC_{\max}$	LLW	1.3098	1.2344	Chapter 5

1.3.1 Online Single Machine Scheduling

In the study of online scheduling problems, a key focus is on the single machine scenario where the objective is to efficiently schedule jobs on a single machine. Single machine scheduling serves as a fundamental problem for understanding more complex scheduling environments. The simplicity of this model allows to develop algorithms and theoretical insights that can later be extended to parallel machines and other environments. Furthermore, online single machine scheduling represents real-world situations in industries where a single resource (such as a machine, worker or server) handles tasks that arrive online and is crucial because many practical applications, including manufacturing processes and cloud computing tasks, often rely on a single processing unit.

In the online scheduling, some models allow *restarts* where a job that has already started may be interrupted and later restarted from the beginning. In the following, we distinguish between settings that allow such restarts and those that do not, beginning with the case without restarts.

Online Single Machine Scheduling without Restarts

In the online single machine scheduling without restarts, each job must be processed in the order it is scheduled without any interruption, once it begins. Once a job is accepted and starts processing, it must run to completion without the possibility of preemption. This constraint adds an additional difficulty to the scheduling problem, as the decision to start a job must take into account not only the current jobs in the system but also potential future arrivals.

The absence of restarts in online scheduling models is particularly relevant in real-world systems where jobs (once initiated) cannot be paused or re-allocated to different resources. For example, in manufacturing processes where a machine must complete a task before moving on to the next or in cloud computing scenarios where data processing jobs are costly to stop and restart. In these environments, decisions must be made in real-time, based only on information about jobs that have already arrived, without knowing the future jobs.

In the context of single machine scheduling for minimizing the total completion time ($\sum C_j$) without restarts, various online algorithms have been developed, ranging from simple heuristics like First Come First Served (FCFS) and Shortest Processing Time (SPT) to more sophisticated algorithms that attempt to predict the impact of future arrivals based on probabilistic models.

It is known that simple greedy strategies such as Shortest Processing Time (SPT) are effective in minimizing the total completion time in offline settings but face challenges in the online context. The best-known competitive ratio for deterministic algorithms is 2 [95]. In the randomized setting, a weaker bound is known: any randomized online algorithm for minimizing $\sum C_j$ on a single machine has a performance bound of at least $\frac{e}{e-1} \approx 1.58$ [49][147].

Another challenge in online scheduling without restarts is balancing job priorities. In environments where jobs have varying levels of importance or deadlines, scheduling decisions must consider both the job's urgency and its processing time. In the weighted setting, where each job is assigned a weight, the SPT rule generalizes to scheduling jobs in non-decreasing order of the ratio $\frac{p_j}{w_j}$ where p_j is the processing time and w_j is the weight of job J_j . This yields an optimal schedule for minimizing the total weighted completion time in the offline setting. An optimal algorithm for the deterministic online total weighted completion time problem on a single machine without preemption was presented by Anderson and Potts [18]. Their delayed-WSPT algorithm guarantees a competitive ratio of 2. Moreover Hoogeveen and Vestjens [96] proved that no deterministic algorithm can achieve a better bound even when all jobs have unit weight.

In the context of single machine scheduling for minimizing the weighted makespan, Feng and Yuan [137] were the first to introduce the $WC_{\max}$ for the single machine scheduling. Later, Li [119] explored the online version of this problem and established a lower bound of 2 on the competitive ratio for deterministic algorithms. For the case of identical parallel machines, Li also proposed the optimal online algorithm. Chai et al. [45] further refined these results by developing two deterministic online algorithms with a competitive ratio of 2.

Online Single Machine Scheduling with Restarts

The concept of restarts was first introduced by Shmoys et al. [142] in the context of makespan minimization. Online scheduling models with restarts allow for greater flexibility (Naturally, in an offline environment nothing changes if restarts are allowed, since all jobs are known in advance).

In this model, it is possible to interrupt a running job in case a more critical job arrives (a smaller one, or a heavier one), but in this case the job that was running must be restarted from the beginning: the work done on it is lost. This can lead to significant improvements but it also introduces new challenges, such as determining when a job should be restarted and how much computational effort is lost by interrupting a partially completed job (Figure 1.1).

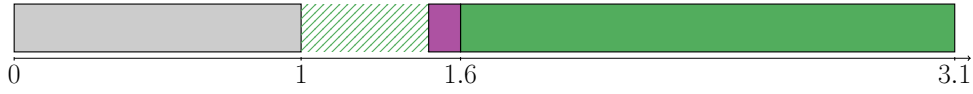


Figure 1.1: Example of preemptive with restarts scheduling model.

This model, also referred to as preemption with restarts, contrasts with the more widely examined preemption with resume model, in which interrupted jobs can continue from the point of interruption (Figure 1.2).

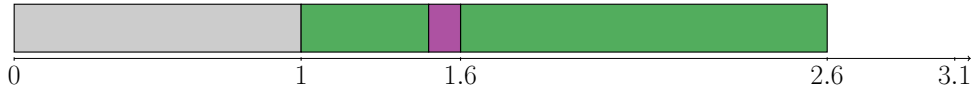


Figure 1.2: Example of preemptive with resume scheduling model.

In preemption with resume model for minimizing the total completion time, an optimal preemptive schedule can be built online by applying the SRPT algorithm. At all times, this algorithm processes the job with the smallest remaining processing time. In the weighted setting, when release times are introduced but preemption is allowed, the classic scheduling rules can be extended: at every moment, the scheduler should execute the job with the Weighted Shortest Remaining Processing Time (WSRPT). In the online setting, the best known competitive ratio of WSRPT is 1.2259 and the best known lower bound is 1.1038 by Jamalabadi and Schwiegelshohn [103]. Furthermore, a range of linear programming (LP)-based approaches have been developed. These include both preemptive and non-preemptive time-indexed LP relaxations, as well as completion-time relaxations [81, 90, 138, 30, 65].

The motivation for the preemption with restarts model is clear, because in many real-world scenarios, it is impractical to interrupt ongoing tasks without incurring loss of progress. As a simple practical example, bread cannot be taken out of the oven partway through baking and then returned later without compromising its quality. Interestingly, when restarts are not allowed, research has shown that there is no difference in competitive ratios between weighted and unweighted models for minimizing the total completion time; both scenarios achieve an optimal competitive ratio of 2, as demonstrated by Hoogetveen and Vestjens [95] and Anderson and Potts [19]. It is hypothesized that the competitive ratio remains at $3/2$ when

restarts are permitted. However, the existing $3/2$ -competitive algorithm for minimizing the total completion time with restarts does not easily extend to weight considerations, emphasizing the need for a deeper understanding of the unweighted problem before tackling the weighted variant.

1.3.2 Related Results

A foundational result by Lenstra, Rinnooy Kan, and Brucker [118] showed that the problem of minimizing the total completion time on a single machine with release dates is strongly NP-hard. However, the problem becomes polynomial-time solvable when all release dates are zero, using the Shortest Processing Time (SPT) rule [145]. Dessouky and Deogun [61] proposed a branch-and-bound algorithm. Deogun [59] presented a partitioning scheme. Chand, Traub, and Uzsoy [46] used a decomposition approach to improve branch-and-bound algorithms. Gazmuri [80] gave a probabilistic analysis. Posner [136] studied a greedy method and proved that it yields an optimal solution under certain conditions. Chu [53] defined a few algorithms which use a local optimal condition to make scheduling decisions.

All of the algorithms in the literature mentioned above are offline in the sense that the algorithms know all the information about jobs even before they arrive in the system.

Shmoys et al. [142] showed that restarts help to minimize the makespan of jobs with unknown sizes on m related machines (machines with speeds). Their algorithm obtains a competitive ratio of $O(\log m)$, while the lower bound without restarts is $\Omega(\sqrt{m})$. Van den Akker et al. [153] considered the objective of minimizing the maximum delivery time on a single machine and introduced a $3/2$ -competitive algorithm, which is optimal. In this problem, after its processing on the machine, each job need to be delivered, which takes a certain delivery time (which depends on the job and is specified in advance).

Further, Hoogeveen et al. [95] showed that restarts help to minimize the number of early jobs (jobs that complete on or before their due date) on a single machine, obtaining an (optimal) competitive ratio of 2 while without restarts, it is not possible to be competitive at all (not even with

preemptions). Chrobak et al. [52] contributed to this area by studying packet scheduling. They consider jobs with release times and deadlines, where the objective is to maximize the throughput, which is the number of completed jobs. They gave a deterministic $3/2$ -competitive algorithm in the model that allows restarts, and showed that in this model the ratio $3/2$ is optimal. The weighted version of this problem was first considered by Fung et al. [75], who gave a 4.56-competitive algorithm. In fact they considered the more general problem of broadcast scheduling. In this problem, there are multiple clients that request pages of possibly different lengths. A single server broadcasts the pages, and broadcasting a page satisfies all existing requests for it. The server is allowed to abort the current page it is broadcasting before its completion and start a new one. To satisfy an aborted request, the requested page has to be broadcasted again from the beginning. They later improved this result using randomized algorithms, achieving a competitive ratio of 3.

Finally, the model of online algorithms with advice, introduced by Bockenhauer et al. [35], adds another dimension to scheduling under uncertainty. In this model, an online algorithm has access to an advice tapes containing bits with information about the input. For instance, the advice can reveal how many jobs the input will contain (using self-delimiting encoding). While most studies in this model focus on online-list inputs, its applicability to online-time inputs remains an open and interesting question. It might appear that if an algorithm has the ability to restart jobs, then it is no longer as important to have advice (put another way, advice cannot help online algorithms very much in this case) as an algorithm can always restart a job if a more important job comes in. However, in fact advice could be very useful here, as it is in many cases by no means clear whether an algorithm should restart its job.

1.4 Structure and Publications

The main part of this thesis is divided into five chapters.

Chapter 2 This chapter provides a survey of recent advancements in the field of online algorithms. While several of the results discussed are closely related to the topics examined in subsequent chapters, we also include

notable developments that although not directly connected, have attracted considerable attention within the field. The content of this chapter is based on the SIGACT News column from 2022 to 2025 [10, 14, 13, 12], which presents a selection of influential papers from major conferences on various aspects of online algorithms. These surveys were prepared in collaboration with Rob van Stee, Sören Schmidt and Paul Hesse.

Chapter 3 This chapter is devoted to the online single machine scheduling problem with restarts, to minimizing the total completion time. Chapter 3 is based on the proposed algorithm in [15] and we present a detailed competitive analysis of our algorithm, proving a 1.4568-competitive ratio for minimizing the total completion time. Additionally, we established lower bounds on the total completion time in different settings. The paper was prepared in collaboration with Rob van Stee and is published in the proceeding of the 15th Workshop on Approximation and Online Algorithms (WAOA 2024).

Chapter 4 This chapter is devoted to the online single machine scheduling problem with restarts, to minimizing the total weighted completion time. Chapter 4 is based on the proposed algorithm in [16] and we present a detailed competitive analysis of our algorithm, proving a 1.3247-competitive ratio for minimizing the total weighted completion time for the case where all jobs have the same processing time. The paper was prepared in collaboration with Rob van Stee and is currently submitted to a conference for presentation.

Chapter 5 This chapter is devoted to the online single machine scheduling problem with restarts, to minimizing the weighted makespan. Chapter 5 is based on the proposed algorithm in [11] and we present a detailed competitive analysis of our algorithm, proving a 1.3098-competitive ratio for minimizing the weighted makespan for the case where all jobs have the same processing time. We established lower bounds demonstrating the limits of this scheduling objective. This paper is currently submitted to a conference for presentation. We also considered the offline version of minimizing the weighted makespan and this paper is currently in preparation for submission to a peer-reviewed journal. These papers were prepared in collaboration with Klaus Jansen, Lis Piroton, Rob van Stee and Corinna Wambsganz.

Chapter 6 Finally, in chapter 6, we explore the broader implications of our findings, discuss the limitations of the proposed algorithms, and outline potential directions for future research.

The following publications are the foundation of this thesis: (All of these papers have been created working together with my advisor Rob van Stee)

- [12] Aflatoun Amouzandeh, Sören Schmitt, Rob van Stee: *SIGACT News Online Algorithms Column 40: 2022 in review*. SIGACT News 54(1): 90-104 (2023)
- [13] Aflatoun Amouzandeh, Sören Schmitt, Rob van Stee: *SIGACT News Online Algorithms Column 41: 2023 in review*. SIGACT News 54(4): 108-125 (2024)
- [15] Aflatoun Amouzandeh, Rob van Stee: *Improved Online Scheduling with Restarts on a Single Machine. Proceedings of the 15th Workshop on Approximation and Online Algorithms (WAOA 2024)*, pp. 16–30 (2024) and Under review in *Theory of Computing Systems*.
- [14] Aflatoun Amouzandeh, Sören Schmitt, Rob van Stee: *SIGACT News Online Algorithms Column 43: 2024 in review*. SIGACT News 55(4): 94-107 (2024)
- [10] Aflatoun Amouzandeh, Paul Hesse, Sören Schmitt, Rob van Stee: *SIGACT News Online Algorithms Column 44: 2025 in review*. To appear in *SIGACT News*.
- [11] Aflatoun Amouzandeh, Klaus Jansen, Lis Pirotton, Rob van Stee and Corinna Wambsganz : *Minimizing the Weighted Makespan with Restarts on a Single Machine* arXiv preprint arXiv:2510.09589 (2025)
- Aflatoun Amouzandeh, Klaus Jansen, Lis Pirotton, Rob van Stee and Corinna Wambsganz: *Minimizing the Weighted Makespan*. In preparation for submission to a journal.
- [16] Aflatoun Amouzandeh, Rob van Stee: *Minimizing the Total Weighted Completion time with Restarts on a Single Machine*. Under review in a conference.

CHAPTER 2

Survey of Online Algorithms

Online optimization considers settings where input is revealed over time and decisions must be made without knowledge of future events. These challenges make the design of effective algorithms both demanding and interesting. As a result, the field is broad and includes a wide range of problems, many of which are actively studied and continue to develop rapidly.

In this chapter, we survey selected results in this field that were published from the year 2022 to 2025. During this time, a number of impactful papers appeared in premier conferences, advancing our understanding of algorithmic techniques under online constraints. We highlight a subset of these results and focus on contributions that are closely related to our work.

While our emphasis lies in scheduling theory, it is important to acknowledge the diverse and rapidly evolving nature of online optimization as a whole. The papers discussed here, along with similar presentations on different topics, can be found in the annual SIGACT News column on online algorithms.

2.1 Online Scheduling

We begin by some of the most recent results in the area. In particular, at SODA 2025, Qingyun Chen, Sungjin Im and Aditya Petety present a scalable algorithm for minimizing the total flow time in several settings. A scalable algorithm is an algorithm that is $O(1)$ -competitive with arbitrarily small resource augmentation on the speed and is in many setting the

best possible result. The results of this paper are inspired by viewing the Shortest Remaining Processing Time algorithm (SRPT) as a form of gradient descent, meaning that we process the jobs that decrease the residual optimum the most. It is shown that this works in settings in which the offline residual optimum problem is supermodular, by generalizing the potential function introduced a few years ago [43]. Finally, it is shown that if maximizing a linear function subject to the constraints is a substitute function, then the residual optimum is supermodular. This is done using a connection to Walrasian markets. Scalable algorithms are presented in this way for matroid scheduling, generalized flow scheduling, speed-up curves on partitioned resources and minimizing the weighted flow time on unrelated machines. This last result reproduces a result from [17], the other results are new. The authors show furthermore that they also recover other results from the literature in an easier way.

In ESA 2025, Ekin Ergen [68] considers an extension of online makespan scheduling on identical parallel machines by introducing scenarios. In this problem, jobs are revealed one by one with their processing times, and each job must be assigned irrevocably to one of m identical machines as soon as it is revealed. Each scenario is defined as a subset of jobs. The numbers of machines m and the number of scenarios K are known in advance. The objective is to minimize the maximum makespan of the worst-case scenario.

Ekin Ergen proves the existence of a $5/3$ -competitive algorithm for the case of two machines and two scenarios ($m = K = 2$). For larger numbers of machines or scenarios, tight and near-tight bounds are established. These lower bounds are proved for the special case of unit processing times using constructions based on hypergraph coloring. Furthermore, an optimal 2-competitive algorithm is proposed for unit processing times, $K = 3$, and varying m .

In ITCS 2023, Varun Gupta et al. [88] explore two fully-dynamic multi-dimensional vector load balancing problems with recourse. The problems involve scheduling and bin packing for a stream of n job insertions and deletions, where each job is represented as a vector in $\mathbb{R}_{\geq 0}^d$. The algorithm can reassign jobs from one machine or bin to another, and the total number of times any job is reassigned is called its recourse. The authors present

two algorithms for the fully-dynamic online vector scheduling problem with recourse. In this problem, the algorithm must maintain an assignment of the active jobs to m identical machines, to minimize the makespan (maximum load on any dimension on any machine). Their first algorithm is a randomized algorithm with a competitive ratio of $O(\frac{\log d}{\log \log d})$ and an $O(1)$ amortized recourse against oblivious adversaries. The second algorithm is a deterministic $O(\log d)$ -competitive algorithm with an instance-dependent recourse bound against adaptive adversaries. The authors also consider the fully-dynamic online vector bin packing problem, where the size of all jobs in all coordinates is $O(\frac{1}{\log d})$. In the vector bin packing problem, the algorithm must maintain an assignment of active jobs into a number of bins of unit capacity in all dimensions, to minimize the number of bins currently used. For this problem, they present a randomized $O(1)$ -competitive algorithm with $O(\log n)$ recourse against oblivious adversaries.

Sungjin Im et al. [100] consider online load balancing on unrelated machines. In this setting, the size of a job depends on the machine: each incoming job is given as a vector of m components, specifying the size of this job on each machine. For the strict online model, it is known that the competitive ratio is $\Theta(\log m)$. The current paper considers the random order model and gives a lower bound of $\Omega(\sqrt{\log m})$ for the special case of graph balancing. In this model, edges representing jobs arrive online, and we must immediately orient each arriving edge towards one of its endpoints. The objective is to minimize the maximum indegree of any vertex. This corresponds to scheduling on unrelated machines where each job has unit size on exactly two machines and infinite size on the rest. The greedy algorithm which assigns each job to the less loaded machine is shown to have a competitive ratio of $\Omega(\log m / \log \log m)$ for graph balancing. For the general case of scheduling on unrelated machines, an algorithm with competitive ratio $O(\log m / \log \log m)$ is presented.

2.2 Online Packing

One of the most studied problems in the field of competitive analysis are packing problems. Items of some size and shape arrive (online) and have to be irrevocably packed into containers of some other size and shape.

The goal is to minimize the used space. Many packing problems allow for $O(1)$ -approximations offline and $O(1)$ -competitive algorithms online. This holds if the arriving items are polygons which may be rotated and either packed into the collection of 1×1 bins (bin packing) or the infinite strip of width 1 (strip packing) even in the online version. If the polygons may not be rotated there still exist $O(1)$ -approximations for the offline version. The conjecture that there also exist $O(1)$ -competitive algorithms for the online version arose naturally. However, in SODA 2023 Anders Aamand et al. disprove this conjecture for several online packing problems, including bin packing and strip packing [1]. They show a lower bound of $\Omega\left(\sqrt{\log n / \log \log n}\right)$ for bin packing and strip packing, where n is the number of polygons in the input. To do so they connect strip packing to the seemingly unrelated problem of online sorting. In online sorting we are given an array A of size $\lfloor \gamma n \rfloor$ and n reals from the interval $[0, 1]$ arrive online. The reals should be placed into the array in such a way that the total sum of the absolute differences of consecutive numbers is minimized. The optimum is to sort the numbers in the array. The two problems are connected in the sense that if there exists a C -competitive algorithm for strip packing, then there exists a $4C$ -competitive algorithm for online sorting with $\gamma = 2C$. Hence a lower bound on online sorting implies a lower bound on strip packing. They complement their negative results with an $O\left(n^{\log 3 - 1} \log n\right)$ -competitive algorithm for online strip packing, which recursively subdivides the strip into parallelograms and embeds the polygons into parallelograms.

Aniket Murhekar et al. [133] explore a variant of the dynamic bin packing problem (DBP) motivated by cloud computing applications called MinUsageTime DBP. In this problem, items (jobs) appear online and must be packed into bins (servers) without exceeding their resource capacity while minimizing server usage time. They further extend this problem to MinUsageTime Dynamic Vector Bin Packing (DVBP), where jobs and servers have multidimensional resource demands.

In SPAA 2023, the authors briefly announce their results about the competitive ratios of Any Fit packing algorithms, specifically First Fit, Next Fit, Best Fit and Move To Front, for solving DVBP. They demonstrate that the competitive ratio of Move To Front for this problem is at most

$(2\mu + 1)d + 1$, where μ represents the ratio of the maximum to minimum item durations. This result implies a significant improvement over previous bounds, particularly when $d = 1$. Their study also establishes lower bounds for Any Fit packing algorithms, demonstrating almost tight lower bounds in specific cases for First Fit, Next Fit and Move To Front. Additionally, their research includes an experimental study of the average-case performance of these algorithms using randomly generated synthetic data, that shows the Move To Front algorithm outperforms other Any Fit packing algorithms.

Packing Integer Programs (PIPs) are an important class of integer programs. In the online setting, we are given a column vectors b, c with nonnegative entries and the columns of an unknown constraint matrix $A \in [0, 1]^{d \times n}$ are revealed sequentially and each having an associated value. We seek a vector $x \in \{0, 1\}^n$, which maximizes the inner product $c^T x$, subject to $Ax \leq b$.

In SODA 2022, robust algorithms for packing integer programs in the Byzantine secretary and Prophet-with-Augmentations models were designed by C.J. Argue et al. [20]. The general idea of these algorithms is based on the robust threshold estimation using tools from online learning. They have also improved known results [38] for the case of packing a single item or an independent set in a matroid.

The Byzantine secretary model is one attempt to model adversarial corruptions in the stochastic models [110, 38]. In this model each column of A is either red or green, where these colors are not known to the algorithm. Red columns $\mathcal{R}$ choose their arrival times adversarially and green columns $\mathcal{G}$ arrive independently and uniformly at random. The benchmark is now the value of the optimal green set. Let $g_{\max}$ and $OPT(\mathcal{G} \setminus g_{\max})$ denote the green column with the largest value and the optimal value of the offline PIP when restricted to the green columns without $g_{\max}$, respectively. They have shown existence of an algorithm for PIPs with value $\Omega(\mathcal{G} \setminus g_{\max})$ in Byzantine secretary model when the entries of b are $\Omega(\text{poly log } n)$. They have also obtained more nuanced results when the algorithms are given a very rough estimate of the $OPT(\mathcal{G})$.

Similarly, the authors use the idea of these algorithms to obtain robust algorithms for PIPs in the prophet model as well. In this model the

columns of matrix A are known up-front, but the value of each column is drawn independently from a distribution. Specifically, they have considered the Prophet-with-Augmentations framework [101] where an adversary is allowed to perturb the values by adding nonnegative values. In this case they have shown algorithms robust to such perturbations.

Packing and covering linear programs (LPs) are the problems of the form:

$$\text{Covering LP}(\mathbb{C}) : \min_{x \in \mathbb{R}_{\geq 0}^n} \{c^\top x \mid Ax \geq b\}, \text{Packing LP}(\mathbb{P}) : \max_{y \in \mathbb{R}_{\geq 0}^m} \{b^\top y \mid A^\top y \leq c\},$$

where $A \in \mathbb{R}_{\geq 0}^{m \times n}$, $b \in \mathbb{R}_{\geq 0}^m$ and $c \in \mathbb{R}_{\geq 0}^n$. The two programs are dual to each other, so we denote by $OPT \geq 0$ their shared optimum. We say x is a ε -approximation for $\mathbb{C}$ if $Ax \geq b$ and $c^\top x \leq (1 + \varepsilon)OPT$. Similarly, we say y is a ε -approximation for $\mathbb{P}$ if $A^\top y \leq c$ and $b^\top y \geq OPT/(1 + \varepsilon)$.

In SODA 2023, Sayan Bhattacharya et al. [33] study the Packing-covering LPs. While the central focus of their paper lies in dynamic setting for packing-covering LPs, the authors also consider the online version of this problem. In the online covering LP problem $\mathbb{C}$ with m constraints and n variables, the constraints of $\mathbb{C}$ arrive online one after another. They improved the existing $O(\log n)$ -competitive lower bound for this problem by allowing some amount of recourse. For the online covering LP problem recourse is defined to be the number of times the value of any variable is decreased. They also consider online packing LP where the constraints of the LP arrive one after another. Consequently, they obtain $(1 + \varepsilon)$ -competitive online algorithms for both covering and packing LPs with polylogarithmic total recourse. Furthermore, they show the first streaming algorithm for general packing covering LPs with linear space and polylogarithmic passes.

2.3 Online Coloring

In ICALP 2022, Yossi Azar et al. [22] consider an online coloring problem motivated by machine scheduling with conflicts. Given a set of n vertices with weights and a k -coloring of the vertices the edges arrive online. An algorithm has to output a proper k -coloring at every step and has to pay the weight of a vertex if it gets recolored. They call this problem *Competitive Vertex Recoloring* or *(Semi-Dynamic) Online Disengagement*. The goal is to minimize the overall cost. Vertices represent jobs, colors represent machines and edges represent a conflict between two jobs. The authors first focus on the bipartite graphs.

By combining a greedy approach that recolors the lighter component and a conservative approach that recolors components to stay close (in the sense of Hamming distance) to the initial coloring they present an $O(\log n)$ -competitive deterministic algorithm. They complement this result by showing a matching lower bound of $\Omega(\log n)$ even for randomized algorithms. The second part of the paper is about general graphs with $(\Delta + 1)$ -colorings, where Δ is an upper bound on the number of neighbors of the nodes. In this setting the authors present an $\Theta(\Delta)$ -competitive deterministic algorithm and an $\Theta(\log \Delta)$ -competitive randomized algorithm. The algorithms are based on the same idea and only differ in the method of selecting the new colors. If edge deletion is also allowed the problem is called *Full-Dynamic Online Disengagement*. For this case any deterministic algorithm is $\Omega(n)$ -competitive and any randomized algorithm is $\Omega\left(\frac{\log n}{\log \log n}\right)$ -competitive.

In the edge coloring problem under online edge arrivals, the input is the vertex set V and the maximum degree Δ of an n -node graph $G = (V, E)$. The edges of G are revealed over time, and the online algorithm has to irrevocably assign a color to each newly arriving edge such that any two incident edges have different colors. The objective is to minimize the number of distinct colors used. For this problem, back in 1992 Bar-Noy et al. [28] showed that the greedy algorithm which assigns every arriving edge any valid color uses at most $2\Delta - 1$ colors, and for graphs with degree $\Delta = O(\log n)$ no algorithm can maintain a proper coloring using fewer than $2\Delta - 1$ colors. This bound just means that any edge coloring algorithm needs $\omega(\log n)$ additional colors to achieve a proper coloring. They conjectured that the online algorithm which uses $\Delta + O(\sqrt{\Delta} \log n)$ colors and samples a color for each edge uniformly at random from the set of valid colors succeeds with constant probability.

In STOC 2022, Janardhan Kulkarni et al. [112] make the first progress towards proving this longstanding conjecture. They consider the recent result by Gamlath et al. [76] that shows a lower bound of 2 for the online matching problem in the edge arrival setting (a valid coloring for a graph can be viewed as a set of matchings on that graph), whereas better algorithms exist for the vertex arrival setting. It seemed possible that the edge coloring problem might have the same property. Instead, this paper gives an $\left(\frac{e}{e-1} + o(1)\right) \Delta$ -edge coloring with high probability in the oblivious adversary setting. The algorithm is based on reducing this problem to a matching problem on locally treelike graphs, and then applying a tree recurrence based approach for arguing correlation decay. This is (apparently) a well known and widely used technique in the statistical

physics and sampling literature [155, 144, 26]. The idea is that as long as the underlying graph is a tree (so far), the probabilities that either endpoint of a new edge are already matched are independent, because those endpoints are in separate connected components. If the graph is sufficiently treelike, the correlation between these probabilities is still very low and the argument also goes through.

In graph coloring, each vertex in a given graph needs to be assigned a color (a positive integer) such that no two adjacent vertex have the same color. This problem, including the dynamic version where vertices are inserted and deleted over time, is a fundamental problem in computer science. Many pessimistic lower bounds for online graph coloring have been revealed. To go beyond these bounds, several settings were proposed. One of them is the limited recourse framework. Here the algorithm is allowed to change a limited number of past decisions. Even in this setting, maintaining a reasonable coloring requires a large amount of recourse even for forests.

As a result, the focus has shifted to coloring interval and unit interval graphs. Bosek et al. [36] showed that a recourse budget of $O(k^2)$ per update is sufficient for maintaining a $(k + 1)$ -coloring of a k -colorable graph. If an actual k -coloring is required, a budget of $O(k!\sqrt{n})$ is sufficient if only vertex insertions occur, where n is the length of the input sequence (number of insertions). A lower bound of $\Omega(\log n)$ was provided as well. In ESA 2022, Bartłomiej Bosek and Anna Zych-Pawlewicz [37] improved both of these bounds by an exponential factor, to $O(k^7 \log n)$ and $\Omega(n)$ respectively.

Rajmohan Rajaraman and Omar Wasim in their paper [141] presented at ESA 2024 consider the *online recoloring problem* introduced recently by Azar et al. [22]. Their paper addresses algorithmic challenges in scheduling virtual machines (VMs) or processes in distributed systems and cloud services. In the online recoloring problem, there is a fixed set V of n vertices and an initial coloring $c_0 : V \rightarrow [k]$ for some $k \in \mathbb{Z}^{>0}$. As edges representing requests arrive over time, a proper coloring must be maintained for the graph G_t induced by requests until time t . This ensure that adjacent vertices receive different colors. In the distributed systems, a vertex corresponds to a VM, an edge corresponds to the requirement that the two endpoint VMs be on different clusters, and a coloring is an allocation of VMs to clusters. The objective is to minimize the total weight of recolored vertices over the sequence of online requests.

The authors propose competitive algorithms for capacitated online recoloring

and fully dynamic recoloring, in which there is a bound on the number or weight of vertices in each color. They mainly introduce an $O(\log n)$ -competitive deterministic algorithm for weighted bipartite graphs using $(1 + \varepsilon)$ -resource augmented, where $\varepsilon \in (0, 1)$ is an arbitrary small constant. They also present an $O(n \log n)$ -competitive algorithm for the fully dynamic recoloring in unweighted setting with $(1 + \varepsilon)$ -augmentation, where $\varepsilon \in [8/n, 1)$. Additionally, they introduce a study on Δ -recoloring in an overprovisioned setting. In this setting, there are n vertices, and Δ colors. Each color is assigned exactly n/Δ vertices initially. The graph induced by the input sequence has a maximum degree of $(1 - \varepsilon)\Delta$ and each color $i \in [\Delta]$ has a capacity of $(1 + \varepsilon)n/\Delta$, for an arbitrarily small constant $\varepsilon > 0$. Their main result in this setting is an $O(1)$ -competitive randomized algorithm for cases where $\Delta = O(\sqrt{n/\log n})$.

2.4 Online Matching

Susanne Albers and Sebastian Schubert study the b -matching problem in bipartite graphs $G = (S, R, E)$ in their ESA paper [8]. The offline set S consists of servers and vertices (request) of the right hand side R arrive online. A server s can answer (or can be matched) to b_s requests. The objective is to maximize the overall number of matched requests. The authors consider (k, d) -graphs, where k is a lower bound on the degree of each server and d is an upper bound on the degree of each request. For this setting they improve the previously best known bound of $1 - 1/e^{k/d}$ significantly by presenting an asymptotically optimal algorithm (for $k \geq d \geq 2$) if b tends to infinity.

In STOC 2022, Zhiyi Huang et al. [98] considered the problem of online stochastic matching. In this problem, there are two sets of vertices, one offline and one online. The offline vertices are known in advance and the online vertices arrive over time. In a strict worst-case setting, the ratio $(e - 1)/e$ is optimal for this problem. We can also consider the case where the online vertices are drawn according to some distribution. In previous papers, the idea of the power of two choices was used. The algorithm finds two candidate offline neighbors based on the distribution of online vertices and matches the online vertex to the first unmatched candidate if there is any. Here the candidates are determined independent of which offline vertices have been matched so far. The authors of the current paper improve on this in two ways: by using more than two choices and by focusing on unmatched neighbors. They analyze this by using online correlated selection and by considering the overall progress of the online

algorithm rather than characterizing the matching probabilities of individual offline vertices.

Niv Buchbinder, Seffi Naor and David Wajc [41] consider online bipartite matching problems in SODA 2023. Currently, for many such problems, the best known fractional algorithms outperform their randomized integral counterparts, even though lossless rounding is possible. Nikhil Devanur et al. [62] explained this by showing that online lossless rounding is impossible.

In the current paper, the fact that each randomized online algorithm induces a fractional algorithm with the same competitive ratio (by setting the fractional values of the decision variables to their expected value under the randomized algorithm) is used for the first time to generate upper bounds. The authors add constraints to reduce the online integrality gap. The added constraints allow for lossless online rounding in this specific setting, and give new online matching algorithms, a resolution to an open problem concerning randomness and advice complexity of online matching and a systematic method of designing algorithms for online correlated selection.

In ESA 2023, Ilan Reuven Cohen and Binghui Peng [55] explore various generalizations of the online bipartite matching problem in the context of (k, d) -bounded degree graphs. These graphs, initially proposed by Naor and Wajc [134], model natural properties on the degrees of advertisers and queries in the allocation and ad-auctions problems. While previous studies concentrated on situations where $k \geq d$, the authors in [55] consider the scenario where $k \leq d$. In this scenario, advertisers express interest in a large number of ad slots and each ad slot attracts only a relatively small number of advertisers.

Their research introduces an algorithm based on fractional techniques and establishes a competitive ratio $\tau(k, d) = 1 - \frac{d-k}{d}(1 - \frac{1}{d})^{d-k}$ for the b-matching [106] and allocation problems under the small-bid assumption. The algorithm's performance is analyzed using primal-dual techniques. They extend their analysis to the online ad-auctions problem (the Adwords problem) under equally valued bids case. Additionally, they demonstrate that achieving $\tau(k, d)$ -competitiveness is impossible for the AdWords problem in arbitrary valued case, even in (k, d) -bounded degree graphs.

Also in ESA, Jingxun Liang et al. [120] consider perturbed versions of the standard Ranking and Balance algorithms for online matching. The randomized algorithm Ranking [108] has a competitive ratio of $1 - 1/e$, and the deterministic

algorithm Balance [106] achieves the same competitive ratio for the fractional version of the problem. Perturbed-Balance is $(1 - 1/e)$ -competitive for the weighted version of the problem, and Perturbed-Ranking is $(1 - 1/e)$ -competitive for the special case in which all edges incident to an offline vertex u have the same weight which is equal to the budget of that vertex. This problem is known as vertex-weighted online bipartite matching.

Both these algorithms select the offline vertex u which maximizes $f(x_u)w_{uv}$ on arrival of the online vertex u , where $f(x) = 1 - e^{x-1}$; only the definition of x_u is different. For Perturbed-Ranking, x_u corresponds to the random rank of u , whereas for Perturbed-Balance, it corresponds to the fraction of budget spent so far. Liang et al. show that this choice of f is the only one that works, in that all other choices (apart from scaling) give inferior results. Interestingly, they also show that for the version of the problem in which the budgets of the offline vertices are unknown, Perturbed-Ranking does not achieve the competitive ratio of $1 - 1/e$ for any perturbation function. As this is the only known algorithm for this problem, the authors conjecture that there is no $(1 - 1/e)$ -competitive algorithm for this problem.

In ICALP 2023, Michal Feldman et al. [72] study truthful mechanisms for welfare maximization in online bipartite matching. Unlike most online matching settings where agents arrive online, they consider a reversed online matching problem that items arrive online while buyers are present throughout the process and can strategize over future rounds. In this setting, every buyer is associated with a (possibly private) desired set of items and has a private value for receiving at least one of them. Truthful mechanisms in addition to finding a good matching, also ensure that it is in the agents' best interest to report their true values. These mechanisms can also charge some payment from the agents in order to encourage truthful reporting of values. The authors explore different scenarios of this problem, including myopic and non-myopic agents, prompt and tardy payment mechanisms, and whether agents have private or public information about their desired items.

Specifically, for the agents who do not strategize over the future (i.e., myopic agents), they present a truthful deterministic prompt mechanism that guarantees achieving at least a half of the optimal welfare and a randomized prompt mechanism that is $e/(e - 1)$ -competitive. In the case of non-myopic agents who can strategize about their values but not about their desired items, similar competitive ratios are achieved when the mechanism is allowed to delay payment decisions until the end of the online phase to set prices (i.e., tardy mechanism).

The authors establish strong impossibility results for prompt mechanism in this scenario. Finally, they consider non-myopic agents when both valuations and desired items are private and they also discuss the concept of ex-post vs. ex-ante truthfulness, considering whether agents should report their true values to maximize their expected utility or their utility in any realization of the mechanism's random choices.

In APPROX 2023, Lindsey Deryckere and Seeun William Umboh [60] introduced the study of the online min-cost perfect matching problem with set delay (MPMD-Set), which generalizes the online min-cost perfect matching problem with delay (MPMD). In this problem, m requests arrive over time in a metric space of n points and the algorithm must choose to either match a request or delay it. The delay costs are determined by the set of pending requests. The goal is to create a perfect matching of all requests while minimizing the sum of distances between matched requests and the total delay costs. The authors consider the non-clairvoyant setting, meaning that the algorithm does not have knowledge of future delay costs.

They demonstrate that achieving a deterministic algorithms with competitive ratio based solely on n or m is impossible for MPMD-Set. Therefore, the authors concentrate on a size-based delay, where the instantaneous delay cost at any point in time is a monotone non-decreasing function of the number of unmatched requests at that time. Their main result includes the development of an $O(2^m)$ -competitive deterministic algorithm and an $O(m^4)$ -competitive randomized algorithm for MPMD-Size (size-based delay) problem in the non-clairvoyant setting. They also establishes lower bounds on competitive ratios of both deterministic and randomized algorithms for MPMD-Set problem. Additionally, the authors provide an $O(m)$ -competitive deterministic algorithm for MPMD with uniform concave delay in the clairvoyant setting.

There are various problems in which it is required to select a subset of elements under some conditions. One example is selecting a subset of edges that form a matching in a graph. A tool for doing this is contention resolution schemes. Calum MacRury et al. [130] consider a setting where each edge e is initially active according to a known probability x_e . The goal of the contention resolution scheme is to select every edge e with probability at least c conditional on it being active, for a constant c which is as large as possible, over both the randomness in the active edges and any randomness in the selection of the algorithm. That is, each edge must be selected with unconditional probability at least cx_e . Since c must always be small if one vertex is incident to edges with very high sum of

the x_e values, it is typically required that the vector of x_e values is feasible in the fractional relaxation, meaning it must lie in the matching polytope: the sum of x_e values of edges incident to any vertex must be at most 1. This condition is satisfied in many settings.

The authors provide improved online and random-order contention resolution schemes for both general graphs and bipartite graphs, as well as new impossibility results. In particular, the online contention resolution scheme provided by Moran Feldman et al. [73] is shown to achieve $c = 0.349$ for bipartite graphs, up from 0.337 (for general graphs) before. For general graphs, the result is improved to $c = 0.344$. It is also shown that a value better than 0.4 cannot be achieved (0.433 on bipartite graphs), and that the scheme considered here can reach at most 0.361 for general graphs and at most 0.382 for bipartite graphs.

In WAOA 2024, Tung-Wei Kuo [114] considers the *online bipartite perfect matching problem with delays*, where m requests and m servers arrive over time and the online algorithm can delay matching between servers and requests at a cost. The goal is to minimize the total distance and delay costs. The author presents a deterministic algorithm with a competitive ratio of $O(\sqrt{m} \log^2 m)$ for the case where servers and requests are on the real line. This algorithm is based on the Robust Matching (RM) algorithm introduced by Sharath Raghvendra [139] for the minimum cost bipartite perfect matching problem, where no delays are allowed and all servers arrive at the beginning.

In SODA 2024, Shuyi Yan [157] proposes an algorithm for the edge-weighted online stochastic matching problem. In this problem, we are given a bipartite graph consisting of offline vertices and a set I of online vertex types, along with a distribution over these types. Each online vertex type $i \in I$ has an arrival rate λ_i , which represents the expected number of online vertices of that type. As each online vertex arrives, it independently samples its type from the given distribution and must be immediately and irrevocably matched to an offline vertex. Each edge in the graph has a weight, and the objective is to maximize the expected total weight of the resulting matching.

Previous work on online stochastic matching in weighted setting includes results on vertex weighted matching [97, 99], edge-weighted matching with free disposal [99] and edge weighted matching under the integral assumption [89, 39]. However, these results applied only to specific models or assumptions, such as requiring free disposal or integral arrival rates. For the general edge-weighted setting, no algorithm had previously surpassed the $1 - \frac{1}{e}$ benchmark.

Yan’s proposed algorithm utilizes an LP-based preprocessing technique and builds on the Poisson arrival model to simplify the analysis of the problem. This algorithm achieves a competitive ratio of 0.645 in the general edge-weighted setting. Specifically, this result improves upon the $1 - \frac{1}{e}$ competitive ratio bound for online stochastic matching [71].

2.5 Allocations and Auctions

Jiang et al. [105] considered several online resource allocation problems in SODA 2022, in particular online stochastic knapsack and k -unit prophet inequalities. There is a single resource that can be arbitrarily subdivided. Queries specifying a reward and a size arrive online. Each query must be immediately served (by allocating the requested capacity and gaining the reward) or rejected. The sizes and rewards are drawn independently from non-identical known distributions. The goal is to maximize the expected total reward.

In the special case of k -unit prophet inequalities, all sizes are equal to $1/k$, where 1 is the total size of the resource (the size of the knapsack). The best known guarantees come from comparing the online algorithm to an LP relaxation of the prophet, which only has to satisfy the capacity constraint in expectation. The prophet is a standard benchmark for online algorithms; it sees the size and reward realizations in advance and can always make an optimal allocation in hindsight.

Alaei [6] showed that an online algorithm can collect at least $1 - 1/\sqrt{k+3}$ of this LP relaxation. The current paper improves this and gives tight bounds relative to the LP relaxation. Of course, this leaves open the possibility that a better result may be achieved for the actual problem in another way. For the online knapsack problem, the new guarantee is $1/(3 + e^{-2}) \approx 0.319$, improving from 0.2. An interesting aspect of this result is that in contrast to many other papers, queries are not split into large or small depending on their size; in fact the paper shows that such an approach cannot give a guarantee better than 0.25.

In online combinatorial auctions one generally has to allocate m items that arrive over time to n agents. In STOC 2023, José Correa and Andrés Cristi [56] consider the opposite setting in which the agents arrive over time. Each agent i is associated with a valuation function v_i which assigns value to each possible subset of the items. The functions are randomly and independently chosen according to a given distribution. The goal is to maximize the expected sum of the agents’ valuations (social welfare) of the allocation.

This problem has been intensively studied for different families of valuation functions, specifically submodular and XOS valuation functions. Correa and Cristi tackle the more general case of subadditive functions, which have the property that for two sets of items the sum of the individual values of the sets is at least as large as the value of the union. For this family they present a 6-competitive algorithm [56], improving the previously best known result of $O(\log \log m)$ [64].

In ESA 2023, Amit Ganz, Pranav Nuti and Roy Schwartz consider the online Submodular Welfare problem [77]. In this problem n bidders each equipped with a general non-negative submodular utility function compete for m items that arrive online. Instead of knowing the agents' utility functions explicitly, the algorithm can query the value of subsets of items that have already arrived. The goal is to assign each item to a bidder while maximizing the sum of utilities. The authors give a simple randomized algorithm that achieves a tight competitive ratio of $1/4$ even if the adversary can determine the items' arrival order. Their algorithm assigns the item to the bidder with the highest marginal value with probability of $1/2$, to the bidder with second highest marginal value with probability of $1/4$ and so on decreasing the probability geometrically. If the adversary is less powerful and the arrival order of the items is uniformly random, the authors improve the previously best known result of $1/4$ to 0.27493 by a better analysis of the Residual Random Greedy algorithm given by Niv Buchbinder et al. [40].

In the classic gambler's problem n candidates are revealed over time, modelled by realizations of i.i.d. non-negative random variable X_i with known distribution F . After every revelation you must either accept or decline a candidate without knowledge of future candidates. The goal is to choose a candidate that maximizes the expected value. Martin Hoefer and Kevin Schewior consider a more realistic setting in ESA 2023 [94]. Instead of revealing the complete realization, each candidate reveals only a part of their realization. The algorithm only learns whether $X_i \geq t_i$ or not, for a threshold t_i chosen by the algorithm. The threshold can be adapted over time. Further, instead of the requirement to make a decision immediately, the algorithm can make its decision after having tested all candidates. The authors give thresholds that result in a $0.869 - o(1)$ -competitive algorithm. The algorithm increases the testing threshold whenever a candidate tested positive for the previous threshold. If candidates can be tested multiple times with n total tests, they can guarantee $1 - o(1)$ -competitiveness.

Tomer Ezra et al. [69] introduce the order-competitive ratio, which is a new

measure for the performance of online algorithms in Bayesian settings. In a prophet inequality setting, there are n boxes. Each box t contains a value v_t drawn from a known probability distribution F_t (which depends on the box). The boxes arrive online and reveal their values. For each box, the online algorithm must decide whether to accept the value v_t or to discard the box. The standard objective is to maximize the expected accepted value. The performance of online algorithms for this problem has typically been measured using the competitive ratio, meaning that the value achieved by the algorithm is compared to that of a prophet which can look into all the boxes in advance. The authors argue that this is too pessimistic and fails to distinguish between some classes of algorithms.

The order-competitive ratio instead compares the value to the best online algorithm that knows the order of the boxes, that is, the order in which the various probability distributions arrive. Interestingly, they show that adaptive algorithms are strictly better for this performance measure. Using the standard competitive ratio, single threshold algorithms which set a threshold value at the start and then pick the first box (if any) which contains at least that value are optimal. Adaptive algorithms instead consider the set of boxes that has arrived so far to make their decisions; intuitively, this should indeed achieve better results. To be precise, the algorithms used by the authors use adaptive thresholds instead of one fixed threshold.

In ICALP 2023, Ilan Reuven Cohen and Debmalya Panigrahi [54] consider online allocation framework with the learning-augmented setting. Online allocation is a class of problems where items arrive online and each item needs to be allocated to agents. The agents have fixed utility or cost associated with each assigned item. The goal in online allocation problems is to maximize or minimize some objective. Their paper focuses on divisible items, which means that items can be partially allocated to agents and also unlike previous work that often focused on specific objectives, their approach applies to a wide range of maximization and minimization problems.

They define two technical conditions for the objective function of online allocations problems. Monotonicity ensures that the function's values consistently increase or stay the same when the costs/utilities of agents in one allocation are greater than or equal to those in another allocation. Homogeneity guarantees that if the costs/utilities for one allocation are uniformly scaled by a positive factor then the function's value scales proportionally by the same factor. A well-behaved function then is one that satisfies both the conditions of monotonicity and homogeneity.

Their main result is to present a framework for designing near-optimal algorithms for any online fractional allocation problem with well-behaved objective, using a single learned parameter for every agent. The authors improve results for learning-augmented makespan minimization (minimizing maximum cost) and they also present the first learning-augmented nearly-optimal algorithms for other objectives such as Santa Claus (maximizing minimum utility), Nash welfare (maximizing geometric mean of utilities), and ℓ_p -minimization. Furthermore, they discuss the resilience of their algorithms to errors in the learned parameters and also examines the learnability of these parameters.

2.6 Advise and Predictions

An area in online algorithms which has received increasing attention is learning-augmented algorithms. This is a further development of the well-known setting of online algorithms with advice, in which an online algorithm has access to an advice tape with information about the input. The advice complexity of a problem is defined as the size of advice (number of bits) that need to be read by the algorithm in order to provide an optimal solution. In the more recently developed learning setting, the online algorithm receives a prediction about the input, but this prediction may not be accurate. This can be viewed as untrusted advice. The goal in this setting is to provide algorithms that give performance close to the best known approximation ratio in case the prediction is correct and close to the best known competitive ratio if it is completely wrong. Hence, the prediction should in general improve the performance of the online algorithm and at worst harm it slightly. Since the publication of [128] by Lykouris and Vassilvitskii in 2018, learning-augmented algorithms have become increasingly important and have been considered for a wide range of online problems.

In SODA 2022, Yossi Azar et al. [23] study online version of classical graph problems. They point out that there are two different kinds of errors in predictions for problems in metric settings: predictions that are completely wrong (do not correspond to any request in the actual input) and predictions that are only inaccurate (some distance away from the actual request). An error measure that focuses only on the second kind of error is very sensitive to individual errors: if only one prediction is completely wrong, the total error can suddenly be very high or even unbounded. On the other hand, focusing on only the first kind of error can give unrealistic results in case all predictions are very close to the actual requests but none of them actually match.

The authors provide a new notion of prediction error for these kind of problems which they call metric error with outliers: it calculates the total distance D of the predictions to the actual requests while ignoring some number Δ of the requests (which correspond to the worst predictions). This gives rise to a Pareto frontier of pairs (D, Δ) for any given prediction and input. They introduce a framework for online algorithms with predictions, and show that their framework provides tight bounds for online Steiner Tree, Steiner forest, facility location and capacitated facility location. The framework requires an online algorithm which is subset-competitive, meaning that its competitive ratio goes up by at most a constant factor if some requests in the input are omitted, and an offline algorithm for the prize-collecting version of the problem, in which the algorithm is allowed to not serve some of the requests by paying a penalty cost.

In the framework, the high-level idea is that the online algorithm is running all the time, but whenever its total cost has doubled, we run the offline approximation algorithm on the *predictions* instead of on the (revealed) input, and serve as many requests as possible while staying within a constant factor of the online cost. Strikingly, this gives a good overall performance independent of whether the predictions were close to the input or not.

In SODA 2022, Sara Ahmadian et al. have introduced a new model of load balancing for very large databases with massive number of queries [4]. Their model was formulated as a two-stage optimization task and they used algorithm design with ML advice [128] to model the pattern of real-time queries. In the network construction stage, the algorithm receives an estimation of the load of each piece of data as the input and outputs a client-server bipartite graph with near optimal approximation guarantee. In the second stage, they considered online load balancing as a problem in which the input is a fixed client-server bipartite graph $G = (C \cup S, E)$ where C represents the set of clients, S represents the set of servers, and for each client v , the edge set E determines the subset $N(v)$ of servers that can be requested by v . The incoming queries arrive one at a time and the clients assign each query to one of their neighbor servers, immediately and irrevocably. The goal is to balance the load of the servers. In terms of the occupancy model, each ball represents an online request with unit weight, while each bin represents a server. The original power of d choices algorithm (d -choice algorithm) looks at d bins uniformly at random in each round and assigns the ball to the bin with the minimum load. In this paper, the parameter d is considered as the upper bound on the number of bins that can be chosen by each ball. Additionally, in contrast to most related literature, the choice of

bins for each ball is neither random nor independent. The authors introduce several analysis techniques for this d -choice algorithm and they show that the algorithm is always constant competitive against the optimal load allocation, and approaches to the optimal allocation asymptotically.

In SODA 2022, Shiddhartha Benerjee et al. [27] consider an online resource allocation problem. In their setting a single supplier allocates a set of divisible resources to N agents. In each round, a single divisible resource becomes available and each agent i has a per-unit value $v_{i,t} \geq 0$ for the resource of this round. The algorithm specifies how to split this resource between agents, immediately and irrevocably. The total utility of agent i is given by $u_i(x) = \sum_t v_{i,t} x_{i,t}$, where $x_{i,t}$ is the fraction of the resource allocated to the agent i in round t . In this paper, the authors have provided an online algorithm which is augmented with a prediction $\hat{V}_i$ of $V_i = \sum_t v_{i,t}$.

Their algorithm is analyzed for two objectives: the social welfare, which is equal to the minimum utility over all agents, and the Nash social welfare, which is equal to the geometric mean of the agents utilities.

In the case of Nash social welfare maximization with perfect predictions, their online algorithm achieves a competitive ratio of $O(\log(\min\{N, T\}))$, and they have shown that no online algorithm can achieve a competitive ratio better than $O(\log^{1-\varepsilon} N)$ or $O(\log^{1-\varepsilon} T)$ for any constant $\varepsilon > 0$. Additionally, in the case of social welfare maximization they have shown no online algorithm can achieve a competitive ratio better than $O(\sqrt{N})$ even if it is equipped with perfect predictions.

Typically, an online vector balancing problem asks to assign positive or negative signs to a sequence of vectors that arrive online one at a time from a normed vector space, such that the L_∞ norm of the total signed prefix-sum (the discrepancy) is as small as possible. Recent works have studied this problem only for an oblivious adversary, such that the choice of arriving vector does not depend on the internal state of algorithm. In SODA 2022, Anupam Gupta et al. [84] introduce a fully-dynamic input model consisting of both arrivals and departures at each time step.

In this paper, they present an algorithm for the fully dynamic vector balancing problem with update vectors in $[-1, 1]^n$, based on results of Barany and Grinberg [29].

Their algorithm maintains discrepancy of $O(\sqrt{n})$ using amortised recourse of

$O(n \log N)$ per update operation, where N is the maximum number of active vectors at any point in time.

Additionally, they introduce a fully dynamic edge orientation problem based on a special case of the carpooling problem [70, 5]. In this problem, we are given a set V of n vertices and at each time step an adaptive adversary either inserts a new undirected edge $e_t = \{u, v\}$ or removes an existing edge e from a graph $G(t)$. Upon arrival of an edge $\{u, v\}$, it has to be oriented as either $u \rightarrow v$ or $v \rightarrow u$. However, the algorithm is allowed to re-orient any edge e , and the amortized recourse is the average number of re-orientations per edge insertion or deletion. Our goal is to minimize the discrepancy of this orientation at any time. The authors show existence of an algorithm for fully dynamic edge orientation problem with $\text{polylog}(n)$ discrepancy and amortized recourse.

2.7 Prophet Inequalities

In the classic set cover problem one is given a set $\mathcal{U}$ of N elements (universe), a subset $U \subseteq \mathcal{U}$ of n elements and a collection $\mathcal{S}$ of m sets, where each set of this collection is associated with a non-negative cost. The objective is to find a cover of minimum cost, that is, a subcollection S of $\mathcal{S}$ which contains all items in U . In an online setting the items that the algorithm has to cover arrive one at a time (the set U is revealed over time). Various other versions have also been studied.

Anupam Gupta et al. [86] consider the *prophet* version in SODA 2024. Here the set U consists of draws from a sequence of non-identical distributions $D_1, \dots, D_n$. Their main contribution is a reduction from the prophet setting to the *random-order online* setting, where the elements of U are chosen by an adversary but revealed in a random order to the algorithm. Since the algorithm LEARNORCOVER [85] presented by a subset of the authors in FOCS 2021 is $O(\log(mn))$ in the random-order online setting, this also gives a polynomial-time $O(\log(mn))$ -competitive for the prophet setting. The algorithm is *universal*, so it a priori maps each element $u \in \mathcal{U}$ to a set $S(u)$ without knowing U and then outputs $S(U)$ once U is revealed.

They also give results for many other settings. For example the *2-stage prophet* setting, where before any element of U is revealed, the algorithm is allowed to purchase sets at a discount and for the *online-with-a-sample* setting, where a random α -fraction of U is revealed to the algorithm in the beginning.

Sariel Har-Peled et al. consider prophet inequalities in [92]. In this problem one is confronted with realizations of random variables in an online manner and can either accept or reject the current value irrevocably. The objective is to select the highest value. The performance of a strategy can be measured by different metrics. Either by the expected result (competitive ratio or *Ratio of Expectations (RoE)*) or by the *Probability of selecting the Maximum realization* ($\mathbb{P}_{\max}$). The authors analyze a model in which the algorithm is permitted to query an oracle up to m times. Each time, the oracle responds by indicating whether the current realization has the highest value among the remaining realizations. They show a relation to the TOP-1-OF- m model, where an algorithm can select up to m values and its result is measured by the maximum. If the objective is to maximize $\mathbb{P}_{\max}$ the oracle model with m oracle calls is equivalent to the TOP-1-OF- $(m+1)$ model. They present a single-threshold algorithm that with probability at least $1 - O(m^{-m/5})$ selects the maximum of iid random variables, which represents new results for general m and improvements for small m for the TOP-1-OF- $(m+1)$ model. For the RoE objective this equivalence fails to hold. In this case the oracle model is superior in the sense that results translate to results in the TOP-1-OF- m model. For the oracle model with non-iid random variables they give an optimal $1 - e^{-\xi_m}$ -competitive single-threshold algorithm, where ξ_m is the unique positive solution to the equation $1 - e^{-\xi_m} = \frac{\Gamma(m+1, \xi_m)}{m!}$.

Vasilis Livanos and Ruta Mehta [125] consider an alternative version of this problem in which one is interested in minimizing the cost. That is, the goal is to select a value that is as low as possible. The values are drawn independently from a known distribution. The authors show that if the expected value of the distribution is unbounded, then the competitive ratio is also unbounded. For bounded expected values, the competitive ratio of the optimal stopping rule is a distribution dependent constant, which is characterized as the solution to a simple inequality. For certain distributions, optimal results are obtained. Finally, an optimal single-threshold strategy is presented, which is $O(\text{polylog } n)$ -competitive.

In STOC 2024, Andrés Cristi and Bruno Ziliotto [57] consider the prophet inequality problem, focusing on how a gambler can determine the optimal moment to stop a sequence of n independent random variables, presented one by one, to maximize the most recent value as a reward, particularly in scenarios where the gambler has access to only a limited number of samples from each distribution.

The authors explore three variants of the problem: the prophet-secretary model,

where the variables arrive in a uniformly random order; the free-order model, in which the gambler can choose the order of arrival; and the i.i.d. model, where all distributions are identical. They demonstrate that a constant number of samples (independent of n) from each distribution is sufficient to approximate the optimal ratios effectively for these variants. Their proof combines minimax arguments with instance decompositions, establishing that the optimal ratio can be achieved without knowledge of the upper tails of the distributions. They complement their results by showing that the proposed algorithms are polynomial-time implementable.

2.8 Various Online Problems

In ICALP 2022, William Kuszmaul and Shyan Narayanan [115] consider a variation of the p -processor cup game. In the classical version two players, a filler and an emptier play against each other. There are n cups and at the start of each round the filler can distribute p volume of water over all the cups (possibly in fractional amounts) with the restriction that it can place at most 1 volume of water in any particular cup. After the filler's move, the emptier can select p distinct cups and remove up to 1 volume of water from each of them. The goal is to minimize the backlog, so the maximum fill level of any cup over all t rounds. This is called the *fixed* p -processor cup game and its optimal backlog is $\Theta(\log n)$ which is independent of n . If the number p of the processors can change between the rounds the problem is called *variable* p -processor cup game. The authors show that the optimal backlog is $\Theta(n)$, but only if the number of rounds is in $\Omega(n^3)$. If the number of rounds is smaller even better results can be obtained. Further, they show that the greedy emptying algorithm is actually the exact optimal online algorithm. If the emptier has access to resource augmentation, so it is allowed to remove $1 + \varepsilon$ instead of 1 unit of water from the cups the optimal backlog can be reduced to $O\left(\frac{1}{\varepsilon} \log n\right)$. However, if instead the filler can only change the number p every $\text{poly}(n)$ time steps, the optimal backlog is unaffected.

In ESA 2022 Rajmohan Rajaraman and Omer Wasim study the online balanced graph re-partitioning problem (OBGR) [140]. In OBGR a graph is partitioned in a set of ℓ clusters, each holding at most k vertices. Communication requests between two nodes (u, v) arrive online. If u and v are located in different clusters the request incurs unit communication cost and none otherwise. To have any chance of staying competitive it is possible to migrate any vertex from one cluster to another at a migration cost of $\alpha > 1$. The objective is to minimize

total communication and migration cost. Previously to this work upper bounds of $O(k^2\ell^2)$ [21] and $O(k\ell 2^{O(k)})$ [34] (both by algorithms that run in exponential time) and a lower bound of $\Omega(k\ell)$ were known. The authors significantly narrow this gap by presenting an polynomial-time $O(k\ell \log k)$ -competitive algorithm and complement it by a lower bound of $\Omega(\alpha k\ell \log k)$ for algorithms under the learning model. Here learning model refers to the condition that there exists a feasible assignment of vertices to clusters without any inter-cluster requests in the sequence. Hence, to stay competitive an online algorithm must learn an optimal partitioning.

The joint replenishment problem is a fundamental model in supply chain management theory with applications in inventory management. In this problem, there are different item types and each has a time-dependent sequence of demands. Each demand must be satisfied by ordering its corresponding item type. The cost of placing an order includes the fixed joint ordering cost, which is independent of the number of different items in the order, and item ordering cost that depends on the number of different types of items included (but not on the number of items per type). The overall goal is to fulfill all demands and minimizing the total cost incurred. In the deadline version of this problem, each request must be served by its deadline. In the delay variant, demands also have a delay cost function and the goal is to minimize the total service and delay costs.

A generalization of this problem is online network design with delays, studied already in FOCS 2020 by Yossi Azar and Noam Touitou [24] in a paper that we appear to have overlooked completely for that year's survey (sorry, Yossi and Noam!). In that setting, the cost of serving a set of connectivity requests is the cost of the subgraph of G that satisfies the requests. They provided a framework to reduce a network design problem with delay or deadline to the classic offline variant without delay or deadline while incurring a logarithmic loss in the competitive ratio. In ISAAC 2021, Touitou [152] showed that a sub-logarithmic competitive ratio is not possible for this problem in its full generality.

In ICALP 2022 Ryder Chen et al. [51] introduced online weighted cardinality joint replenishment problem with delay as a natural generalization for the online joint replenishment problem (JRP) with delay under a make-to-order coordination mechanism [58]. In the online weighted cardinality JRP with delays, each item type i has an associated weight w_i , and the cost of satisfying a set of demands on a set λ of item types is $f(\sum_{i \in \lambda} w_i)$, where f is an initially given non-negative and non-decreasing concave function. They obtained a constant

competitive algorithm for this problem, which is a refinement of the Azar-Touitou framework [24] using ideas by Gupta et al. [87]. Specifically, their algorithm keeps track of how much has been “invested” in each request in order to make better decisions. They also show that using the Azar-Touitou framework directly leads to a logarithmic competitive ratio at best.

Nick Gravin, Enez Sun and Zhihao Gavin Tang [83] consider the relative performance of ordinal and cardinal online algorithms for two fundamental problems in FOCS 2023. It is well known that sorting a list of length n requires $\Omega(n \log n)$ running time. What is possibly less well-known is that this barrier only holds for ordinal algorithms, that only do pairwise comparisons between the input elements. If you are allowed to look at the values (that is, use a cardinal algorithm) and all of the values are in the range from 0 to some value K , there is a randomized $O(\sqrt{n \log \log K})$ algorithm [150] as well as a deterministic $O(n \log \log K)$ algorithm [91] in the word RAM model of computations. Intuition suggests and Martin Gardner [78] argued (informally) that for large enough numbers, ordinal and cardinal problems are the same. The authors prove this formally for two fundamental problems and give explicit bounds for the maximum value needed so that a cardinal algorithm has at most a factor of $1 + \varepsilon$ advantage. As could perhaps be expected, the numbers are exceedingly large, requiring the use of Graham’s double arrow notation.

Another paper in SODA, by Étienne Bamas et al. [25], gives an in-depth analysis of several greedy algorithms for the online Steiner Forest problem. In this problem, a set of k terminal pairs (pairs of vertices in a weighted graph) need to be connected (by selecting edges from the graph), and the goal is to find a minimum weight subgraph (which is always a forest) which connects all the pairs. In the online version of this problem, the pairs are revealed one by one. Already for the Steiner Tree problem, where a set of k terminals needs to be connected, no algorithm can be better than $\Omega(\log k)$ -competitive, and it has long been known that the greedy algorithm is $O(\log^2 k)$ -competitive. Whether greedy is even better than this, potentially $O(\log k)$ -competitive, remains a big open question. This paper makes some progress on this question by giving several structural results for Greedy, culminating in a proof that if the sequence of costs paid by Greedy or the sequence of distances of the pairs in the input are non-increasing, then Greedy is $O(\log k \log \log k)$ -competitive. This excludes a large class of instances and the authors conjecture because of this that Greedy is $O(\log^{1+\varepsilon} k)$ -competitive or better.

In SODA 2023 Haim Kaplan et al. [107] analyze Meyerson’s algorithm for the

facility location problem in the random-order model. The facility location problem involves a metric space (M, d) with a non-negative and symmetric distance function d satisfying the triangle inequality. A multiset of demand points U in M and a facility opening cost f are given. The goal is to minimize the total cost, which includes the facility opening cost and the assignment cost incurred when demand points are assigned to open facilities. In the online random-order model, we reduce the adversary's power by assuming that the demand points arrive in random order. Therefore, the input sequence for an algorithm is a uniformly random permutation of the demand points in U . When a demand point v arrives, the online algorithm must decide immediately, whether and where to open new facilities. Then, v is irrevocably assigned to its nearest open facility.

Meyerson's algorithm [131] operates by measuring the distance from a new demand point to the closest already-open facility, denoted as d . It opens a new facility with probability $\min\{d/f, 1\}$, or assigns the demand point to the nearest open facility. The authors in [107] refer to Meyerson's algorithm by the name DistProb. They prove that DistProb in the random-order model has an exact competitive ratio of 4 and hence settling a long-standing open question. They also examine DistProb's performance in partial adversarial order settings, demonstrating its competitive ratio remains close to that in the random-order case.

In ESA 2023, Martin Bullinger and René Romen [42] consider the online coalition formation problem where agents arrive in a random order. The concept of coalition formation involves dividing a group of n agents into separate coalitions based on their preferences. In the online variant, agents arrive sequentially and must be assigned to a coalition immediately. The objective is to maximize social welfare.

In a fully deterministic model, the greedy algorithm which assigns an agent to the coalition with the most significant benefit, achieves an optimal competitive ratio, depending heavily on the range of utilities. Their research aims to eliminate the influence of utility dependencies on the competitive ratio. The authors study a scenario where agents arrive in a random order. In this case, the competitive ratio of the greedy algorithm is shown to be $\Theta(\frac{1}{n^2})$, while an alternative algorithm that alternates between waiting and greedy phases can achieve a competitive ratio of $\Theta(\frac{1}{n})$. Furthermore, they consider a model that relaxes the irrevocability of decisions, allowing coalitions to be dissolved into singleton coalitions. For this model a matching-based algorithm is presented, and once again they achieve a

competitive ratio of $\Theta(\frac{1}{n})$.

Yann Disser et al. [63] considered the incremental maximization framework introduced by Bernstein et al. [31]. This is a framework for problems that grow over time and in which a solution needs to be grown along with it while maintaining competitiveness. An instance of such a problem consists of a countable set of elements and a monotone objective function. A solution for such a problem is an order of the elements such that each prefix yields a good solution with respect to the objective function.

It is reasonable to consider only so-called accountable goal functions, which have the property that removing a single item from a set does not make the function value decrease too steeply. Otherwise it is not possible to have algorithms that are competitive. Accountable functions occur in many previously studied settings like weighted matching, set packing and submodular function maximization.

The authors show that problems like these can always be partitioned into a countable set of uniform and modular subsets such that the overall objective is the maximum over the modular functions on the subsets. They then consider the continuization of this problem in which arbitrary fractions of elements can be selected, so that the goal function of the current selection grows continuously. They show that a greedy continuous algorithm is $(\varphi + 1)$ -competitive and provide evidence that this ratio cannot be improved. They also give a 1.772-competitive randomized algorithm and show that any randomized algorithm is at least 1.447-competitive.

In ESA 2024, Mikkel Abrahamsen et al. [2] consider the *online sorting problem* recently introduced by Aamand et al. [1]. In this problem, a sequence of real values $x_1, x_2, \dots, x_n$ is revealed one by one, and each item must be assigned immediately and irrevocably to a size- n array A . The objective is to minimize $\sum_{i=1}^{n-1} |A[i+1] - A[i]|$, the sum of absolute differences between items in consecutive cells.

The authors address an open question from Aamand et al. regarding the potential of randomized algorithms to improve the competitive ratio for the online sorting problem. Their results demonstrate that randomization does not improve the competitive ratio for the online sorting problem and both deterministic and randomized algorithms for this problem have a competitive ratio of $\Theta(\sqrt{n})$. Furthermore, the lower bound $\Omega(\sqrt{n})$ holds even when the input numbers are from $[0, 1]$. Their second main contribution is a generalization of the online sorting problem to higher dimensions, which can be viewed as an online variant

of the classical Traveling Salesperson Problem (TSP) in $\mathbb{R}^d$, using Euclidean distance between items. They consider this generalized online sorting problem for arbitrary fixed d in an adversarial model and demonstrate a competitiveness guarantee close to the one-dimensional case.

In addition, Abrahamsen et al. extend the problem to a stochastic variant, where input values are drawn independently and uniformly at random from a fixed interval. In this setting, they designed an algorithm that achieves competitive ratio of $O((n \log n)^{1/4})$ with probability at least $1 - 2/n$.

Motivated by the task of partitioning news articles that arrive over time into clusters of similar articles, in APPROX 2024 Benjamin Moseley et al. [132] consider the *online k -median clustering problem*. In a metric space n points x_j arrive online and must be immediately assigned to one of k clusters. Each cluster C_i assigns its geometric median as the center c_i of the cluster. The objective is then to minimize the k -median $\sum_{i=1}^k \sum_{x_j \in C_i} d(x_j, c_i)$, where $d(x, y)$ denotes the distance function for the metric space. Even for $k = n = 2$ there exist small examples that show that the competitive ratio can only be bounded by the aspect ratio of the points, so the ratio between the distance of the furthest pair of points and the distance between the closest pair of points. In particular the competitive ratio cannot be bounded by a function of k or n which may be much smaller than the aspect ratio. The problem is that algorithms do not have a sense of scale regarding what is close or far without any additional knowledge about the instance. This motivates the authors to provide the algorithm with an upper bound on the optimal objective value at the start. They measure an online algorithm's performance against this upper bound. First, the authors show that any deterministic algorithm for this problem is $\Omega(k)$ -competitive. They follow up the lower bound with a poly-time algorithm whose competitive ratio depends only on k and not n or Δ . To be precise, it is $O(k^5 3^k)$ -competitive.

In STOC 2024, Janardhan Kulkarni et al. [113] introduce new results for *online discrepancy minimization*. In this problem, we receive vectors $v_1, \dots, v_T \in \mathbb{R}^n$, which are bounded in some norm, for example $\|v_i\|_2 \leq 1$. The vectors arrive one at a time and upon receiving each vector v_i , we must immediately and irrevocably decide a sign $x_i \in \{-1, 1\}$. The objective is to minimize the discrepancy, defined as the signed sum $\sum_{i=1}^T x_i v_i$ in a given norm, such as infinity norm. A natural variant of this problem, called the prefix discrepancy problem, where the goal is to keep the prefix sums $\sum_{i=1}^t x_i v_i$ bounded for all $t \in [T]$.

Kulkarni et al. proved the existence of an online algorithm that against any

oblivious adversary and for any sequence of vectors $v_1, \dots, v_T \in \mathbb{R}^n$ with $\|v_i\|_2 \leq 1$, decides random signs $x_1, \dots, x_T \in \{-1, 1\}$, such that for every $t \in [T]$, the prefix sum $\sum_{i=1}^t x_i v_i$ is 10-subgaussian. Additionally, they showed a matching $\Omega(\sqrt{\log T})$ strategy for an oblivious adversary.

Their framework also extends to *online edge orientation*. They showed the existence of an online algorithm that can decide edge orientations such that for every vertex, the absolute difference between the in-degree and out-degree remains bounded by $O(\sqrt{\log T})$ with high probability.

In WAOA 2024, Philip Whittington [156] explores the concept of *online string attractors*. A string attractor (k -attractor) is defined as a set of positions $\Gamma \subseteq [1, n]$ in a string $T \in \Sigma^n$, such that every distinct substring (of length at most k) contains at least one of these selected positions.

Whittington introduces a natural online variant for the k -attractor and string attractor problems, demonstrating that Lempel-Ziv factorization is the optimal online algorithm for this string attractor problem, with a competitive ratio bounded by $O(\log(n))$. Additionally, the paper shows that the online k -attractor problem for constant k has a strict competitive ratio of k . Furthermore, the author highlights families of sparse strings that have constant-size optimal attractors and shows that any online algorithm for the Fibonacci and Thue-Morse words incurs a cost that grows with the length of the word, for a matching lower bound of $\Omega(\log(n))$.

CHAPTER 3

Minimizing the Total Completion Time

3.1 Introduction

In this chapter, we study online algorithms for the single machine scheduling problem with restarts, focusing on the objective of minimizing the total completion time. The problem is denoted by $1|r_j, \text{online}, \text{restart}|\sum C_j$.

We use the notation and terminology introduced in Chapter 1. For clarity, we briefly restate the key elements. We denote by r_i and p_i , the release date and the processing time of job J_i , respectively. Given a specific set of jobs $S = \{J_1, \dots, J_n\}$, we use r_i^S and p_i^S to denote the release date and the processing time of job J_i ($i = 1, \dots, n$), respectively. Throughout this chapter, we use σ to denote the schedule of the algorithm and π^* to denote an optimal offline schedule. The completion time of J_i in a schedule ψ will be denoted by ψ_i . The relative completion time of a job J_i in σ compared to a feasible schedule ψ is $rci_i^\psi = \sigma_i / \psi_i$.

The Restart Shortest Processing Time (RSPT) algorithm [154] is an online scheduling strategy that achieves a deterministic competitive ratio of $3/2$ for minimizing total completion time on a single machine with using restarts, surpassing the lower bound of $e/(e-1) \approx 1.582$ for randomized non-preemptive online algorithms [48]. In RSPT algorithm, the currently running job J_i which started most recently at time s is only interrupted if there exists an incoming job J_q that can complete before $2/3$ times the completion time of the running job (if left to run). This interruption condition can be represented by the equation

$$r_q + p_q \leq \frac{2}{3}(s + p_i).$$

The current lower bound for this problem is 1.21 and is due to Epstein and van Stee [66]. It uses an exponentially growing (but relatively short) sequence of

jobs interspersed with jobs of (essentially) zero size which arrive at inopportune moments, forcing restarts in each step.

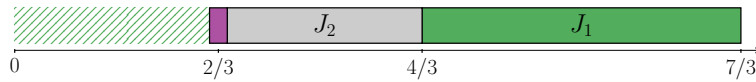
Table 3.1: Example set of jobs for the RSPT algorithm

Jobs(J_j)	Release time (r_j)	Processing time (p_j)
J_1	0	1
J_2	ε	$2/3$
J_3	$2/3$	0

Here, we present a numerical example of RSPT. Consider the following instance, where we have three jobs: J_1 , J_2 , and J_3 , each with their respective release times and processing times. Job J_1 is available from time 0 with a processing time of 1, job J_2 arrives at a small positive release time ε and requires $2/3$ units of time, and job J_3 is released at time $2/3$ with a processing time of 0. The input is provided in the Table 3.1.

Initially, the RSPT algorithm begins processing J_1 at time 0. When J_2 arrives at time ε , the algorithm evaluates whether it should interrupt J_1 based on the interruption condition. Since this condition does not hold, the algorithm continues processing J_1 and only starts J_2 after completing J_1 . When J_3 arrives at time $2/3$, the interruption condition holds, meaning the algorithm interrupts J_1 and starts processing J_3 immediately. Once J_3 completes, the algorithm selects the next available job. In this case, both J_1 and J_2 are available. The algorithm selects J_2 first since its processing time is smaller. Finally, J_1 restarts and runs to completion.

RSPT: $\sum C_j = 2/3 + 4/3 + 7/3 = 13/3$



OPT: $\sum C_j = 2(2/3 + \varepsilon) + 5/3 + \varepsilon = 9/3 + 3\varepsilon$

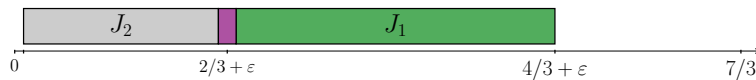


Figure 3.1: Illustration of the schedule for the jobs from Table 3.1

The optimal offline algorithm, which has complete knowledge of all job arrivals in advance, would schedule the jobs differently to minimize the total completion time. Instead of starting with J_1 , the optimal algorithm waits until J_2 arrives at

ε and then processes J_2 first. This strategy ensures that all jobs finish as early as possible, leading to a lower total completion time.

The above example demonstrates that the RSPT algorithm does not always function correctly in every scenario. Specifically, there are cases where the algorithm's interruption condition does not behave as expected, leading to incorrect scheduling decisions.

3.1.1 Our Results

In our study, we consider the power of restarts to achieve a competitive ratio below $3/2$. The primary contribution of this chapter lies in presenting an online algorithm for the problem which achieves a competitive ratio better than 1.4568 (the real root of the equation $6 + 3\rho^2 = 4\rho^3$). The analysis of our algorithm is tight. This is the first improvement for this problem in twenty years and the analysis has become easier and shorter than in [154] for the previous best result.

3.2 Improved RSPT algorithm (IRSPT)

Our new algorithm introduces a more nuanced interruption condition that taken into account the size of the incoming jobs more. At any time when the machine is idle or the algorithm completes a job, the algorithm starts processing the smallest available job. If there is a choice, it takes the job with the smallest release date. Consider a job J_i and an incoming job J_q . There are two conditions for interrupting a job. The incoming job must be smaller than the running job and

$$r_q + p_q < \frac{1}{R}(s_i + p_i + p_q), \quad (3.1)$$

where R is the real root of the equation $6 + 3\rho^2 = 4\rho^3$. For any incoming job that satisfies (3.1) and that is smaller than J_i , the algorithm interrupts J_i and starts processing of J_q at time r_q . The interrupted job loses all its previous progress and must be started again later.

In (3.1), the value $r_q + p_q$ is the completion time of J_q if it starts at time r_q , whereas the value $s_i + p_i + p_q$ is the completion time of J_q if it starts immediately after J_i completes. This inequality implies that an incoming job J_q will only cause an interruption if it can complete before time $\frac{1}{R}(s_i + p_i + p_q)$ after starting at its release date. This means we start J_q immediately if its relative completion time compared to the optimal schedule could otherwise be larger than R . An interruption based solely on this inequality is not always beneficial, as there may

be incoming jobs larger than p_i that also satisfy the inequality. Therefore, as mentioned we refine our rule with a second condition, so that the algorithm only restarts for incoming jobs smaller than p_i . Figure 3.2 illustrates an example of our restarting rule.

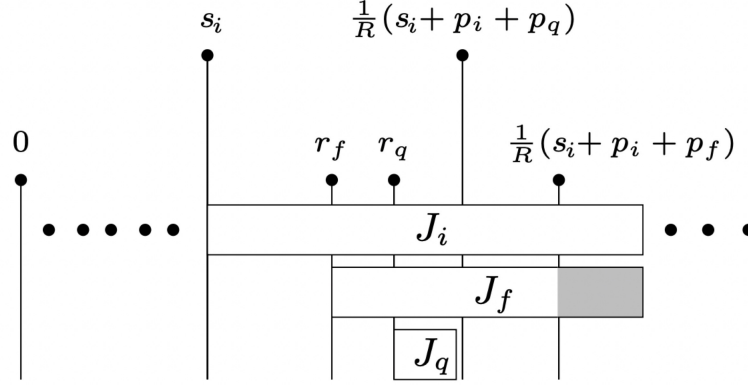


Figure 3.2: Illustration of incoming jobs J_f and J_q . The job J_f does not cause an interruption of J_i , but the job J_q does.

Next, we present a numerical example of IRSPT algorithm. The details of input is provided in the Table 3.2.

Table 3.2: Example set of jobs for the IRSPT algorithm

Jobs(J_j)	Release time (r_j)	Processing time (p_j)
J_1	0	1.5
J_2	0	1
J_3	1.5	0.1
J_4	2.6	0.5

Consider the following instance, where we have four jobs: J_1 , J_2 , J_3 , and J_4 , with their respective release times and processing times. Jobs J_1 and J_2 are available from time 0 with a processing time of 1.5 and 1, respectively. A job arrives at time 1.5 and requires 0.1 units of time, and job J_4 is released at time 2.6 with a processing time of 0.5.

Initially, the IRSPT will begin processing J_2 at time 0. As it processes J_2 , it continuously monitors incoming jobs. After J_2 completed, algorithm starts J_1 .

When J_3 arrives at time 1.5, IRSPT evaluates whether it should interrupt J_1 .

IRSPT:



OPT:

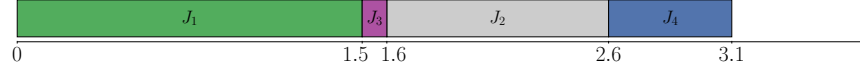


Figure 3.3: Illustration of the schedule at time 0 for the jobs from Table 3.2

This condition holds true and the algorithm will interrupt J_1 and start processing J_3 instead. After completing J_3 , the algorithm can then restart J_1 , where it is the only available job. In this instance, J_4 started immediately after J_1 completes.

IRSPT: $\sum C_j = 1 + 1.6 + 3.1 + 3.6 = 9.3$



OPT: $\sum C_j = 1.5 + 1.6 + 2.6 + 3.1 = 8.8$



Figure 3.4: Illustration of the schedule at time 0 for the jobs from Table 3.2

The total completion time is thus derived from the individual completion times of all jobs scheduled with the IRSPT algorithm.

3.3 IRSPT-like algorithms

It can be seen that the competitive ratio of IRSPT does not improve beyond R : consider a job of size 1 that arrives at time 0, and N jobs of size 0 that arrive at time $1/R + \varepsilon$. IRSPT will run these jobs in order of release date and have a total completion time of $N + 1$. However, it is possible to achieve a total completion time of $(1/R + \varepsilon)(N + 1) + 1$ by running the jobs of size 0 first. Therefore, as N grows and ε tends to 0, the competitive ratio tends to R . Generally, we define a family of algorithms for the problem $1|r_j, \text{online}, \text{restart}|\sum C_j$. Any IRSPT-like algorithm behaves exactly like IRSPT algorithm, but with using a value ρ instead of R .

We show in the next theorem that no algorithm from the IRSPT-like family outperforms IRSPT for any value of ρ .

Theorem 3.1. *For the problem $1|r_j, \text{online}, \text{restart}|\sum C_j$, any IRSPT-like algorithm with $1 < \rho < 2$ has a competitive ratio of at least R (that is, the real root of the equation $6 + 3\rho^2 = 4\rho^3$).*

Proof. Consider an IRSPT-like algorithm ALG and the following job sequence: a job J_1 of size $1 + \varepsilon$ arrives at time 0, a job J_2 of size 1 at time $\frac{2-\rho}{\rho}$, and a job J_3 of size 0 at time $\frac{2-\varepsilon}{\rho^2}$. Both J_2 and J_3 cause interruptions upon their release dates: J_2 interrupts J_1 and J_3 in turn interrupts J_2 . As ε approaches 0, if $\rho < \sqrt{2}$ then the optimal cost for this sequence tends to $2 + \frac{4}{\rho^2}$ (using the order J_1, J_3, J_2 and noting that $r_3 > 1 + \varepsilon$ for ε small enough and $\rho^2 < 2$), while the cost of ALG tends to $\frac{6}{\rho^2} + 3$ (using the order J_3, J_2, J_1). Therefore, $\mathcal{R}(ALG) \geq 1.5$. Otherwise, if $\rho \geq \sqrt{2}$ then the optimal cost for this sequence tends to 4 (using the order J_1, J_3, J_2), while the cost of ALG still tends to $\frac{6}{\rho^2} + 3$ (using the order J_3, J_2, J_1). Note that the competitive ratio of ALG is at least ρ . Hence, $\mathcal{R}(ALG) \geq \max\left(\rho, \frac{6+3\rho^2}{4\rho^2}\right) \geq R$, where R is the real root of the equation $6 + 3\rho^2 = 4\rho^3$. $\square$

Note that In this thesis we have used jobs of size 0. If it is required that job sizes are strictly positive, our upper bound of course still holds, and our lower bounds also continue to hold by using jobs of arbitrarily small size instead of size 0.

3.4 Lower Bound

In this section, we give a new lower bound for scheduling with k -limited restarts. A generalization of restart and limited restart is k -limited restart. Under the assumption of k -limited restart, a job can be interrupted at most k times. Then limited restart is equivalent to 1-limited restart, and restart is equivalent to ∞ -limited restart. We provide a bound for the problem with k -limited restart.

Lemma 3.1. *Any algorithm for the problem $1|r_j, \text{online}, k\text{-restart}|\sum C_j$ is at least 2-competitive.*

Proof. For a contradiction, suppose that there exists an online algorithm A with a competitive ratio R of less than 2. We consider the following job instance provided by the adversary. At time $r_1 = 0$, job J_1 with size $p_1 = 1$ arrives. Suppose that algorithm starts to process J_i at the time S_i . In order to lower bound the competitive ratio of algorithm, we consider two cases for the value of S_i . If S_i is too large, job J_i has a high relative completion time and the

input sequence continues with a new job of significantly larger size arriving after J_i is completed such that $p_{i+1} > \frac{R-1}{2-R}r_{i+1}$. This scenario leads to inefficient scheduling of J_i and adversely affects the overall cost of the algorithm. On the other hand, if S_i is small, we force the algorithm to interrupt J_i . Again, the input sequence continues with the arrival of a new significantly larger job, after J_i is completed as above. However, since the problem allows for a constant number of restarts k , this interruption will utilize one of the permitted restarts.

Case 1. If $S_i \geq \frac{p_i}{R-1}$, then no more jobs arrive during the job J_i , the scheduling time of J_i is independent from other jobs because it arrives when every job is completed in both ALG and OPT (if it exists). It is not difficult to see the relative completion time of J_1 in this case is greater than R (the relative completion time of J_1 is $S_i + 1 \geq \frac{1}{R-1} + 1 = \frac{R}{R-1} > R$, where $1 < R < 2$). For $i > 1$, we show the relative completion time of J_i when $p_i > \frac{R-1}{2-R}r_i$ is

$$\frac{S_i + p_i}{r_i + p_i} > \frac{S_i + \frac{R-1}{2-R}r_i}{r_i + \frac{R-1}{2-R}r_i} \geq \frac{R \left(\frac{R-1}{2-R}r_i \right)}{(R-1) \left(\frac{1}{2-R}r_i \right)} = R.$$

Case 2. If $S_i < \frac{p_i}{R-1}$, then n jobs with size 0 arrive after time S_i at time $S_i + \varepsilon$, where ε is a small positive number. We have

$$\frac{S_i + p_i}{S_i + \varepsilon} > \frac{RS_i}{S_i + \varepsilon} > R \text{ for all } \varepsilon > 0.$$

Hence, if algorithm A does not interrupt the job J_i then with a sufficiently large value of n , we get a contradiction. Therefore, the algorithm A interrupts the job J_i and starts it again at (say) S'_i . Consequently, the job J_i must either be interrupted or, if it is not interrupted, its relative completion time is greater than R . In both cases, if the competitive ratio so far is less than R , the adversary will add a new job J_{i+1} with $r_{i+1} = S'_i + p_i$ and $p_{i+1} > \frac{R-1}{2-R}r_{i+1}$ to the instance. Then, we have the above cases for the value of S_{i+1} , respectively.

Therefore, after enough new jobs, because the value of k is constant, the algorithm can not restart anymore and the competitive ratio surpasses R as the number of jobs with relative completion time greater than R tends to infinity. Hence, the competitive ratio of any algorithm for the problem $1|r_j, \text{online}, k\text{-restart}|\sum C_j$ is at least 2. $\square$

3.5 Competitive analysis of IRSPT

To simplify the analysis, we will focus exclusively on the IRSPT algorithm. In order to prove that IRSPT has the competitive ratio R , we assume there exists a smallest counterexample consisting of a minimum number of jobs, and show by contradiction that such a counterexample does not exist. We partition the processing of this input into two events, which we will analyze separately.

- Job completions. While the completed job was running, other jobs may have arrived. We will need to show that these jobs complete early enough (on average) and that they do not delay other waiting jobs too much.
- Job restarts. While the interrupted job was running, other jobs may have arrived earlier without causing a restart. The input may also contain larger jobs that get delayed as well due to this interruption.

To better understand the behavior of the IRSPT algorithm, we explore several properties. In our analysis, we will use these properties to upper bound the competitive ratio of the IRSPT algorithm.

Observation 3.1. *Any interruption of a job J_i occurs before time $\frac{1}{R}(s_i + p_i)$ and job J_i cannot be interrupted at $\frac{p_i}{R-1}$ or later.*

Proof. If an interruption of J_i happens by J_q , then

$$r_q + p_q < \frac{1}{R}(s_i + p_i + p_q) = \frac{1}{R}(s_i + p_i) + \frac{1}{R}p_q \Leftrightarrow r_q + \left(1 - \frac{1}{R}\right)p_q < \frac{1}{R}(s_i + p_i).$$

Hence, $r_q < \frac{s_i + p_i}{R}$, meaning that any interruption of J_i is before time $\frac{1}{R}(s_i + p_i)$. Additionally, $r_q < \frac{s_i + p_i}{R} < \frac{r_q + p_i}{R}$ implies $r_q < \frac{p_i}{R-1}$, so J_i cannot be interrupted at $\frac{p_i}{R-1}$ or later. $\square$

Definition 3.1. Let $\delta(t, s, p) = \max(0, (s + p - Rt)/(R - 1))$ be the smallest possible size of an incoming job at time t that does not cause the job J of size p that started at time s (and is still running at time t) to be interrupted.

Observation 3.2. *The algorithm interrupts the processing of J_i for another job J_q if and only if less than $\frac{1}{R}$ of J_q will remain unfinished at time $\frac{1}{R}(s_i + p_i)$.*

Proof. By Observation 3.1, an interruption happens if more than $1 - \frac{1}{R}$ of J_q will be completed strictly before time $\frac{1}{R}(s_i + p_i)$, when J_q is started at time r_q . Therefore, less than $\frac{1}{R}$ of J_q will remain unfinished at time $\frac{1}{R}(s_i + p_i)$. $\square$

Observation 3.3. *A job J_i that starts at time s_i and is still running at time $\frac{1}{R}(s_i + p_i)$ will not be interrupted anymore.*

Proof. This follows immediately from Observation 3.1. $\square$

Lemma 3.2. *Suppose ALG interrupts job J_i at time t . Then, at time t , OPT has not completed any job of size less than p_i that ALG has not completed.*

Proof. At time s_i , ALG has completed all jobs of size less than p_i that have arrived because the algorithm always prefers smaller available jobs. All jobs that arrived in the time interval (s_i, t) are unfinished at time t in any feasible schedule by Observation 3.2, since they do not cause a restart. (In fact, more than $1/R$ of each such job is unfinished) $\square$

We explain several structural properties of a smallest counterexample (with the minimal number of jobs), denoted by I . Let σ be the schedule produced by IRSPT for I . The schedule σ starts at a nonnegative time. A *gap* is a maximal period of contiguous idle time between two scheduled jobs. Since I is a smallest counterexample and the algorithm does not base its decision on anything that happened before the last gap, there are no gaps in σ after the time of the last interruption, and no gaps at all if there was no interruption.

Observation 3.3 implies that for every job of nonzero size, there is an interval of length $\min(p_i, (1 - 1/R)(s_i + p_i))$ in which it will not be restarted. We use this to simplify the input as follows. If a job J_q arrives exactly at a time t when a job of nonzero size completes, we do the following. Let J_i be the unique job of nonzero size that completes at time t . We decrease r_q to some value in the open interval $(\max(s_i, (s_i + p_i)/R), s_i + p_i)$ without affecting the schedule σ created by IRSPT. The optimal cost can only decrease as a result. This adjustment ensures that each job after the last interruption arrives while another job J_i of nonzero size is running (strictly before J_i completes, but after the incoming job could cause an interruption).

We next demonstrate how the instance I can be modified to achieve a simpler and more specific structure. The technique involves constructing a smallest counterexample by adjusting job sizes and release dates. This approach simplifies instances for further analysis and has also been applied in studies of semi-online scheduling algorithms [129, 149].

Definition 3.2. Consider the last release date t in I . We denote the job being processed at that time in σ by J . Let p and r denote its size and release time, respectively and let s be its most recent starting time before t . Let the set of jobs that arrive while J is processing be S . Let $p_{\max}^S$ represent the largest size among the jobs in S and $r_{\max}^S$ be the earliest release date among the jobs of size $p_{\max}^S$.

We will show that without increasing the number of jobs, we can construct a new instance I' by modifying I so that no jobs arrive while J is processing, and the competitive ratio of modified instance does not decrease. First we establish a number of properties of a smallest counterexample that hold without loss of generality.

Property 3.1. While the job J is running, only smaller jobs arrive.

Proof. Suppose not. Then we have $p_{\max}^S \geq p$ and we can set the release date of the job(s) with size $p_{\max}^S$ to time r_i without changing the behavior of the algorithm (apart from possibly the order in which some jobs of size p are being run) or increasing the optimal cost. As a result, such jobs are no longer in S . Therefore, without loss of generality we have $p_{\max}^S < p$. $\square$

Property 3.2. While the job J is running, all jobs of size $p_{\max}^S$ arrive at time $r_{\max}^S$.

Proof. If there are multiple jobs with size $p_{\max}^S$ in S , we can adjust their release dates so that all these jobs arrive at the earliest release date among jobs of size $p_{\max}^S$ in S without changing the processing order in σ (because ALG did not interrupt J with the earliest job of size $p_{\max}^S$) and this does not increase the optimal cost. $\square$

In the next step, when we change the value of $p_{\max}^S$, we change the size of all the jobs of size $p_{\max}^S$ in S (so that all these jobs continue to have size $p_{\max}^S$).

Property 3.3. In the smallest counterexample I , if J is not interrupted at time t , then $p_{\max}^S$ is equal to the maximum of the second largest size in S (taken to be 0 in case $|S| = 1$) and $\delta(r_{\max}^S, s, p)$.

Proof. Consider the cost of the algorithm and the optimal cost as functions of parameter $p_{\max}^S$ as it varies in within the size range $[x_0, p]$, where x_0 is the maximum of the second largest size in S and $\delta(r_{\max}^S, s, p)$. We observe that in the entire size range $[x_0, p]$, the cost of the algorithm is linearly increasing because there are no gaps in its schedule. The optimal cost is a piecewise linear

function (in each piece, a fixed permutation of the jobs is used) with a slope that does not increase with $p_{\max}^S$, thus the optimal cost function is concave. The worst-case instance (the highest competitive ratio) is therefore achieved on one of the endpoints of this size range $[x_0, p]$. Therefore, the competitive ratio is maximized by choosing the value of $p_{\max}^S$ to be either minimal or maximal. If $p_{\max}^S$ is maximal, we can move its arrival time to s exactly as in the proof of Property 1, hence removing the job from S . We conclude that there exists a smallest counterexample I that has Properties 1 and 3. $\square$

Property 3.4. *In the smallest counterexample I , we have $\delta(s, s, p) > 0$.*

Proof. If $\delta(s, s, p) = 0$, we have $s \geq \frac{p}{R-1}$ meaning that J would never be restarted, no matter what other jobs arrive (Observation 3.1). Then we can move the release date of all jobs in S to time $s + \varepsilon$ for some very small ε . This can only decrease the optimal cost and does not change the schedule of ALG. Now all jobs in S have the same release date and will be processed in the same order by ALG and OPT, since no further jobs arrive after time $s + \varepsilon$.

By Property 3 and because $\delta(s, s, p) = 0$, $p_{\max}^S$ is equal to the size of the second largest job in S . Repeating this argument, all jobs in S have the same size, which is 0. However, the relative completion time of all jobs in S is at most R and therefore it cannot be the case that any of these jobs have size 0, because a job of size 0 that has relative completion time of at most R could be removed without affecting the schedule and this would imply that I is not a smallest counterexample and that is a contradiction. $\square$

Property 3.5. *In the smallest counterexample I , at time t , J is interrupted.*

Proof. By Property 3.4, no job arrives during a job with $\delta(s, s, p) = 0$. Therefore, we only need to consider the situation that $s < \frac{p}{R-1}$. For a job J_j denote the time that ALG and OPT start to run it by t_j and t_j^* , respectively. Let d_j be the number of unfinished jobs in ALG schedule at time $c_j = t_j + p_j$ and d_j^* be the number of unfinished jobs in the optimal schedule at time $c_j^* = t_j^* + p_j$. Consider the jobs in S , where $|S| = n$, numbered according to their positions in σ . We define the above variables analogously for the job J_n^S with size $p_{\max}^S$ that ALG starts to run last (among all the jobs in S). when both ALG and OPT complete J_n^S last. If the job J is not interrupted for any job in S , then when both ALG and OPT complete J_n^S last, it is clear that the relative completion time of J_n^S is less than or equal to R . Additionally, if both ALG and OPT complete J_n^S last,

we can assume J_n^S is the last arriving job. Then the contribution of J_n^S to the online cost is $c_n + d_n p_{\max}^S$ and its contribution to the offline cost is $c_n^* + d_n^* p_{\max}^S$. Given that the instance is a smallest counterexample we have $\frac{c_n + d_n p_{\max}^S}{c_n^* + d_n^* p_{\max}^S} > R$, which then implies $d_n/d_n^* > R$. Therefore, the size of J_n^S in the worst-case instance should be maximized and set equal to the size of J and then it can be moved back to the release date of J as in the proof of Property 1. Now we are in the same situation as before, without changing the processing order in σ and without increasing the optimal cost, but with one less job in S .

If OPT does not run J_n^S last, let J_f be the job in S that OPT starts first. Then we can modify the instance I by letting all jobs that arrive before OPT completes J_f to the optimal finishing time of J_f (or to $s + p - \varepsilon$ for some very small ε if that is earlier), without increasing the optimal cost. Furthermore, all jobs that arrive after OPT completes J_f but before ALG completes J can be shifted to arrive at the time at which OPT finishes J_f (or to $s + p - \varepsilon$). Now, all jobs in S except for J_f arrive at the same time and are processed in nondecreasing order after this time in the optimal schedule. Therefore, in the schedule of OPT there is only one job in S , that is out of sequence, and that one job (if it exists) is run first. Hence, if OPT does not run J_n^S last, thus we know that it starts this job first.

Let J_ℓ denote the job in S that OPT starts last and assume OPT starts the first job among all the jobs in S at time t^* . The variables for J_ℓ are defined analogously as before. Because job J is not interrupted by any job in S , the following inequality holds:

$$\frac{c_\ell}{c_\ell^*} = \frac{s + p + \sum_{j=1}^{n-1} p_j^S}{t^* + p_{\max}^S + \sum_{j=1}^{n-1} p_j^S} \leq \frac{s + p + \sum_{j=1}^{n-1} p_j^S}{r_n^S + \frac{1}{R} p_{\max}^S + \sum_{j=1}^{n-1} p_j^S} \leq R.$$

Therefore, the relative completion time of J_ℓ is less than or equal to R . Without loss of generality, we can assume that J_ℓ is the last arriving job, because it does not change the processing order by ALG and OPT. Then the contribution of J_ℓ to the online cost is $c_\ell + d_\ell p_\ell$ and its contribution to the offline cost is $c_\ell^* + d_\ell^* p_\ell$. Since the instance is a smallest counterexample, we have $\frac{c_\ell + d_\ell p_\ell}{c_\ell^* + d_\ell^* p_\ell} > R$ and this immediately means that $d_\ell/d_\ell^* > R$. Hence, if the size of J_ℓ is not equal to J_n^S , it should be maximized and set equal to the size of J_n^S and then it can be moved back to the release date of J_n^S . Therefore, J_n^S and J_ℓ with the same size have the same release date. This allows us to exchange them in the optimal schedule without any change in the optimal cost. As a result, both ALG and OPT complete J_n^S last, and then the release date of this job can be decreased

to time r_i as described above. Repeating this as often as needed (finitely many times) means that S contains no jobs at all, and more generally that no jobs arrive after the last interruption. $\square$

In the following proofs, we will use the concept of credit. The credit of a schedule indicates how much slack there is in the schedule, that is, the total amount by which jobs could be delayed while remaining competitive. That is, for a schedule σ for an input I , we define

$$CREDIT(\sigma) = R \cdot OPT(I) - ALG(I),$$

where $ALG(I)$ is the cost of the schedule σ .

It will often be convenient to consider the credit of individual jobs. If a job J_i is currently scheduled to complete at time σ_i and we know that OPT completes it at time ψ_i , then we define the credit of job J_i to be

$$CREDIT(J_i) = R\psi_i - \sigma_i.$$

In general, the credit of an individual job could be negative, for instance if a job gets interrupted twice but OPT completes it at time 1. It is sufficient for our purposes that the total credit of the input remains nonnegative at all times. We will show that this is always maintained, meaning that credits could be reassigned so that each job again has nonnegative credit. An example is shown in Figure 3.5. The instance consists of only two jobs, which ALG and ALG' schedule in different orders as soon as they become available.

As we are considering a minimal counterexample, and no jobs arrive after the last interruption, we know that the total amount of credit was nonnegative before that last interruption. In our proofs below, we will show that this is maintained and still holds after that interruption. We will derive bounds on the credit of a schedule on a case by case basis depending on the optimal schedule. Consider the job J_i before being interrupted by J_q at time t . Denote the starting time of J_i in the optimal schedule by s_i^* . For $j \in \mathbb{N}$, let D_j denote the set containing J_j and the jobs at least as large as J_j that ALG will complete after it (and have already arrived). Let S_j denote the set containing all jobs smaller than J_j that arrived in the time interval $(s_i, t]$. The jobs in D_j and S_j are numbered according to their position in the schedule of ALG , respectively. We say a job J_j in an instance I *delays* the processing of another job (or jobs) if that other job (or jobs) starts processing earlier in the ALG schedule of the instance $I \setminus \{J_j\}$.

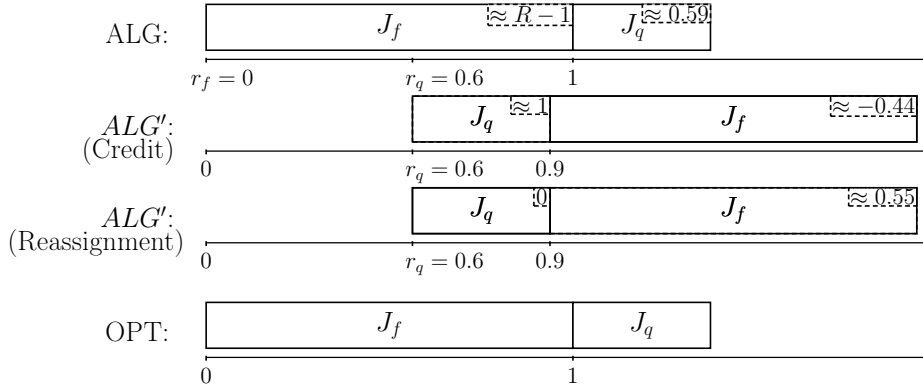


Figure 3.5: Example of credit and credit reassignment

We consider all possible cases for the schedule of OPT based on the number of jobs in D_i that OPT completes before J_q . It is sufficient to consider the possible optimal schedules for the jobs that have arrived by time t . The optimal schedule might change if some jobs arrive later, but obviously the total cost of the existing jobs by time t cannot decrease in such a scenario.

Observation 3.1 indicates that interruptions of job J_i do not occur after time $\frac{p_i}{R-1}$. Therefore, since $\frac{p_i}{R-1} < 3p_i$, OPT completes at most three jobs from D_i before it starts processing J_q . This is because J_q is smaller than every job in D_i , and OPT processes jobs after time t in non-decreasing order of size, under the condition that no other jobs arrive. Hence, we only need to consider a few cases.

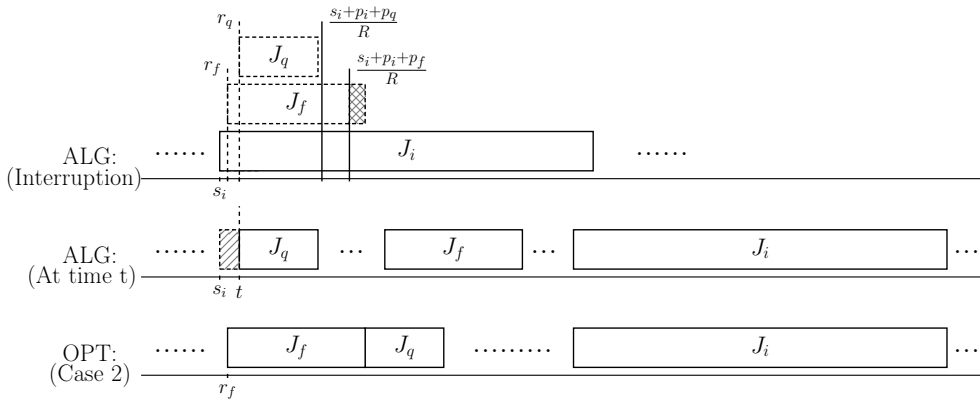


Figure 3.6: Illustration of Case 2 in Lemma 3.3

Lemma 3.3. *If no job from the set D_i is completed before J_q by OPT, then immediately after an interruption of J_i at time t , the total credit is nonnegative.*

Proof. We distinguish two cases depending on the job that OPT runs first.

Case 1. OPT processes the job that causes the interruption first. In this scenario, the same set of unfinished jobs is processed in non-decreasing order of size after time t in both of OPT's and ALG's schedules, since we analyze the schedule immediately after the interruption. Therefore, the credit of J_i is $(R - 1)(t + \sum_j p_j^{S_i} + p_i)$, so that all jobs have nonnegative credit.

Case 2. OPT runs another job first, say a job J_f that arrives at time t_1 . Note that the size of J_f is smaller than J_i because OPT runs J_f first while J_i is also unscheduled at time t_1 . In this case, the jobs in S_f delay the completion time of the other jobs except J_f by the same amount in the schedule of ALG and OPT. We have that OPT completes the ℓ th job in S_f no earlier than at time $t_1 + p_f + \sum_{j=1}^{\ell} p_j^{S_f}$, whereas ALG completes the ℓ th job in S_f at time $t + \sum_{j=1}^{\ell} p_j^{S_f}$. This means the credit of the ℓ th job in S_f is at least $R(t_1 + p_f) - t + (R - 1) \sum_{j=1}^{\ell} p_j^{S_f}$. The job J_f does not satisfy (3.1), so we have $R(t_1 + p_f) - p_f \geq s_i + p_i > t$. Since $p_{\ell}^{S_f} \leq p_f$, it implies that $p_f + (R - 1) \sum_{j=1}^{\ell} p_j^{S_f} \geq p_{\ell}^{S_f}$. The delay of J_f caused by the ℓ th job is at most $p_{\ell}^{S_f}$, and we have shown that the credit of ℓ th job is at least $p_{\ell}^{S_f}$. Therefore, we have enough credit for the delay of J_f . Hence, in the worst-case scenario, S_f only contains the job that caused the interruption (J_q). Now, we consider the jobs in $D_f \setminus \{J_f\}$ (the larger jobs). The job J_f does not satisfy (3.1), so we have

$$\frac{t + p_q + 2p_f}{t_1 + p_q + 2p_f} < \frac{s_i + p_i + p_f + (p_q + p_f)}{t_1 + p_f + (p_q + p_f)} = \frac{s_i + p_i + p_q + 2p_f}{t_1 + p_q + 2p_f} \leq R.$$

Therefore, the relative completion time of $J_2^{D_f}$ is less than R . Hence, for the jobs in $D_f \setminus \{J_f\}$, the relative completion time is at most R because ALG and OPT process these jobs in a same order. Consequently, in the worst-case instance (the highest competitive ratio), D_f consists of J_f , the interrupted job (J_i) and the large jobs (at least as large as J_i) that caused delays before time s_i to the processing of at least one other job. The ratio between the completion time of J_f in ALG schedule and completion time of J_q in the optimal schedule is less than R , since J_f does not satisfy (3.1). This implies the total credit is nonnegative. $\square$

Lemma 3.4. *If OPT completes one job from D_i before J_q (the interrupted job J_i or a job that is at least as large as it), then immediately after an interruption of J_i at time t , the total credit is nonnegative..*

Proof. In this scenario, the interrupted job J_i is delayed by $t + \sum_j p_j^{S_i} - s_i$

because of the restart. We first consider the jobs smaller than J_i that are available at the time of interruption (the set S_i). ALG will complete these jobs sequentially from smallest to largest, while OPT might use a different order. Additionally, OPT may run a job different from J_i (but at least as large as it) first. However, if OPT completes a job at least as large as J_i first, the completion time of all jobs will increase, thus the credits of all jobs increase or remain the same (if the size matches with the size of J_i , we can simply swap the labels and the cost of OPT remains unchanged). Therefore, it is sufficient to only consider the case where OPT runs the interrupted job J_i first. Here, we begin by analyzing the jobs in S_i that did not cause an interruption to J_i . We consider two scenarios based on the interruption time t .

For $t \leq p_i$, under the condition that OPT completes J_i before J_q , OPT completes the ℓ th job in S_i no earlier than at time $p_i + \sum_{j=1}^{\ell} p_j^{S_i}$, whereas ALG completes the ℓ th job in S_i at most at time $t + \sum_{j=1}^{\ell} p_j^{S_i}$. This implies the credit of the ℓ th job in S_i is at least $Rp_i - t + (R-1) \sum_{j=1}^{\ell} p_j^{S_i}$. The additional delay of J_i caused by the ℓ th job in S_i is at most $p_\ell^{S_i}$. Since $Rp_i - t \geq Rp_i - p_i = (R-1)p_i$, the job J_i can receive $(R-1)(p_i + \sum_{j=1}^{\ell} p_j^{S_i})$ credit from the ℓ th job. This credit is greater than $p_\ell^{S_i}$ when $p_\ell^{S_i} \leq \frac{R-1}{2-R}(p_i + \sum_{j=1}^{\ell-1} p_j^{S_i})$.

For $t > p_i$, OPT may choose to run another job J_f in S_i before J_q . Suppose J_f has size p_f and OPT starts it at time t^* . Other jobs in S_i completed by ALG before J_f gain more credit in this scenario. Consider the case that S_i only contains J_q and J_f . It is important to note that J_f did not cause a restart, so its completion time is after $t + p_q$. Therefore, the difference in credit for J_q , compared to when OPT completes J_q first, is

$$R(t^* + p_f + p_q) - R(t + p_q) = R(t^* - t + p_f) > t^* - t + p_q.$$

Hence, we only need to consider the case where OPT also completes J_q first. Thus, the credit of the ℓ th job in S_i is at least $(R-1)(t + \sum_{j=1}^{\ell} p_j^{S_i})$. It follows that the interrupted job can receive $(R-1)(t + \sum_{j=1}^{\ell} p_j^{S_i})$ credit from the ℓ th job. This credit is greater than $p_\ell^{S_i}$ when $p_\ell^{S_i} \leq \frac{R-1}{2-R}(p_i + \sum_{j=1}^{\ell-1} p_j^{S_i})$ as above.

Consequently, if $p_\ell^{S_i} > \frac{R-1}{2-R}(p_i + \sum_{j=1}^{\ell-1} p_j^{S_i})$, in the worst-case instance there is one job in $S_i \setminus \{J_q\}$ and at most $(1 - \frac{R-1}{2-R})p_i$ additional credit is needed for the delay caused by S_i to J_i .

Next, we consider the jobs in $(D_i \setminus \{J_i\})$, which are at least as large as J_i . It is sufficient to analyze the case where OPT processes the jobs in D_i sequentially

from smallest to largest. Additionally, note that the jobs in S_i delay the jobs in D_i by the same amount in both ALG's and OPT's schedules: their total size. Therefore, the only additional delay for all of these jobs is $t - s_i$. Ignoring the sizes of the jobs in S_i , which only improve the relative completion times of jobs in $D_i \setminus \{J_i\}$, we analyze two cases. If $t \leq p_i$, OPT completes the ℓ th job in $D_i \setminus \{J_i\}$ no earlier than at time $p_i + \sum_{j=2}^{\ell} p_j^{D_i}$. Alternatively, if $t > p_i$, OPT completes the ℓ th job in D_i no earlier than at time $t + \sum_{j=2}^{\ell} p_j^{D_i}$. ALG completes this job at time $t + p_i + \sum_{j=2}^{\ell} p_j^{D_i}$ in both cases. Therefore, the relative completion time of the ℓ th job in $D_i \setminus \{J_i\}$ is at most R for $\ell = 3, \dots, k$.

Consequently, we define the worst-case instance where the set S_i consist of job J_q , which caused the interruption, and potentially another job $J_2^{S_i}$ that does not have enough credit. The set D_i consists of the current interrupted job J_i and other interrupted jobs (if any) that ALG will complete after J_i , where these jobs are at least as large as J_i and delayed the processing of other jobs before time s . Moreover, D_i may include an additional job with a relative completion time greater than R under the constraint that S_i exclusively contains J_q .

In the following, we analyze the interruption at time t in the worst-case instance. We consider the following scenarios based on the interruption time t . Recall that the starting time of job J_i in the optimal schedule is denoted by s_i^* . If $t \leq p_i$, ALG completes J_q at time $t + p_q$, and OPT completes J_q no earlier than at time $s_i^* + p_i + p_q$, meaning that $(R - 1)p_q - t + R(s_i^* + p_i)$ credit remains available thereafter, which can be allocated to the other jobs. Alternatively, if $t > p_i$ OPT completes J_q no earlier than at time $t + p_q$, ALG completes this job exactly at time $t + p_q$, meaning that $(R - 1)(p_q + t)$ credit remains available which can be allocated to the other jobs.

In the case that the number of jobs in $D_i \setminus \{J_i\}$ that delayed processing of at least one other job (before time s_i) is $k \neq 0$, we can say that any job in D_i arrives during the last running of its next job in D_i ; otherwise, the larger job that delayed running of another job can not run in the ALG schedule before time s_i . The larger job $J_j^{D_i}$ that delayed processing of at least one other job implies the existence of a job that is released during the running of $J_j^{D_i}$ but does not interrupt it and ALG completes this job in $I \setminus \{J_j^{D_i}\}$ before $s_j^{D_i} + p_j^{D_i}$. Additionally, the algorithm interrupts for every incoming job before time $(2 - R)(s_j^{D_i} + p_j^{D_i})$, that can potentially be completed before the currently running job $J_j^{D_i}$. This is because any incoming job J_f with $p_f < s_j^{D_i} + p_j^{D_i} - r_f$ satisfies (3.1), when $r_f \leq (2 - R)(s_j^{D_i} + p_j^{D_i})$. Therefore, we can observe that in any schedule of OPT, s_i^* is not earlier than $\frac{R^k - 1}{(R - 1)R^k} p_i$. Hence, there is no job in $D_i \setminus \{J_i\}$ with

relative completion time greater than R , and $J_2^{S_i}$ does not exist in the worst-case instance. Now if $t > s_i^* + p_i$, then we show

$$k((1-R)p_q + t - s_i) \leq (R-2)(p_q + t) + s_i,$$

and if $t \leq s_i^* + p_i$, we need to check that

$$k((1-R)p_q + t - s_i) \leq (R-2)p_q - 2t + R(s_i^* + p_i) + s_i.$$

Based on our rule of restarting when $s_i^* \geq \frac{R^k-1}{(R-1)R^k}p_i$, we prove both cases by mathematical induction on k . For the base case $k = 1$, we need to verify that

$$(1-R)p_q + t - s_i \leq (R-2)(p_q + t) + s_i,$$

where $t > s_i^* + p_i$. Rearranging terms, this is equivalent to

$$0 \leq (2R-3)p_q + (R-3)t + 2s_i.$$

This inequality holds when $Rt > (R+1)p_i$, by the interruption rule and the value of R , so the base case is verified. Assuming that the inequality holds for $k = n$, we have

$$n((1-R)p_q + t - s_i) \leq (R-2)(p_q + t) + s_i,$$

where $s_i^* \geq \frac{R^n-1}{(R-1)R^n}p_i$. We must show that the inequality holds for $k = n+1$, which means proving that

$$(n+1)((1-R)p_q + t - s_i) \leq (R-2)(p_q + t) + s_i.$$

After expanding and simplifying the right side in the induction step, we need to check that $(1-R)p_q + t - s_i \leq 0$ and that is true because based on our rule of restarting and $s_i^* \geq \frac{R^{n+1}-1}{(R-1)R^{n+1}}p_i$ we have $(1-R)p_q + t - s_i \leq 2(1-R)p_q - (R-1)t + p_i < 1 - R^{k+1} - R^{k+2} \leq 0$. This confirms that the inequality holds for $k = n+1$. Moreover, if $t \leq s_i^* + p_i$, we need to verify that

$$((1-R)p_q + t - s_i) \leq (R-2)p_q - 2t + R(s_i^* + p_i) + s_i,$$

in the base case $k = 1$. Rearranging terms, this is equivalent to

$$0 \leq (2R-3)p_q - 3t + R(s_i^* + p_i) + 2s_i.$$

This inequality holds under the provided constraints and the value of R as above because

$$(2R - 3)p_q - 3t + R(s_i^* + p_i) + 2s_i \geq (2R - 3)p_q + (R - 3)t + 2s_i,$$

so the base case is verified. Assuming that the inequality holds for $k = n$, we have

$$n((1 - R)p_q + t - s_i) \leq (R - 2)p_q - 2t + R(s_i^* + p_i) + s_i,$$

where $s_i^* \geq \frac{R^n - 1}{(R - 1)R^n} p_i$. We must show that the inequality holds for $k = n + 1$, which means proving that

$$(n + 1)((1 - R)p_q + t - s_i) \leq (R - 2)p_q - 2t + R(s_i^* + p_i) + s_i,$$

where $s_i^* \geq \frac{R^{n+1} - 1}{(R - 1)R^{n+1}} p_i$. After expanding and simplifying the right side in the induction step, we need to check that $(1 - R)p_q + t - s_i \leq \frac{p_i}{R^n}$ and that is true because based on our rule of restarting and $s_i^* \geq \frac{R^{n+1} - 1}{(R - 1)R^{n+1}} p_i$ we have $(1 - R)p_q + t - s_i \leq 2(1 - R)p_q + (1 - R)t + p_i \leq \frac{p_i}{R^{n+1}}$. Therefore, the credit of every job in D_i is always nonnegative.

Now, consider the case that the number of jobs in $D_i \setminus \{J_i\}$ that delayed processing of at least one other job (before time s_i) is 0. Then, it is straightforward to verify that the credit of every large job does not decrease after the interruption when $s_i \geq \frac{2-R}{R-1} p_i \approx 1.18914 p_i$, as we only need to check

$$t + p_q - s_i \leq (R - 1)(p_q + t).$$

Rearranging terms, this is equivalent to $(R - 1)(p_q + t) \leq p_i$ when $s_i \geq \frac{2-R}{R-1} p_i$. Based on our rule of restarting this holds because $s_i < t$. This ensures that the job causing the interruption always has enough credit to compensate for the lost work.

In the case that $s_i < \frac{2-R}{R-1} p_i$, if J_i delays processing of at least one other job (before time s_i), either J_i is started in the optimum schedule no earlier than p_i/R , ensuring that the job causing the interruption always has sufficient credit to compensate for the lost work, or no job requires J_i 's initial credit, which is at least $(R - 1)p_i$. If J_i does not delay the processing of other jobs or no job requires J_i 's initial credit, then all jobs smaller than J_i that arrive when OPT is running J_i have positive relative completion times. This is because OPT completes these smaller jobs after J_i . Therefore, an additional credit gained by having OPT completes a job from D_i before J_q (denoted as K_i) remains. We consider the

interruptions of J_i before it starts at time s_i , since $s_i < \frac{2-R}{R-1}p_i$, the smallest amount of K_i occurs when J_i is interrupted twice by two jobs of size 0 with the maximum possible lost work. The total credit of these two jobs of size 0 exceeds $2(R-1)p_i$, that implies K_i is always more than $(3(R-1) - \frac{2-R}{R-1})p_i \approx 0.18p_i$. Hence, to show that the credit of the large jobs is always nonnegative, we only have the following two cases. If $t \leq p_i$ we have that

$$\begin{aligned} & t + p_q - s_i - \min(R(p_i + p_q + p_2^{D_i}) - (t + p_q + p_i + p_2^{D_i}), 0) \\ & \leq (R-1)p_q - t + Rp_i + K_i. \end{aligned}$$

Otherwise, when $t > p_i$ we have

$$\begin{aligned} & t + p_q - s_i - \min(R(t + p_q + p_2^{D_i}) - (t + p_q + p_i + p_2^{D_i}), 0) \\ & \leq (R-1)(p_q + t) + K_i. \end{aligned}$$

If $t \leq p_i$, assuming $R(p_i + p_q + p_2^{D_i}) - (t + p_q + p_i + p_2^{D_i}) \geq 0$, by the interruption rule and the value of R , we have

$$(2-R)t \leq (2-R)p_i < (3R-3.82)p_i < (2R-3)p_q + (R-1)p_i + 0.18p_i,$$

and if $R(p_i + p_q + p_2^{D_i}) - (t + p_q + p_i + p_2^{D_i}) < 0$ we have

$$(3-R)t \leq (3-R)p_i < (3R-2.82)p_i \leq (3R-4)p_q + (3R-3)p_i + 0.18p_i.$$

Otherwise, if $t > p_i$ and $R(t + p_q + p_2^{D_i}) - (t + p_q + p_i + p_2^{D_i}) \geq 0$, by the interruption rule and the value of R , we have

$$(1-R)t + (2-R)p_q + p_i \leq (2-2R)t + (3-2R)p_q + p_i < (6-4R)p_i < 0.18p_i,$$

and if $R(t + p_q + p_2^{D_i}) - (t + p_q + p_i + p_2^{D_i}) < 0$, we have $(2-2R)t + (3-2R)p_q + (3-R)p_i < (5-3R)p_i \leq (R-0.82)p_i$. As a result, there exists an assignment of credits such that all jobs have nonnegative credit. $\square$

Lemma 3.5. *If OPT completes at least two jobs from D_i before J_q , then immediately after an interruption of J_i at time t , the total credit is nonnegative.*

Proof. Let T denote the set containing two jobs that OPT completes from D_i before J_q . Without loss of generality we can assume that two jobs in T are J_i and another job $J_2^{D_i}$ that is completed immediately after it. Again, if OPT

completes a job at least as large as J_i first, then all the bounds below increase or stay the same. The jobs in S_i delay the jobs in $D_i \setminus T$ by the same amount in the schedule of ALG and OPT. Therefore, these additional delays decrease the relative completion times of these jobs. No matter what order OPT uses for S_i and the larger jobs, under the condition that it completed at least two jobs from D_i before J_q , OPT completes the ℓ th job in S_i no earlier than at time $2p_i + \sum_{j=1}^{\ell} p_j^{S_i}$, whereas ALG completes the ℓ th job in S_i at time $t + \sum_{j=1}^{\ell} p_j^{S_i}$. This means that its credit is at least $2Rp_i - t + (R-1)\sum_{j=1}^{\ell} p_j^{S_i}$. Therefore, the credit of the ℓ th job in S_i is enough for the additional delay caused by the ℓ th job which is at most $p_{\ell}^{S_i}$. As a result, we ignore the sizes of the jobs in S_i which only improve the relative completion times of jobs in $D_i \setminus T$. Hence, every job in $D_i \setminus T$ is only additionally delayed by $t - s_i$ (relative to the difference between OPT and the schedule of ALG for the instance without interruption).

Suppose that OPT starts J_q at time t^* . Given that $t^* \geq 2p_i$, $t < 2p_i$ and $p_j^{D_i} \geq p_i$ for every j , we have

$$\frac{ALG(J_5^{D_i})}{OPT(J_5^{D_i})} = \frac{t + \sum_{j=1}^5 p_j^{D_i}}{t^* + \sum_{j=3}^5 p_j^{D_i}} < \frac{2p_i + \sum_{j=1}^5 p_j^{D_i}}{2p_i + \sum_{j=3}^5 p_j^{D_i}} \leq \frac{2+5}{2+3} < R.$$

Therefore, after the second job ($J_4^{D_i}$) in $D_i \setminus T$, subsequent jobs in $D_i \setminus T$ will have a relative completion time smaller than R . This is because ALG and OPT will process these jobs sequentially from smallest to largest, starting after $J_4^{D_i}$.

Consequently, we define the worst-case instance as follows: S_i contains only the job J_q that caused the interruption. D_i consists of the current interrupted job J_i and other jobs (at least as large as J_i) that are completed by ALG after J_i , under the condition that these jobs have relative completion times greater than R , delayed the processing of other jobs before time s_i , or satisfy both properties.

Now, consider the case that the number of jobs in $D_i \setminus \{T\}$ that delayed processing of at least one other job (before time s_i) is 0. We show that enough credits for $J_1^{D_i}, J_2^{D_i}, J_3^{D_i}$ and $J_4^{D_i}$ are available. OPT competes J_q at the earliest at time $2p_i + p_q$. ALG completes this job at the latest at time $t + p_q$, meaning that $(R-1)p_q - t + 2Rp_i$ credit remains, which can be allocated to $J_1^{D_i}, J_2^{D_i}, J_3^{D_i}$ and $J_4^{D_i}$, then we have the following inequality which is true because of our interruption rule, $(R-1)p_q - t + 2Rp_i \geq 2(t + p_q - s_i) + (t - s_i) + \min(t + (1-R)(p_q + \sum_{j=1}^4 p_j^{D_i}), 0)$.

In the case that the number of jobs in $D_i \setminus \{T\}$ that delayed processing of

at least one other job (before time s_i) is $k \neq 0$, the condition $s_i^* \geq \frac{R^k-1}{(R-1)R^k} p_i$ holds because the algorithm interrupts with every incoming job before time $(2-R)(s_j^{D_i} + p_j^{D_i})$ that can potentially be completed before the currently running job $J_j^{D_i}$. Therefore, given $t > s_i^* + p_i$, we have $k((1-R)p_q + t - s_i) \leq (2R-4)p_q - 4t + R(s_i^* + 2p_i) + 3s_i$, as in the proof of Lemma 3.4. Consequently, an appropriate credit assignment exists where all jobs maintain nonnegative credit. $\square$

Theorem 3.2. *The IRSPT algorithm is R -competitive for the scheduling problem $1|r_j, \text{online}, \text{restart}| \sum C_j$.*

Proof. Suppose there is no restart in the schedule. By Property 3.5, this implies that all jobs arrive at a time when the machine is idle in the smallest counterexample (all at the same time). However, in this case the competitive ratio of our algorithm is actually 1, as it processes the jobs in their optimal sequence. It follows that an instance which is a smallest counterexample contains at least one restart. However, as shown in lemmas 3.3, 3.4 and 3.5, all jobs maintain nonnegative credit after the last interruption. This contradicts the assumption that the given instance is a counterexample and implies the competitive ratio of at most R . Hence, the IRSPT algorithm is R -competitive for $1|r_j, \text{online}, \text{restart}| \sum C_j$. $\square$

CHAPTER 4

Minimizing the Total Weighted Completion Time

4.1 Introduction

In this chapter, we study the problem of minimizing the total weighted completion time on a single machine with job restarts, which generalizes the classical total completion time minimization problem. In [129], Jiping Tao et al. presented the first semi-online algorithm for the problem $1|p_j = p, r_j| \sum w_j C_j$, achieving the best known competitive ratio of $(1 + \sqrt{5})/2 \approx 1.618$. We consider the power of *limited restarts* in minimizing the total weighted completion time. In particular, we present a deterministic online algorithm with job restarts that achieves a competitive ratio better than 1.325 with only at most three restarts for the entire schedule, in the case where all jobs have equal processing times. Additionally, we establish a new lower bound on the competitive ratio for deterministic online algorithms with restarts.

A natural generalization of total completion time is the *total weighted completion time*, where weights model priorities among jobs. This objective is more challenging because jobs with small processing times and large weights can dominate the objective, requiring careful scheduling decisions. The problem with release dates is strongly NP-hard [116] and there is a PTAS for the problem [3]. In the offline single-machine setting without release dates, the Weighted Shortest Processing Time (WSPT) rule, which schedules jobs in non-increasing order of the Smith ratio w_j/p_j , is known to be optimal [146] and achieves the approximation factor $(1 + \sqrt{2})/2$ in a parallel identical machine environment [109].

In this chapter, we assume that jobs arrive over time, i.e., each job becomes available for processing only at its release time. A sequence of jobs must be scheduled on a single machine. Each job j is characterized by a processing time

p_j and a non-negative arrival time r_j , where $0 \leq j \leq n$. Each job j is assigned a positive weight w_j , reflecting its relative importance. The goal is to minimize the *total weighted completion time*, defined as the sum of the weighted completion times of all jobs in the schedule. The classical total completion time problem is a special case in which all weights are equal. In the standard three-field notation, this problem is denoted as $1 \mid r_j, \text{online} \mid \sum w_j C_j$.

The *limited restarts* setting, where each job may be restarted at most once, and the more general k -limited restart where each job can be restarted at most k times and k can be a positive integer, allows the analysis of certain scheduling objectives to become tractable. This approach has been studied particularly for the weighted makespan problem in the context of parallel batch machines [122, 74, 151]. However, when jobs have arbitrary processing times, the analysis becomes significantly more challenging, even when limited or k -limited restarts are allowed.

4.1.1 Our Results

We present the first positive results that show how to use restarts effectively for this objective function by focusing on the case where all jobs have identical processing times. To the best of our knowledge, the only known result is the algorithm β -delayed WSPT (β D-WSPT), which achieves a competitive ratio of $(1 + \sqrt{5})/2 \approx 1.618$ [149] and does not use restarts. We establish a lower bound of 1.09725 on the competitive ratio achievable by any deterministic online algorithm. The bound was obtained by using a computer to identify difficult inputs. After approximating a worst-case input, we identified the critical cases. This resulted in a system of equalities that we solved to get an exact lower bound.

Our main result in this chapter is a deterministic online algorithm that achieves a competitive ratio better than 1.325 based on carefully structured decision intervals. The algorithm uses at most three restarts for the entire schedule and carefully redefines these decision intervals in certain situations to avoid further interruptions. The approach significantly simplifies the analysis by reducing the number of possible scheduling cases.

At the beginning of the input and whenever feasible, the algorithm will idle the machine as long as possible without violating the competitive ratio. This simplifies the analysis by having less and later intervals in which decisions on interruptions need to be made. Each job is only possibly interrupted only during

the initial portion of its processing. Since we focus on the total completion time, the decisions on whether to interrupt a running job tend to become easier as the input progresses, and after some time no further interruptions are required at all. However, at the start we need to be very careful. We will only interrupt a running job for another job that is significantly heavier; the weight threshold will depend on the current interval.

The competitive ratio of our algorithm is optimal for this number of interruptions, as we also establish a corresponding lower bound in the k -limited restart setting.

Notation We introduce the notation $j' \succ j$ to mean that job j' interrupts job j , meaning that j is running and then j' arrives and is started instead (possibly not immediately, but j is definitely interrupted when j' arrives). This notation is of course only used in relation to online algorithms, as an offline algorithm does not use restarts (it would simply not start a job instead of interrupting it later).

4.2 Lower Bound

We now consider the problem $1|r_j, \text{online}, p_j = 1, \text{restart}|\sum w_j C_j$. The lower bound presented here shows that the problem maintains its difficulty, even when unlimited restarts are allowed.

Theorem 4.1. *Any algorithm for the problem $1|r_j, \text{online}, p_j = 1, \text{restart}|\sum w_j C_j$ is at least $R_\infty \approx 1.09725$ -competitive.*

Proof. We consider the following job instance I . At time $r_0 = 0$, job 0 with $w_0 = 1$ arrives. Suppose that algorithm A starts to process it at time 0 as waiting would not improve the result. A second job arrives at time $r_1 = 0.42$ with $w_1 = \frac{(r_1+2)R_\infty-1}{2-(r_1+1)R_\infty} \approx 3.74598$.

Case 1. If $1 \not\succ 0$, no more jobs arrive and we have

$$\frac{A(I)}{\text{OPT}(I)} = \frac{1 + 2w_1}{(r_1 + 1)w_1 + (r_1 + 2)} = R_\infty.$$

Case 2. If $1 \succ 0$, then job 1 is started. Now, one more job arrives at time 1: job 2 with $w_2 = \frac{5.84R_\infty^2 + R_\infty r_1 - 5.42}{2.84R_\infty^2 - 7.4364R_\infty + 4.84} \approx 20.7935$. If $2 \not\succ 1$, then we have

$$\frac{A(I)}{\text{OPT}(I)} = \frac{(r_1 + 1)w_1 + (r_1 + 2)w_2 + r_2 + 3}{1 + 2w_2 + 3w_1} = R_\infty.$$

However, if $2 \succ 1$, then 7 more jobs arrive at time 2 with weight 1. The final ratio becomes

$$\frac{A(I)}{\text{OPT}(I)} = \frac{2w_2 + 3w_1 + 4 + \sum_{j=3}^9 (j+1)}{1 + 2w_2 + 3w_1 + \sum_{j=3}^9 j} \geq R_\infty,$$

thus $347900R_\infty^3 - 1287427R_\infty^2 + 1614187R_\infty - 680750 = 0$, and we have $R_\infty \approx 1.09725$. $\square$

4.3 Algorithm LW-3R

We consider the problem $1 \mid r_j, \text{online}, p_j = 1, \text{restart} \mid \sum w_j C_j$, where each job j has release time r_j , weight $w_j > 0$, and unit processing time. The objective is to minimize the total weighted completion time. Jobs may be interrupted but in that case they must restart from scratch.

The algorithm divides time into geometrically increasing intervals based on the arrival of the first job. These intervals are used to give structure to the online schedule and to limit the possible number of interruptions, which significantly helps in the analysis due to the reduced number of possible scheduling cases. In addition, we redefine these thresholds in certain situations, which avoids further interruptions. Let $R \approx 1.32472$ be the plastic ratio, which is the real root of the equation $R^3 = R + 1$. We have $R^5 = R^4 + 1$. At time r_0 , initial thresholds are defined as:

$$t_i = r_0 + R^i - 1 \text{ for } i = 0, \dots, 4 \text{ and } t_5 = \infty.$$

The corresponding thresholds define the initial intervals $I_i = [t_i, t_{i+1})$ for $i = 0, \dots, 4$ and have approximate values $t_1 \approx r_0 + 0.325$, $t_2 \approx r_0 + 0.755$, $t_3 = r_0 + R$ and $t_4 \approx r_0 + 2.080$. The machine is idle during interval I_0 . The algorithm starts its schedule with the heaviest available job at time t_1 and before t_4 , always starts the heaviest available job at the beginning of a new interval whenever it interrupted a job. If there is a choice, it takes the job with the smallest release date.

Interruptions: A job started at time t_i is interrupted only during interval I_i . This happens if during interval I_i with $0 < i < 4$, a new job arrives with weight strictly greater than a factor F_i times the weight of the currently running job, where F_i is defined as follows.

i	1	2	3
F_i	$R^2 \approx 1.755$	$R^2 + 2R \approx 4.404$	$(R^2 + 4R)(R^i - t_i)^{5/3} \approx 7.054(R^3 - t_3)^{5/3}$

If the running job is interrupted, the machine remains idle for the rest of the interval and starts the heaviest available job at the beginning of next interval. After an interruption in interval I_2 , the beginning of the next interval is shifted as follows. Let m be the interrupted job and let q be the heaviest job that satisfies the interruption rule. We set

$$t_3 \leftarrow \frac{w_m}{2R^3w_q} + R + r_0.$$

After time t_4 , no jobs are interrupted. A job started at time t_4 or later continues until it completes, even if heavier jobs arrive.

Completions: When a job that starts anytime after time t_2 is completed, then the next job starts immediately. Consider the case where a job m that started at time t_1 or t_2 is not interrupted, but completes successfully. If during processing of job m , a job with weight greater than $R^2 + 2R \approx 4.404$ times the weight of job m arrives, we denote the heaviest such a job by q . If m started at time t_2 , then if no such job q arrives, the next job starts immediately (in this case the value of t_3 is changed to $r_0 + R^2$), else it starts at time t_4 .

Suppose m started at time t_1 and is not interrupted. Then it completes at time $r_0 + R$. Note that q does not arrive in I_1 , because then m would have been interrupted. We increase t_3 as follows.

no job q arrives in I_2	$r_q \in I_2$
$r_0 + 2R - 1 \approx r_0 + 1.649$	$\frac{w_m}{2R^3w_q} + R + r_0 \approx 0.215 \frac{w_m}{w_q} + 1.325$

That is, if the job started at time t_1 is not interrupted, we will always idle the machine until the new value of t_3 is reached.

Observation 4.1. *LW-3R does not start any job at time $r_0 + R$ (the original value of t_3). Until after time t_4 , each new job is started at some time t_i (but the value of t_3 may be changed once).*

In Fig. 4.1, we illustrate an example of the algorithm's behavior. The figure contains several schedules, depicting the algorithm's complete schedule for the

jobs available at the time t . Suppose job 0 with weight 1 arrives at time 0 and job 1 with weight 0.5 arrives at time 0.2. In the schedule shown in (1) at time t_1 , the algorithm starts the heaviest available job. The starting time of job 1 is shown as a dashed rectangle to indicate it may change. In (2) at time $t = R$, no other job has arrived during the processing of job 0, t_3 is updated and job 1 starts at time $2R - 1$.

In an alternative scenario, if job 2 with weight 2 arrives in I_1 , it interrupts job 0 and starts at the beginning of the next interval. In the schedule shown in (3) at time $t = t_2$, the algorithm starts job 2 while the starting time of job 0 is again shown as a dashed rectangle, since it may change. Finally, in (4) at the completion time of job 2, since no job of weight greater than $2(R^2 + 2R)$ arrives during this job, the algorithm updates t_3 and starts job 0.

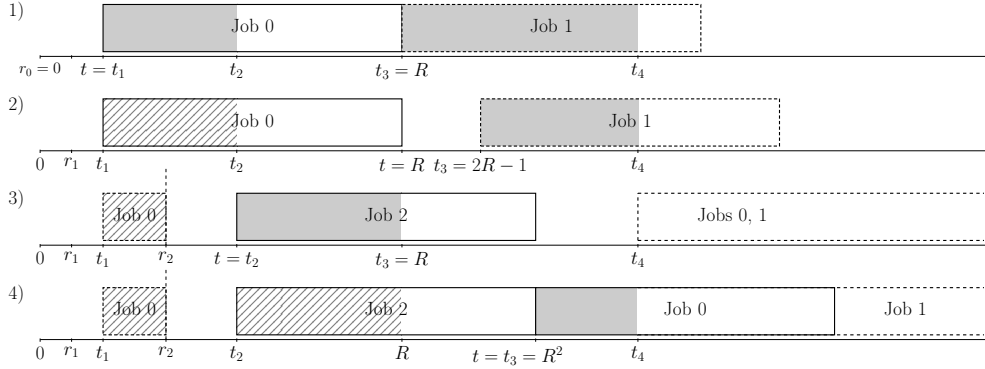


Figure 4.1: An example of an input for the LW-3R algorithm.

4.4 Competitive analysis of LW-3R

We now analyze the performance of LW-3R. In order to prove that LW-3R has a competitive ratio R (that is, the positive real solution to the equation $R^3 = R + 1$), we assume there exists a minimal counterexample consisting of a minimum number of jobs and we show by contradiction that such a counterexample does not exist. The method aims to identify a minimal counterexample with a specific structure within the space of all hypothetical minimal counterexamples. It begins with an arbitrary minimal counterexample and iteratively modifies it toward a structure that is likely to be the simplest for our algorithm. Each modification ensures that the performance ratio does not decrease. Finally, this process produces a relatively simple instance with a well-defined structure, whose performance ratio can be analyzed by checking a finite number of cases. This

allows us to show that no such minimal counterexample exists, implying that there is no counterexample to the proposed competitive ratio of our algorithm.

Consider a hypothetical minimal counterexample Γ . A **block** is defined as a period of job processing during which the machine is never idle due to the absence of available jobs. Any idle time in the block occurs only when at least one job is already available but intentionally delayed.

Lemma 4.1. *The schedule σ generated by LW-3R for the minimal counterexample Γ consists of a single block that starts with a non-negative period of idle time due to the absence of available jobs.*

Proof. Let B be a block that implies a competitive ratio no less than that of Γ , since Γ is a minimal counterexample. Idle time after a block means no jobs are waiting. The existence of the jobs that are completed before or after the period of idle time does not influence the starting time of jobs in the block B or the order that the jobs are processed. Therefore block B by itself forms a counterexample. $\square$

Lemma 4.2. *Let $I(w)$ be a family of problem instances in which one or more jobs have weight $w \geq 0$ and all other jobs have fixed weights. The release times of all jobs are fixed. Let $\text{OPT}(w)$ be the optimal total completion time of $I(w)$. Then $\text{OPT}(w)$ is a concave function of w .*

Proof. The optimal schedule for any given input is completely determined by a permutation of the jobs. Given the permutation, the schedule can be determined by scheduling each job as early as possible, but after the previous job in the permutation completes and not earlier than its release date. The cost of the schedule corresponding to any fixed permutation is an affine function of w : the cost of all jobs with fixed weight is a constant, and the cost of the jobs with weight w is linear in w . Therefore, $\text{OPT}(w)$ is the minimum of a (finite) set of affine functions, and therefore concave. $\square$

We consider a job j that starts at time s to be running in the half-open interval $(s, s + 1]$ (or until it gets interrupted). If j is not interrupted, a job that arrives at time $s + 1$ is considered to arrive during j . The *relative completion time* of a job j in the schedule σ generated by the algorithm, relative to a feasible schedule ψ , is defined as the completion time of job j in σ divided by its completion time in ψ .

For simplicity of presentation, we introduce the notation t^+ for the moment just after the algorithm made its decision about what to schedule at time t . A job that arrives at time t^+ thus has no effect on scheduling decisions at time t , and it arrives in an open interval starting at time t .

Observation 4.2. *The algorithm does not take the precise arrival time of any job into account, only the interval in which it arrives.*

Definition 4.1. *Let ℓ denote the index of the last interval during which at least one job arrives, and let f be the last arriving job, while p is the job that is running when f arrives (if it exists).*

Observation 4.3. *For any input, if the release date of some job can be decreased without changing the online schedule, the competitive ratio for this input does not decrease as this release date is decreased.*

Proof. The optimal cost of the input does not increase as a result of this change, whereas the online cost remains the same. $\square$

Observation 4.4. *Starting at the latest from time $t_{\ell+1}$, both OPT and LW-3R complete all their remaining jobs (which are not necessarily the same ones) in order of decreasing weight (without interruptions). If the algorithm completes no job by time $t_{\ell+1}$ it completes all jobs in this order.*

Lemma 4.3. *In a minimal counterexample Γ , we may assume that job p exists, and that job f is heavier than job p .*

Proof. Suppose p does not exist. Then f arrives while the machine is idle. If the machine is idle because another job was previously interrupted and the algorithm is waiting for time $t_{\ell+1}$, then we can decrease the arrival time of f to the time of the last interruption without changing the completion time of any job in the online schedule. Otherwise, starting from time $t_{\ell+1}$ (or from time r_f if $r_f \geq t_4$), the remaining jobs are run in the optimal order by Observation 4.4, and none of these jobs arrived before time $r_f \geq t_\ell$, so Γ is not a minimal counterexample as $(t_{\ell+1} + 1)/(t_\ell + 1) = R$ for $\ell \leq 3$.

Finally, if f is not heavier than p , then r_f can be decreased to r_p without changing the online schedule, and we apply Observation 4.3. $\square$

We next demonstrate how the instance Γ can be modified to achieve a simpler and more specific structure. We use a sequence of steps; each time the current

minimal counterexample will be denoted by Γ and the modified one will be denoted by Γ' .

Lemma 4.4. *If a counterexample Γ exists, another minimal counterexample Γ' can be constructed such that first job arrives at time 0.*

Proof. If the arrival time of the first job is $r_0 \neq 0$, we construct another instance Γ' in which all arrival times are reduced by r_0 . As the intervals are defined based on r_0 , this does not change the behavior of the algorithm. The cost of both the algorithm and the optimal cost both decrease by exactly $n \cdot r_0$, where n is the number of jobs. Since both values are reduced by the same additive term, the competitive ratio increases. $\square$

Lemma 4.5. *If a counterexample Γ exists, another minimal counterexample Γ' can be constructed such that no job arrives during any job that starts at time t_4 or later.*

Proof. If no job is executing during the interval I_4 , we are done. Let j be the last job during which other jobs arrive and $s \geq t_4$ be the time at which j started (for the last time). Let S be the set of jobs that arrive while j is processing. We will use Observation 4.3 extensively. For each job in S we reduce the release time to s^+ . Furthermore, for each job in S that is not heavier than j , we reduce the release time to s and remove the job from S . Both changes do not affect the behavior of algorithm. Therefore, we are left with the situation that all jobs in S are heavier than j and all arrive at the same time, which is s^+ .

Consider the weight w of the heaviest job h in S , and the cost of algorithm and the optimal cost as functions of the parameter w . The cost of algorithm is linearly increasing in w (as long as h remains the heaviest job), because after t_4 there are no gaps in its schedule. By Lemma 4.2, the optimal cost is a concave function of w . Therefore, the competitive ratio is maximized either for $w \rightarrow \infty$ or for w minimal, that is, equal to the second highest weight in S .

For $w \rightarrow \infty$, the competitive ratio tends to the relative completion time of h , as the contribution of all other jobs to the total optimal and online cost is constant once w is sufficiently large (where the permutation that determines the schedule does not change anymore). Since h is the heaviest waiting job, the algorithm starts it immediately after j completes. The relative completion time of h is less than

$$\frac{t_4 + 2}{t_4 + 1} = \frac{R^4 + 1}{R^4} = \frac{R^5}{R^4} = R$$

(since $r_0 = 0$ by Lemma 4.4), so this does not give a counterexample. Consequently, the competitive ratio is maximized by setting w equal to the second highest weight in S .

We can now consider instances in which the two heaviest jobs in S have weight w , leading to the same conclusion as above as all jobs in S have relative completion time less than R , and continue in this manner. We reach the conclusion that all jobs in S have the same weight. Moreover, the competitive ratio is maximized if this weight is minimal, that is, equal to the weight of j . We already discussed above that the release time of such jobs can be reduced to s . Hence, no jobs arrive during any job that starts at or after time t_4 . $\square$

Lemma 4.6. *If a counterexample Γ exists, another minimal counterexample Γ' can be constructed such that at most one job arrives at each time t_0, t_1^+, t_2^+ .*

Proof. By Observation 4.2, we only need to consider the intervals. We will decrease release dates using Observation 4.3. During the interval I_0 , the machine idles no matter when the jobs arrive, thus without loss of generality the arrival time of any job that arrives in $[0, t_1]$ is equal to 0. Moreover, since all these jobs now arrive at the same time 0, if OPT chooses to run any of them before t_1 , it may do so for the heaviest one without loss of generality. Now, we shift all jobs arriving at time in $[0, t_1]$ except for the heaviest one to arrive at t_1^+ . After this step, only one job arrives at time 0.

In the interval I_1 , a job is always running (initially). Decreasing the arrival time of jobs within interval $(t_1, t_2]$ that do not affect the earlier scheduling decision of algorithm may only cause the algorithm to interrupt the running job earlier (if the algorithm interrupts any running job), but it does not change the completion time of any job in the algorithm's schedule. Therefore, we shift the arrival times of all jobs in this interval to t_1^+ . We also do this if there is no interruption.

Finally, OPT starts at most one job during the interval $(t_1, t_1 + 1)$. Exactly as above, we shift all jobs arriving in $(t_1, t_2]$ except for the heaviest one to arrive at t_2^+ . This modification does not affect the optimal cost or the completion time of any job in the algorithm's schedule. The same reasoning applies to the interval I_2 , because $t_3 - t_2 \leq 1$ in all cases. $\square$

Lemma 4.7. *Let Γ be a counterexample in which some set of jobs S arrives at time t_ℓ^+ . Among the jobs in S , let h be the heaviest one that is completed first by the algorithm. Let j be a different job in S . Suppose the relative completion*

time of j is at most R , then if the weight of j can be decreased without changing the schedule of algorithm, the implied competitive ratio increases. Therefore, if j is the last job in the schedule of algorithm, then Γ is not minimal (because j can be removed). Moreover, if the algorithm does not interrupt a job at time t_ℓ^+ , these statements also hold for h .

Proof. Job j has no effect on the decisions of algorithm at time t_ℓ^+ , since the algorithm ignores j at least until h completes. A decrease in the weight of a job does not increase the optimal cost. Therefore, if the relative completion time of j is at most R and Γ is a counterexample, decreasing the weight of j without changing the schedule of algorithm increases $\text{ALG}(\Gamma)/\text{OPT}(\Gamma)$. Moreover, if j is the last job in the schedule of algorithm, then removing j from the input entirely increases $\text{ALG}(\Gamma)/\text{OPT}(\Gamma)$ using Observation 4.4: the only change in the schedule of algorithm is that it ends one job earlier. $\square$

Corollary 4.1. *In the situation described in Lemma 4.7, the weight of j can be decreased as long as the relative completion time of j remains at most R in at least one schedule with the optimal cost. If in this process we reach a state in which j is the last job in the schedule of algorithm, then Γ is not minimal (because j can be removed).*

Proof. By Lemma 4.7 and Observation 4.4, we can decrease the weight of j until it has the same weight as the job following it in the schedule of algorithm, because this does not change the schedule of algorithm. At this point, we can exchange these two jobs in the schedule of algorithm without changing $\text{LW-3R}(\Gamma)$. We can then repeat this process until we reach a schedule in which the relative completion time of j is more than R in every permutation of the optimal schedule or job j can be removed from the input because the algorithm completes it last. $\square$

Lemma 4.8. *If a counterexample Γ exists, another minimal counterexample Γ' can be constructed such that no job arrives after time t_3^+ .*

Proof. If the algorithm does not start a job at time t_3 , then if a job arrives at time t_4 , its arrival time can be shifted to t_3^+ without changing the completion time of any job in algorithm's schedule or increasing the optimal cost. This holds because after LW-3R completes a job that it starts at time t_2 , it waits until time t_4 before starting the next job.

Otherwise, the algorithm starts running a job at time t_3 . Given Lemma 4.5 and because $t_4 - t_2 > 1$, the only case left is that a job j that starts at time t_3 is still running at time t_4 . The schedule of the algorithm consists of a single block and all jobs are processed contiguously. This follows from Lemma 4.1. Therefore, similar to the proof of Lemma 4.5, we can consider the set S of jobs that arrive in interval $[t_4, t_3 + 1]$. We assume that all jobs in S are heavier than the job that arrives at time t_3^+ (if such a job exists), since any job lighter than this can be shifted to time t_3^+ , without loss of generality. The relative completion time of the heaviest job in S is less than R . We again get that all of these jobs have the same weight and arrive at time t_4 . Consequently, the weight of these jobs is minimal in a counterexample (equal to the job that arrives at time t_3^+ , if such a job exists and is heavier than j), thus their release time can be further shifted to t_3^+ and no job arrives afterwards. $\square$

Consider a minimal counterexample Γ' for the algorithm described in Lemma 4.8. The next Lemma in particular covers the case where the input contains of only one job, and shows that we only need to consider cases where OPT differs from the algorithm on at most the first two jobs. In the following, we index the jobs according to their arriving time and starting from 0.

Lemma 4.9. *In any minimal counterexample, LW-3R and OPT do not complete the first $\ell - 1$ jobs in the same order.*

Proof. Suppose they do. By the structure of the algorithm, these jobs have relative completion time less than R and since no job arrives after the completion time of the $(\ell - 1)$ -st job, both OPT and the algorithm schedule the remaining jobs in the same order by decreasing weight (Observation 4.4) and the arrival times do not affect this ordering. Therefore, every job has relative completion time of at most R , and we have $\text{ALG}/\text{OPT} \leq R$. $\square$

Corollary 4.2. *For the minimal counterexample Γ , LW-3R schedules a job to start at time t_3 .*

Theorem 4.2. *For the scheduling problem $1|r_j, \text{online}, p_j = 1, 3\text{-restart}|\sum w_j C_j$, LW-3R is the best possible online algorithm with the competitive ratio of $R \approx 1.3247$, that is, the real solution to the equation $R^3 = R + 1$.*

Proof. Suppose, for contradiction, that LW-3R is not R -competitive. Then there exists a minimal counterexample Γ' from lemma 4.8 in which the ratio between $\text{LW-3R}(\Gamma)$ and the cost of the optimal schedule exceeds R .

By Lemma 4.9, we already covered the case where the input consists of a single job, thus we assume there are at least two jobs. If LW-3R does not start a job at time t_3 , then there are at least three jobs. In this situation, job J_2 has an arriving time strictly greater than R and $w_2 > (R^2 + 2R)w_1$. Consequently, OPT will always start the same two jobs as the algorithm and by Lemma 4.9 we obtain Corollary 4.2. Therefore, for such minimal counterexample, algorithm schedules a job to start at time t_3 , where the value of t_3 depends on what happens until time $r_1 + R$.

We denote our hypothetical counterexample after all the modifications above simply by Γ again. We use the notation introduced in Definition 4.1; note that the value of ℓ may have been changed in the modifications (see the proof of Lemma 4.6). By Lemma 4.6, if $\ell < 3$, then $|\Gamma| \leq 3$. We scale the weights so that the weight of the job that LW-3R completes first is 1.

Case 1: The final value of t_3 is $R^2 \approx 1.755$. The job completed at time R^2 started at time $R^2 - 1 = t_2 \approx 0.755$ although the first job arrived at time 0 and LW-3R starts a job at time $R - 1$. Therefore, LW-3R interrupts job 0 and does not interrupt job 1 but completes it at time R^2 . Moreover, no job of weight strictly greater than $R^2 + 2R$ arrives during its processing. Since the online schedule consists of a single block, either at least one job arrives while job 1 is running or the input ends.

Case 1.1: ($|\Gamma| = 2$) We have $w_1 = 1$ and $1 \succ 0$ in I_1 , thus $1/R^2 > w_0$. We have

$$\begin{aligned} \text{LW-3R}(\Gamma) &\leq t_2 + 1 + (t_2 + 2)w_0 \\ \text{OPT}(\Gamma) &\geq w_0 + 2. \end{aligned}$$

Since $R^2 + 1 > R$, we obtain the following bound:

$$\frac{\text{LW-3R}(\Gamma)}{\text{OPT}(\Gamma)} \leq \frac{R^2 + (R^2 + 1)w_0}{w_0 + 2} < \frac{R^2 + (R^2 + 1)/R^2}{1/R^2 + 2} \approx 1.29374 < R.$$

Case 1.2: ($\ell = 2, |\Gamma| \geq 3$) We have $|\Gamma| \leq 3$ by Lemma 4.6, so in this case $|\Gamma| = 3$ and $r_2 = t_2^+$. By Lemma 4.3, job 2 is heavier than job 1, so LW-3R uses the order 1,2,0 (since $1 \succ 0$). Then $\text{LW-3R}(\Gamma) = (t_2 + 1)w_1 + (t_3 + 1)w_2 + (t_3 + 2)w_0$.

We have that $2 \not\succ 1$ in I_2 , thus $w_2 \leq R^6 - 1$ and $1 \succ 0$, thus $1/R^2 > w_0$. Both job 1 and job 2 are available at time 1. Hence if OPT schedules job 1 first, it uses the order 1,2,0 which is the same as LW-3R. If OPT schedules job 0 first,

it uses the order 0,2,1. Since $R^2 + 2 > R$ and $(R^2 + 1)/2 > R$, we obtain the following bounds:

$$\begin{aligned} \frac{\text{LW-3R}(\Gamma)}{\text{OPT}(\Gamma)} &\leq \frac{R^2 + (R^2 + 1)w_2 + (R^2 + 2)w_0}{w_0 + 2w_2 + 3} \\ &< \frac{R^2 + (R^6 - 1)(R^2 + 1) + (R^2 + 2)/R^2}{1/(R^2) + 2(R^6 - 1) + 3} \approx 1.29482 < R. \end{aligned}$$

Otherwise, $\text{OPT}(\Gamma) \geq R^2w_2 + R^2 + 1 + (R^2 + 2)w_0$ (using the order 2, 1, 0), and we obtain the following bounds:

$$\frac{\text{LW-3R}(\Gamma)}{\text{OPT}(\Gamma)} \leq \frac{R^2 + (R^2 + 1)w_2 + (R^2 + 2)w_0}{R^2w_2 + R^2 + 1 + (R^2 + 2)w_0} < \frac{R^2 + (R^6 - 1)(R^2 + 1)}{R^2(R^6 - 1) + R^2 + 1} = R.$$

Case 1.3: ($\ell = 3, |\Gamma| = 3$) We consider two possible scenarios with respect to job f .

Case 1.3.1: ($f \succ p$) We have, $\text{LW-3R}(\Gamma) = (t_2 + 1)w_1 + (t_4 + 1)w_2 + (t_4 + 2)w_0$. The worst case to consider is that $\text{OPT}(\Gamma) \geq w_0 + 2w_1 + 3w_2$ (using the order 0, 1, 2), because job 2 is available at time R^2 and any other ordering results in a higher cost. We have that $2 \succ 0$ in I_3 , thus $w_2 > (R^2 + 4R)(R^3 - R^2)^{5/3}w_0 > 2.76w_0$ and $1 \succ 0$, thus $1/R^2 > w_0$. We obtain the following bounds, using $R^4/3 < R$ and $R^4 + 1 > R$:

$$\begin{aligned} \frac{\text{LW-3R}(\Gamma)}{\text{OPT}(\Gamma)} &\leq \frac{R^2 + (R^4)w_2 + (R^4 + 1)w_0}{w_0 + 2 + 3w_2} \\ &< \frac{R^2 + (2.76275R^4 + R^4 + 1)/R^2}{(1 + 3(2.76275))/R^2 + 2} \approx 1.2242 < R. \end{aligned}$$

Case 1.3.2: ($f \not\succ p$) We have

$$\begin{aligned} \text{LW-3R}(\Gamma) &\leq (t_2 + 1)w_1 + (t_2 + 2)w_0 + (t_2 + 3)w_2 \\ \text{OPT}(\Gamma) &\geq Rw_1 + (R^2 + 1)w_2 + (R^2 + 2)w_0 \end{aligned}$$

(using the order 1, 2, 0). We have that $2 \not\succ 0$, thus $w_2 \leq (R^2 + 4R)(R^3 - R^2)^{5/3}w_0$ and $1 \succ 0$, thus $1/R^2 > w_0$. We obtain the following bounds, using $(R^2 + 2)/(R^2 + 1) > R$:

$$\begin{aligned} \frac{\text{LW-3R}(\Gamma)}{\text{OPT}(\Gamma)} &\leq \frac{R^2 + (R^2 + 1)w_0 + (R^2 + 2)w_2}{R + (R^2 + 1)w_2 + (R^2 + 2)w_0} \\ &< \frac{R^2 + (2.76275(R^2 + 2) + (R^2 + 1))w_0}{(R^2 + 2 + 2.76275(R^2 + 1))w_0 + R} < R, \end{aligned}$$

where, $(R^2 + 1)/(R^2 + 2) < R$ and $w_0 > 0$.

Case 1.4: ($\ell = 3, |\Gamma| \geq 4$) Suppose first that $|\Gamma| = 4$. We consider two possible scenarios with respect to job f .

Case 1.4.1: ($f \succ p$) Then

$$\text{LW-3R}(\Gamma) = (t_2 + 1)w_1 + (t_4 + 1)w_3 + (t_4 + 2)w_2 + (t_4 + 3)w_0.$$

The worst case to consider is that job 2 arrives at time t_2^+ , and $2 \not\succ 1$ in I_2 , thus $w_2 \leq R^2 + 2R$. The optimal cost is at least $R^2w_2 + (R^2 + 1)w_3 + (R^2 + 2)w_1 + (R^2 + 3)w_0$ (using the order $2, 3, 1, 0$). We have that $3 \succ 2$ in I_3 , thus $w_3 > (R^2 + 4R)(R^3 - R^2)^{5/3}w_2 > 2.76w_2$ and $1 \succ 0$, thus $1/R^2 > w_0$. Since $(R^4 + 2)/(R^2 + 3) < R$ and $R^4/(R^2 + 1) < R$, we obtain the following bounds:

$$\begin{aligned} \frac{\text{LW-3R}(\Gamma)}{\text{OPT}(\Gamma)} &\leq \frac{R^2 + (R^4)w_3 + (R^4 + 1)w_2 + (R^4 + 2)w_0}{R^2w_2 + (R^2 + 1)w_3 + R^2 + 2 + (R^2 + 3)w_0} \\ &< \frac{R^2 + (2.76275R^4 + (R^4 + 1))(R^2 + 2R)}{(R^2 + 2.76275(R^2 + 1))(R^2 + 2R) + R^2 + 2} \approx 1.27086 < R. \end{aligned}$$

Case 1.4.2: ($f \not\succ p$) Then

$$\begin{aligned} \text{LW-3R}(\Gamma) &\leq (t_2 + 1)w_1 + (t_3 + 1)w_2 + (t_3 + 2)w_3 + (t_3 + 3)w_0 \\ \text{OPT}(\Gamma) &\geq R^2w_2 + (R^2 + 1)w_3 + (R^2 + 2)w_1 + (R^2 + 3)w_0 \end{aligned}$$

(using the order $2, 3, 1, 0$), because job 3 is available at time R^2 . We have that $2 \not\succ 1$ in I_2 , thus $w_2 \leq R^2 + 2R$. We have $3 \not\succ 2$ and $1 \succ 0$, thus $w_3 \leq 2.76275w_2$ and $1/R^2 > w_0$, respectively. We obtain the following bound, using $(R^2 + 2)/(R^2 + 1) > R$ and $(R^2 + 1)/R^2 > R$:

$$\begin{aligned} \frac{\text{LW-3R}(\Gamma)}{\text{OPT}(\Gamma)} &\leq \frac{R^2 + (R^2 + 1)w_2 + (R^2 + 2)w_3 + (R^2 + 3)w_0}{R^2w_2 + (R^2 + 1)w_3 + R^2 + 2 + (R^2 + 3)w_0} \\ &< \frac{R^2 + ((R^2 + 1) + 2.76275(R^2 + 2))(R^2 + 2R)}{(R^2 + 2.76275(R^2 + 1))(R^2 + 2R) + R^2 + 2} \approx 1.32379 < R. \end{aligned}$$

Finally, by Corollary 4.1 and Observation 4.4, any additional jobs that arrive at time t_3^+ can only improve the bounds in Case 1.4. Therefore, the competitive ratio is at most R .

Case 2: The final value of t_3 is $2R - 1 \approx 1.649$. In this case, LW-3R does not interrupt job 0 and no job arrives that is a factor of $R^2 + 2R$ heavier than job 0 in $(t_1, t_1 + 1)$.

Case 2.1: ($|\Gamma| = 2$) We have $r_1 \in I_1$ or $r_1 \in I_2$, else we would have two separate blocks (Lemma 4.1). If $r_1 \in I_1$, then $w_1 \leq R^2$. By Lemma 4.9, OPT completes job 1 first, we have

$$\frac{\text{LW-3R}(\Gamma)}{\text{OPT}(\Gamma)} \leq \frac{R + 2Rw_1}{Rw_1 + R + 1} \leq \frac{R + 2R(R^2)}{R(R^2) + R + 1} \approx 1.28492 < R.$$

If $r_1 \in I_2$, then $w_1 \leq R^2 + 2R$. By Lemma 4.9, OPT completes job 1 first, then using $2R/R^2 > R$ we have

$$\frac{\text{LW-3R}(\Gamma)}{\text{OPT}(\Gamma)} \leq \frac{R + 2Rw_1}{R^2w_1 + R^2 + 1} \leq \frac{R + 2R(R^2 + 2R)}{R^2(R^2 + 2R) + R^2 + 1} \approx 1.26277 < R.$$

Case 2.2: ($\ell = 2, |\Gamma| > 2$) By Lemma 4.6, $|\Gamma| = 3$ in this case, and exactly one job arrives at times $0, t_1^+, t_2^+$. It can be checked that the ratio does not get worse than in Case 2.1: either job 2 has a low relative completion time, or OPT now completes job 0 even later.

Case 2.3: ($\ell = 3$) Suppose that $|\Gamma| = 3$. We consider two possible scenarios with respect to job 1.

Case 2.3.1: ($r_1 \in I_1$) If job f does not interrupt the running job.

$$\text{LW-3R}(\Gamma) = (t_1 + 1)w_0 + (t_3 + 1)w_1 + (t_3 + 2)w_2.$$

If OPT completes job 1 first, then $\text{OPT}(\Gamma) \geq (t_1 + 1)w_1 + (t_3 + 1)w_2 + (t_3 + 2)w_0$ (using the order $1, 2, 0$). We have that $1 \neq 0$ in interval I_1 , thus $1 \leq w_1 \leq R^2$ and $2 \neq 1$ in interval I_3 , thus $w_2 \leq (R^2 + 4R)(R^3 - 2R + 1)^{5/3}w_1 > 3.66w_1$. Using $(2R + 1)/(2R) > R$, we obtain the following bound:

$$\begin{aligned} \frac{\text{LW-3R}(\Gamma)}{\text{OPT}(\Gamma)} &\leq \frac{R + (2R)w_1 + (2R + 1)w_2}{Rw_1 + (2R)w_2 + 2R + 1} \\ &\leq \frac{R + (2R)(R^2) + (2R + 1)3.66R^2}{R(R^2) + 3.66R^2(2R) + 2R + 1} \approx 1.27 < R. \end{aligned}$$

If OPT completes job 1 later, we obtain the following bound:

$$\frac{\text{LW-3R}(\Gamma)}{\text{OPT}(\Gamma)} \leq \frac{R + (2R)w_1 + (2R + 1)w_2}{1 + (2R)w_2 + (2R + 1)w_1} < \frac{R + (2R + 3.661(2R + 1))w_1}{1 + (3.66(2R) + 2R + 1)w_1} < R,$$

where $w_1 > 1$. Suppose that the last arriving job f interrupts the running job, then

$$\begin{aligned}\text{LW-3R}(\Gamma) &\leq (t_1 + 1)w_0 + (t_4 + 1)w_2 + (t_4 + 2)w_1 \\ \text{OPT}(\Gamma) &\geq Rw_1 + 2Rw_2 + (2R + 1)w_0\end{aligned}$$

(using the order $1, 2, 0$). We have that $2 \succ 1 \not\succeq 0$ thus $w_2 > 3.66631w_1$ and $R^2 \geq w_1$. Using $(R^4 + 1)/R > R$ and $R^4/(2R) < R$, we obtain the following bound:

$$\begin{aligned}\frac{\text{LW-3R}(\Gamma)}{\text{OPT}(\Gamma)} &\leq \frac{R + R^4w_2 + (R^4 + 1)w_1}{Rw_1 + 2Rw_2 + 2R + 1} \\ &< \frac{R + (3.66631R^4 + (R^4 + 1))R^2}{(R + 3.66631(2R))R^2 + (2R + 1)} \approx 1.229 < R.\end{aligned}$$

Case 2.3.2: ($r_1 \in I_2$) If job f does not interrupt the running job.

$$\text{LW-3R}(\Gamma) = (t_1 + 1)w_0 + (t_3 + 1)w_1 + (t_3 + 2)w_2.$$

OPT completes job 1 first, then $\text{OPT}(\Gamma) \geq (t_2 + 1)w_1 + (t_2 + 2)w_2 + (t_2 + 3)w_0$ (using the order $1, 2, 0$). We have that $w_1 \leq R^2 + 2R$ and $2 \not\succeq 1$ in interval I_3 , thus $w_2 \leq (R^2 + 4R)(R^3 - 2R + 1)^{5/3}w_1 \approx 3.66631w_1$. Using $2R/R^2 > R$ and $(2R + 1)/(R^2 + 1) = R$, we obtain the following bound:

$$\begin{aligned}\frac{\text{LW-3R}(\Gamma)}{\text{OPT}(\Gamma)} &\leq \frac{R + 2Rw_1 + (2R + 1)w_2}{R^2w_1 + (R^2 + 1)w_2 + R^2 + 2} \\ &\leq \frac{R + 2R(R^2 + 2R) + 3.66631(2R + 1)(R^2 + 2R)}{R^2(R^2 + 2R) + 3.66631(R^2 + 1)(R^2 + 2R) + R^2 + 2} \\ &\approx 1.28507 < R.\end{aligned}$$

Suppose that $f \succ p$, then

$$\text{LW-3R}(\Gamma) \leq (t_1 + 1)w_0 + (t_4 + 1)w_2 + (t_4 + 2)w_1.$$

If $\text{OPT}(\Gamma) \geq R^2w_1 + (R^2 + 1)w_2 + (R^2 + 2)w_0$ (using the order $1, 2, 0$). We have that $2 \succ 1$, thus $w_2 > 3.66631w_1$ and $w_1 \leq R^2 + 2R$. Since $R^4/(R^2 + 1) < R$,

we obtain the following bound:

$$\begin{aligned} \frac{\text{LW-3R}(\Gamma)}{\text{OPT}(\Gamma)} &\leq \frac{R + R^4 w_2 + (R^4 + 1)w_1}{R^2 w_1 + (R^2 + 1)w_2 + R^2 + 2} \\ &< \frac{R + (3.66631R^4 + R^4 + 1)w_1}{(R^2 + 3.66631(R^2 + 1))w_1 + 2R + 1} < R, \end{aligned}$$

where, $w_1 > 1$. Otherwise, $\text{OPT}(\Gamma) \geq w_0 + 2w_1 + 3w_2$ (using the order $0, 1, 2$), we obtain the following bound using $R^4/3 < R$:

$$\frac{\text{LW-3R}(\Gamma)}{\text{OPT}(\Gamma)} \leq \frac{R + R^4 w_2 + (R^4 + 1)w_1}{1 + 2w_1 + 3w_2} < \frac{R + (3.66631R^4 + R^4 + 1)w_1}{1 + (2 + 3 \cdot 3.66631)w_1} < R,$$

where, $w_1 > 1$. Hence, the input consist of at least four jobs. Since, no job arrives with a factor of $R^2 + 2R$ heavier than the currently running job in $(t_1, t_1 + 1)$, by Corollary 4.1 and Observation 4.4, the additional jobs that arrive at time t_3^+ can only improve the bounds in this case. Therefore, the competitive ratio is at most R .

Case 3: The final value of t_3 is $R + w_m/(2R^3 w_q) < 1.38$. In this case, a job with weight strictly greater than $R^2 + 2R$ times the weight of the running job arrives before time R . This job either interrupts job 1 or it arrives in I_2 without causing an interruption to job 0. We have that $R + w_m/(2R^3 w_q) < R + 0.049$, since in this case the ratio $w_q/w_m \geq R^2 + 2R$.

Case 3.1: (Job 0 is completed at time R) Since LW-3R starts a job at $R + \frac{1}{2R^3 w_1}$, job 1 arrives in $(t_1, t_1 + 1)$ that is a factor of $R^2 + 2R$ heavier than job 0. Suppose $|\Gamma| = 2$.

$$\begin{aligned} \text{LW-3R}(\Gamma) &\leq (t_1 + 1)w_0 + (t_3 + 1)w_1 \\ \text{OPT}(\Gamma) &\geq R^2 w_1 + (R^2 + 1)w_0. \end{aligned}$$

Since $w_1 \geq R^2 + 2R$, we obtain the following bound:

$$\frac{\text{LW-3R}(\Gamma)}{\text{OPT}(\Gamma)} \leq \frac{R + \frac{1}{2R^3} + R^3 w_1}{R^2 w_1 + R^2 + 1} < \frac{R + (R^2 + 2R)R^3 + \frac{1}{2R^3}}{R^2(R^2 + 2R) + R^2 + 1} \approx 1.12349 < R.$$

If $|\Gamma| \geq 3$, the additional jobs that arrive can only improve the bounds in this case.

Case 3.2: (Job 1 is interrupted) We consider two possible scenarios with respect to this job. In the worst-case instance, the schedule contains the largest

possible number of jobs with relative completion time greater than R .

Case 3.2.1: (Job 2 is not interrupted) If $\ell = 2$, then

$$\text{LW-3R}(\Gamma) \leq (t_3 + 1)w_3 + (t_3 + 2)w_1 + (t_3 + 3)w_0 + (t_3 + 4)w_2.$$

We have that $3 \succ 1 \succ 0$ thus $1 > (R^2 + 2R)w_1$, $w_1 > R^2w_0$ and $w_2 \leq w_0$. We only need to consider the situation that OPT completed job 0, thus $\text{OPT}(\Gamma) \geq w_0 + 2w_3 + 3w_1 + 4w_2$ (using the order 0, 3, 1, 2), because the last arriving job in the minimal counterexample is heavier than the job that arrived just before it. Since $\frac{w_1}{2R^3} < 0.049$, we obtain the following bound using $(R^3 + 3)/4 > R$ and $R^3 + 2 > R$:

$$\begin{aligned} \frac{\text{LW-3R}(\Gamma)}{\text{OPT}(\Gamma)} &\leq \frac{R^3 + \frac{w_1}{2R^3} + (R^3 + 1 + \frac{w_1}{2R^3})w_1 + (R^3 + 2 + \frac{w_1}{2R^3})w_0 + (R^3 + 3 + \frac{w_1}{2R^3})w_2}{w_0 + 2 + 3w_1 + 4w_2} \\ &< \frac{R^3 + 0.049 + (R^3 + 1.049)w_1 + (R^3 + 2.049)w_0 + (R^3 + 3.049)w_2}{w_0 + 2 + 3w_1 + 4w_2} \\ &< \frac{R^3 + 0.049 + ((R^3 + 1.049) + (R^3 + 2.049)/R^2 + (R^3 + 3.049)/R^2)/(R^2 + 2R)}{2 + (3 + 5/R^2)/(R^2 + 2R)} \\ &\approx 1.32235 < R. \end{aligned}$$

The additional jobs that can arrive at time t_2^+ have relative completion time of at most R and can only improve this ratio, by Observation 4.4. Therefore, we have $\ell = 3$ and then

$$\text{LW-3R}(\Gamma) \leq (t_3 + 1)w_2 + (t_3 + 2)w_3 + (t_3 + 3)w_1 + (t_3 + 4)w_0.$$

We have that $3 \not\succ 2 \succ 1 \succ 0$, thus $w_3 \leq (R^2 + 4R)(R^3 - t_3)^{5/3}w_2$, $w_2 > (R^2 + 2R)w_1$ and $w_1 > R^2w_0$. We only need to consider the situation that OPT completed job 1, thus

$$\text{OPT}(\Gamma) \geq Rw_1 + (R + \frac{w_1}{2R^3w_2} + 1)w_3 + (R + \frac{w_1}{2R^3w_2} + 2)w_2 + (R + \frac{w_1}{2R^3w_2} + 3)w_0,$$

because the last arriving job in the minimal counterexample is heavier than the job that arrived just before it. Since $\frac{w_1}{2R^3} < 0.049$, thus $w_3 < (R^2 + 4R)(R^3 - (R + 0.049))^{5/3} > 6.48$. Using $(R^3 + 3)/(R + 3) < R$, $(R^3 + 1.049)/(R + 1.049) > R$ and $(R^3 + 2)/R > R$ we obtain the following bound:

$$\begin{aligned} \frac{\text{LW-3R}(\Gamma)}{\text{OPT}(\Gamma)} &\leq \frac{R^3 + \frac{w_1}{2R^3} + (R^3 + 1 + \frac{w_1}{2R^3})w_3 + (R^3 + 2 + \frac{w_1}{2R^3})w_1 + (R^3 + 3 + \frac{w_1}{2R^3})w_0}{Rw_1 + (R + \frac{w_1}{2R^3} + 1)w_3 + R + \frac{w_1}{2R^3} + 2 + (R + \frac{w_1}{2R^3} + 3)w_0} \\ &< \frac{R^3 + 0.049 + (R^3 + 1.049)w_3 + (R^3 + 2.049)w_1}{Rw_1 + (R + 1.049)w_3 + R + 2.049} \\ &< \frac{R^3 + 0.049 + (R^3 + 1.049)6.48715 + (R^3 + 2.049)/(R^2 + 2R)}{R/(R^2 + 2R) + (R + 1.049)6.48715 + R + 2.049} \approx 1.324 < R. \end{aligned}$$

Case 3.2.2: (Job 2 is interrupted) If OPT completes job 0 first, then

$$\text{LW-3R}(\Gamma) \leq (t_4 + 1)w_5 + (t_4 + 2)w_2 + (t_4 + 3)w_1 + (t_4 + 4)w_0 + (t_4 + 5)w_3 + (t_4 + 6)w_4$$

$$\text{OPT}(\Gamma) \geq w_0 + 2w_2 + 3w_5 + 4w_1 + 5w_3 + 6w_4$$

We have that $5 \succ 2 \succ 1 \succ 0$, thus $1 > (R^2 + 4R)(R^3 - (R + 0.049))^{5/3}w_2 \approx 6.48715w_2$, $w_2 > (R^2 + 2R)w_1$ and $w_1 > R^2w_0$. Since $w_4 \leq w_3 \leq w_0$ we obtain the following bound:

$$\begin{aligned} \frac{\text{LW-3R}(\Gamma)}{\text{OPT}(\Gamma)} &\leq \frac{R^4 + (R^4 + 1)w_2 + (R^4 + 2)w_1 + (R^4 + 3)w_0 + (R^4 + 4)w_3 + (R^4 + 5)w_4}{w_0 + 2w_2 + 3 + 4w_1 + 5w_3 + 6w_4} \\ &< \frac{R^4 + (R^4 + 1)w_2 + (R^4 + 2)w_1 + ((R^4 + 3) + (R^4 + 4) + (R^4 + 5))w_0}{2w_2 + 3 + 4w_1 + 12w_0} \\ &< \frac{R^4 + (R^4 + 1)w_2 + ((R^4 + 2) + ((R^4 + 3) + (R^4 + 4) + (R^4 + 5))/R^2)w_1}{2w_2 + 3 + (4 + 12/R^2)w_1} \\ &< \frac{R^4 + ((R^4 + 1) + ((R^4 + 2) + ((3R^4 + 12)/R^2)/(R^2 + 2R)))/6.48715}{3 + (2 + (4 + 12/R^2)/(R^2 + 2R))/6.48715} \\ &\approx 1.16873 < R. \end{aligned}$$

Otherwise, OPT completes job 1 first and then

$$\text{LW-3R}(\Gamma) \leq (t_4 + 1)w_4 + (t_4 + 2)w_2 + (t_4 + 3)w_1 + (t_4 + 4)w_0 + (t_4 + 5)w_3$$

$$\text{OPT}(\Gamma) \geq Rw_1 + (t_3 + 1)w_4 + (t_3 + 2)w_2 + (t_3 + 3)w_0 + (t_3 + 4)w_3,$$

thus $1 > (R^2 + 4R)(R^3 - (R + 0.049))^{5/3}w_2 \approx 6.48715w_2$, $w_2 > (R^2 + 2R)w_1$ and $w_1 > R^2w_0$. Since $w_3 \leq w_0$ we obtain the following bound:

$$\begin{aligned} \frac{\text{LW-3R}(\Gamma)}{\text{OPT}(\Gamma)} &\leq \frac{R^4 + (R^4 + 1)w_2 + (R^4 + 2)w_1 + (R^4 + 3)w_0 + (R^4 + 4)w_3}{Rw_1 + R + \frac{w_1}{2R^3} + 1 + (R + \frac{w_1}{2R^3} + 2)w_2 + (R + \frac{w_1}{2R^3} + 3)w_0 + (R + \frac{w_1}{2R^3} + 4)w_3} \\ &< \frac{R^4 + (R^4 + 1)w_2 + (R^4 + 2)w_1 + (2R^4 + 7)w_0}{Rw_1 + R + \frac{w_1}{2R^3} + 1 + (R + \frac{w_1}{2R^3} + 2)w_2 + (2R + \frac{w_1}{R^3} + 7)w_0} \\ &< \frac{R^4 + (R^4 + 1)w_2 + ((R^4 + 2) + (2R^4 + 7)/R^2)w_1}{R + \frac{w_1}{2R^3} + 1 + (R + \frac{w_1}{2R^3} + 2)w_2 + (R + (2R + \frac{w_1}{R^3} + 7)/R^2)w_1} \\ &< \frac{R^4 + (R^4 + 1)w_2 + ((R^4 + 2) + (2R^4 + 7)/R^2)w_1}{R + \frac{w_1}{2R^3} + 1 + ((R + \frac{w_1}{2R^3} + 2) + (R + (2R + \frac{w_1}{R^3} + 7)/R^2)/(R^2 + 2R))w_2} \\ &< \frac{R^4 + ((R^4 + 1) + ((R^4 + 2) + (2R^4 + 7)/R^2)/(R^2 + 2R))w_2}{R + \frac{w_1}{2R^3} + 1 + ((R + \frac{w_1}{2R^3} + 2) + (R + (2R + \frac{w_1}{R^3} + 7)/R^2)/(R^2 + 2R))w_2} \\ &< \frac{R^4 + ((R^4 + 1) + ((R^4 + 2) + (2R^4 + 7)/R^2)/(R^2 + 2R))/(6.48715)}{R + 1.049 + ((R + 2.049) + ((R + (2R + 7.098)/R^2))/(R^2 + 2R))/(6.48715)} \\ &\approx 1.32354 < R. \end{aligned}$$

The additional jobs that can arrive at time t_3^+ in both have relative completion time of at most R , thus we still have $\text{ALG}/\text{OPT} \leq R$, by Corollary 4.1 and

Observation 4.4. □

Now, we show that the competitive ratio of LW-3R is optimal for this number of interruptions, as we establish a corresponding lower bound in the k -limited restart setting for our problem. In the next construction, we force the algorithm to use all allowed restarts that required the existence of many jobs with large weights.

Theorem 4.3. *Any algorithm for $1|r_j, \text{online}, p_j = 1, k - \text{restart}|\sum w_j C_j$ has competitive ratio at least R_k , which is the real solution to the equation $R_k^{k+1} + 1 = R_k^{k+2}$.*

Proof. For a contradiction, suppose that there exists an online algorithm A with a competitive ratio R . We construct an adversarial sequence of jobs I . Suppose that algorithm A starts processing job j at time S_j , where the release time r_j is given by $r_j = S_{j-1} + \varepsilon$ for $j > 0$, with $\varepsilon > 0$. Let the weight of job j be w_j , and assume $r_0 = 0$. We begin with job 0 and analyze the following two cases.

Case 1. If $S_j \geq R_k^{j+1} - 1$, then no more jobs arrive, the competitive ratio is at least R_k .

Case 2. If $S_j < R_k^{j+1} - 1$, then another job with w_{j+1} arrives at time $r_{j+1} = S_j + \varepsilon$. If $j + 1 \not\succ j$, no more jobs arrive. The optimal schedule in this scenario is to process job $j + 1$ first and $\text{OPT}(\Gamma) \leq (S_j + 1)w_{j+1} + S_k + 2$. Therefore, we have that there exists a sufficiently large w_{j+1} such that

$$\frac{A(I)}{\text{OPT}(I)} = \frac{S_j + 1 + (S_j + 2)w_{j+1}}{(S_j + 1)w_{j+1} + S_j + 2} \geq R_k.$$

If $j + 1 \succ j$, then job $j + 1$ is started at S_{j+1} . Since $r_{j+1} = S_j + \varepsilon < R_k^{j+1} - 1$, then if $S_{j+1} \geq R_k^{j+2} - 1$ no more jobs arrive, and the competitive ratio is at least R_k for a sufficiently large w_{j+1} . Because, when the algorithm delays a job of sufficiently large weight to avoid a potential interruption, the job's relative completion time becomes at least R_k since its weight can be chosen in a way that dominates the contribution to the competitive ratio in this scenario, this implies that the ratio is at least R_k . Therefore, we only analyze the Case 2 scenario recursively for S_{j+1} .

This process can repeat several times. However, since the algorithm allows at most k interruptions, it must eventually complete a job without further interruption. At that point, we are again in the situation of Case 2 but without further interruption, and the competitive ratio is at least R_k . □

CHAPTER 5

Minimizing the Weighted Makespan

5.1 Introduction

In this chapter, we study the problem of minimizing the weighted makespan. *Makespan minimization* is a fundamental and extensively studied problem in scheduling theory, with a significant body of research devoted to it. A sequence of jobs must be scheduled non-preemptively on m machines. Each job j is characterized by a processing time p_j and a non-negative arrival time r_j , where $1 \leq j \leq n$. The goal is to minimize the *makespan*, defined as the maximum completion time of any job in the schedule. The problem is obviously NP-hard as it generalizes the makespan problem where all jobs arrive at time 0. Regarding the online setting, the best competitive ratio currently known, for general m , is equal to 1.5 and attained by the LPT (Longest Processing Time First) algorithm [50].

A more general objective is the *weighted makespan*, denoted as $WC_{\max} = \max_j \{w_j C_j\}$, which measures the maximum *weighted completion time* of all jobs. Each job j is assigned a positive weight w_j , reflecting its relative importance. The classical makespan minimization problem is a special case in which all weights are equal. This objective function was first considered only fairly recently and has received some attention [45, 119, 137, 148]. Even on a single machine, it is nontrivial to optimize the weighted makespan non-preemptively if jobs arrive over time. In the standard three-field notation, this problem is denoted as $1 \mid r_j \mid WC_{\max}$. If all jobs arrive at time 0 or preemptions are allowed, then the algorithm Largest Weight (LW) which always runs the job that has largest weight is 1-competitive.

Feng and Yuan [137] were the first to consider the weighted makespan in the context of single-machine scheduling. Li [119] explored the online version of this problem and established a lower bound of 2 on the competitive ratio for

deterministic algorithms. Moreover, for the problem on a single machine, he provided an online algorithm with a competitive ratio of 3. For the case of identical machines where all jobs have the same processing time, Li proposed an online algorithm with the best-possible competitive ratio of $\frac{\sqrt{5}+1}{2}$. Chai et al. [45] further improved the results on a single machine, by developing two deterministic online algorithms with a competitive ratio of 2.

One important extension to the classical scheduling framework is the use of *restarts*. However, for the important objective function of total weighted flow time, where the flow time of a job is the time between its arrival and its completion, it is known that restarts do not help [67]: any online algorithm that can restart jobs has a competitive ratio of $\Omega(n)$.

Liang et al. [121] recently explored the weighted makespan problem in the restart model for online single-machine scheduling. They proved that when only a single restart is allowed across all jobs, no online algorithm can achieve a competitive ratio lower than 2, matching the lower bound for the problem without restarts [119]. However, in the case where all jobs have unit processing times, they presented the best possible online algorithm with a competitive ratio of 1.4656.

In the *offline* setting, early developments of approximation schemes for makespan minimization on parallel machines can be found in the works of Hochbaum and Shmoys [93] and Alon et al. [9]. Additionally, Jansen et al. [104] proposed an efficient polynomial-time approximation scheme (EPTAS) for identical parallel machines and Kones and Levin [111] introduced a unified framework for designing EPTAS for general makespan minimization on parallel machines. Lu et al. in [126] considered the single-machine scheduling problem with release dates and job delivery to minimize weighted makespan and provided a polynomial-time approximation scheme (PTAS) for the problem. In [127], they considered the offline single-machine problem with job rejection to minimize weighted makespan but without release dates. They showed that this problem is also NP-hard and proposed a fully polynomial-time approximation scheme (FPTAS). Ruiqing Sun [148] introduced a $(2 - \frac{1}{m})$ -approximation algorithm for the weighted makespan scheduling problem on m identical parallel machines, again in the setting without release dates ($P||WC_{\max}$). The algorithm runs List Scheduling on the jobs after sorting them by decreasing weight. Ruiqing also introduced a randomized EPTAS for this problem, as well as a randomized FPTAS for the case when m is fixed. Recently, Lili Zuo et al. [158] studied the single-machine scheduling problem with a restricted rate-modifying activity (RMA) aimed at

minimizing the weighted makespan. They showed that several special cases are binary NP-hard and developed an FPTAS for these variants.

5.1.1 Our Results

We study the problem of online scheduling with restarts for minimizing the weighted makespan $WC_{\max}$. Specifically, we establish a new lower bound of 1.4656 on the competitive ratio for deterministic online algorithms in the general setting with restarts. Remarkably, this is the same value that Liang et al. [121] achieved for jobs of unit size and a single allowed restart. However, the problems are different.

Our main result is a deterministic online algorithm that achieves a competitive ratio better than 1.3098 for the case where all jobs have identical processing times. The algorithm balances a greedy approach, favoring heavier jobs, with the consideration that not too much time should be wasted on jobs that are eventually interrupted anyway. If a job arrives that is heavier than the currently running job, the new job is started instead unless the current job has already been running for sufficiently long. What sufficiently long is depends on the current time: we do not interrupt the running job in the case that the new job, if it is started after the current job completes, still completes within a factor of 1.3098 of its smallest possible completion time (which is achieved if the new job starts immediately when it arrives).

We also prove a lower bound of 1.2344 for this case. In determining both lower bounds, we first identified some difficult inputs and then used a computer to optimize the parameters. For the optimized values we then considered in which cases the goal competitive ratio was achieved. This resulted in a system of equalities that we solved to get exact lower bounds.

5.2 Lower Bound

Consider the online problem $1|r_j, \text{online, restart}|WC_{\max}$. The lower bound presented here shows that the problem maintains its difficulty, even when unlimited restarts are allowed.

Theorem 5.1. *Any algorithm for the problem $1|r_j, \text{online, restart}|WC_{\max}$ is at least $R_1 \approx 1.4656$ competitive which is the real root of the equation $R_1^3 - R_1^2 - 1 = 0$.*

Proof. For contradiction, suppose that there exists an online algorithm ALG

with a competitive ratio of less than R_1 .

We consider the following job instance I . At time $r_1 = 0$, job 1 with $w_1 = 1$ and $p_1 = 1$ arrives. Then job 2 with $w_2 = \frac{1}{R_1-1} \approx 2.1478$ and $p_2 = 0$ arrives at time $r_2 = \frac{1}{R_1(R_1-1)} - 1 \approx 0.4655$.

Case 1. If ALG runs job 1 before job 2, no more jobs arrive. Then we have $\text{ALG}(I) \geq \max(1, w_2)$ and $\text{OPT}(I) \leq \max(r_2 w_2, 1 + r_2) = 1 + r_2$ and we get the following contradiction to our assumption

$$\frac{\text{ALG}(I)}{\text{OPT}(I)} \geq \frac{\max(1, w_2)}{\max(r_2 w_2, 1 + r_2)} = \frac{\frac{1}{R_1-1}}{\frac{1}{R_1(R_1-1)}} = R_1.$$

Case 2. If ALG runs job 2 first, then job 2 is started and also finishes no earlier than at time r_2 and job 1 is restarted no earlier than at time r_2 . Now, two more jobs arrive at time $r_3 = r_4 = 1$: job 3 with $w_3 = \frac{1}{R_1-1} \approx 2.1478$ and $p_3 = 0$ and job 4 with $w_4 = 1$ and $p_4 = \frac{1}{R_1-1} - 1 \approx 1.1478$. Note that processing job 3 before job 4 does only improve (or maintain) the result, since job 3 and job 4 are released at the same time and $p_3 = 0$ holds. Therefore, we do not consider the other case.

OPT schedules first job 1 and then the jobs 2,3 and 4 in this order, thus $\text{OPT}(I) = \max(1, w_2, w_3, 1 + p_4) = 1 + p_4 = \frac{1}{R_1-1}$.

Case 2.1. If ALG runs job 3 and job 4 before job 1, then we have

$$\frac{\text{ALG}(I)}{\text{OPT}(I)} \geq \frac{\max(r_2 w_2, w_3, 1 + p_4, p_4 + 2)}{\max(1, w_2, w_3, 1 + p_4)} = \frac{\frac{1}{R_1-1} + 1}{\frac{1}{R_1-1}} = R_1.$$

Since job 4 and job 1 have equal weight, the job order 3,4,1 has the same weighted makespan as 3,1,4.

Case 2.2. If ALG runs job 1 directly after job 2, then job 3 starts and finishes not before $r_2 + 1$ and job 4 finishes not before time $r_2 + 1 + p_4$. In this case we

have $\text{ALG}(I) \geq \max(r_2 w_2, r_2 + 1, (r_2 + 1)w_3, r_2 + 1 + p_4) = (r_2 + 1)w_3$. We have

$$\begin{aligned}
& (r_2 + 1)w_3 \geq r_2 + 1 + p_4 \\
& \Leftrightarrow \frac{1}{R_1(R_1 - 1)} \cdot \frac{1}{R_1 - 1} \geq \frac{1}{R_1(R_1 - 1)} + \frac{1}{R_1 - 1} - 1 \\
& \Leftrightarrow 1 \geq (R_1 - 1) + (R_1^2 - R_1) - (R_1^3 + -2R_1^2 + R_1) \\
& \Leftrightarrow 0 \leq R_1^3 - 3R_1^2 + R_1 + 2 \\
& \Leftrightarrow 0 \leq (R_1^3 - R_1^2 - 1) - 2R_1^2 + R_1 + 3 \\
& \Leftrightarrow 0 \leq -2R_1^2 + R_1 + 3
\end{aligned}$$

using $R_1^3 - R_1^2 - 1 = 0$ and $-2R_1^2 + R_1 + 3 = (-2R_1 + 3)(R_1 + 1) \geq 0$ for $R_1 \in (-1, 1.5)$. Thus,

$$\begin{aligned}
\frac{\text{ALG}(I)}{\text{OPT}(I)} & \geq \frac{\max(r_2 w_2, r_2 + 1, (r_2 + 1)w_3, r_2 + 1 + p_4)}{\max(1, w_2, w_3, 1 + p_4)} \\
& = \frac{\frac{1}{R_1(R_1 - 1)} \cdot \frac{1}{R_1 - 1}}{\frac{1}{R_1 - 1}} = \frac{1}{R_1(R_1 - 1)} = R_1
\end{aligned}$$

using $R_1^3 - R_1^2 - 1 = 0$. □

Theorem 5.2. *Any algorithm for the problem $1|r_j, \text{online}, p_j = 1, \text{restart}|WC_{\max}$ is at least $R_2 \approx 1.2344$ competitive which is the real root of the equation $4R_2^3 - R_2^2 - 6 = 0$.*

Proof. For a contradiction, suppose that there exists an online algorithm ALG with a competitive ratio of less than R_2 . We consider the following job instance. At time $r_1 = 0$, job 1 with $w_1 = 1$ arrives. Then job 2 arrives with $w_2 = \frac{4}{3}$ at time $r_2 = \frac{3}{R_2} - 2 \approx 0.430$.

Case 1. If ALG completes job 1 before job 2, a final job 3 with $w_3 = \frac{3+r_2}{2+r_2} \approx 1.411$ arrives at time $r_3 = 1 + r_2 = \frac{3}{R_2} - 1 \approx 1.430$. The following schedule is feasible: job 2 is completed at time $1 + r_2$, job 3 is completed at time $2 + r_2$ and job 1 is completed at time $3 + r_2$. Thus, $\text{OPT}(I) \leq 3 + r_2 = 3/R_2 + 1$.

If ALG completes job 2 before job 3, the job order is 1, 2, 3. Thus, we have

$$\begin{aligned}
 \frac{\text{ALG}(I)}{\text{OPT}(I)} &\geq \frac{\max(1, 2w_2, 3w_3)}{3/R_2 + 1} = \frac{\max(8/3, (9+3r_2)/(2+r_2))}{3/R_2 + 1} \\
 &= \frac{\frac{3(3+(3/R_2-2))}{3/R_2}}{3/R_2 + 1} \\
 &= \frac{\frac{3(3/R_2+1)}{3/R_2}}{3/R_2 + 1} \\
 &= R_2.
 \end{aligned}$$

If ALG completes job 3 before job 2, then ALG would schedule job 1, then job 3 and finally job 2. Since $w_3(r_3 + 1) \leq w_2(r_3 + 2)$, we obtain

$$\begin{aligned}
 \frac{\text{ALG}(I)}{\text{OPT}(I)} &\geq \frac{\max(1, w_3(r_3 + 1), w_2(r_3 + 2))}{3/R_2 + 1} = \frac{w_2(r_3 + 2)}{3/R_2 + 1} \\
 &= \frac{w_2((3/R_2 - 1) + 2)}{3/R_2 + 1} = \frac{4}{3} \geq R_2.
 \end{aligned}$$

Case 2. If ALG completes job 2 before job 1, then a (different) third job 3 with $w_3 = 2$ arrives at time $r_3 = \frac{6}{R_2^2} - 3 \approx 0.938$.

If now ALG completes job 2 before job 3, no more jobs arrive and OPT could first schedule job 3, then job 2 and finally job 1. Therefore, $\text{OPT}(I) \leq \max(w_3(r_3 + 1), w_2(r_3 + 2), r_3 + 3) = r_3 + 3$. However, after completing job 2, in the best case, ALG schedules job 3 after job 2 and before job 1, since job 3 has higher weight. Thus, $\text{ALG}(I) = \max(w_2(r_2 + 1), w_3(r_2 + 2), r_2 + 3) = w_3(r_2 + 2)$ and we have

$$\frac{\text{ALG}(I)}{\text{OPT}(I)} \geq \frac{w_3(r_2 + 2)}{r_3 + 3} = \frac{2((3/R_2 - 2) + 2)}{(6/R_2^2 - 3) + 3} = \frac{6/R_2}{6/R_2^2} = R_2.$$

Now, if ALG completes job 3 before job 2, a final fourth job 4 with $w_4 = 1$ arrives at time $r_4 = 3$. A feasible schedule s would be the job order 1, 3, 2, 4 with $WC_s = 4$. However, if ALG completes job 3 before job 2, at best, ALG first schedules job 3 at time r_3 , then job 2 and finally the two jobs with weight 1 in arbitrary order, since job 2 has higher weight than jobs 1 and 4 which have the

same weight. Therefore, we have

$$\begin{aligned} \frac{\text{ALG}(I)}{\text{OPT}(I)} &\geq \frac{\max((r_3 + 1)w_3, (r_3 + 2)w_2, r_3 + 3, r_3 + 4)}{4} \\ &= \frac{r_3 + 4}{4} = \frac{6/R_2^2 + 1}{4} = R_2, \end{aligned}$$

using $4R_2^3 - R_2^2 - 6 = 0$. □

5.3 Limited Largest Weight (LLW) algorithm

In this section, we consider the problem when all jobs have identical processing times. We first present our deterministic online algorithm Limited Largest Weight (LLW) which has a competitive ratio of $R \approx 1.3098$, the positive real root of $3R^3 - 2R^2 - R - 2 = 0$. The other roots of this function are complex. We also provide a lower bound for this problem.

We first define two versions of the Largest Weight (LW) strategy.

- *LW with interruptions:* If the machine is idle, the next arriving job is started when it arrives. A running job is interrupted if a new job with higher weight arrives. If a job is interrupted or finishes and there are jobs waiting, the heaviest waiting job starts processing immediately.
- *LW without interruptions:* The same rules apply, except that once a job has started it is never interrupted.

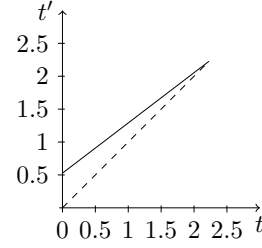
The idea of our algorithm is as follows. In principle, we prefer to run the heaviest job. This would mean running LW with interruptions. However, if a significant part of the currently running job c has already been completed, it can be better to let it finish first, rather than interrupt it for an incoming job j . This depends on what the completion time of j will be when it completes immediately following c versus what its completion time will be if it starts immediately, and also on the relative weights of c and j . If c started at time t and j arrives at time t' , then if c is allowed to complete j will complete not before time $t + 2$. We use the following very simple rule: we will interrupt c if j is heavier than c and $t + 2$ is more than R times the completion time of j if it starts immediately, which is $t' + 1$ (assuming no further interruptions occur). We will see shortly that this condition will never be satisfied after time $(2 - R)/(R - 1)$; from then on we simply run LW without interruptions. A complete description of our algorithm follows.

Our algorithm starts the first arriving job when it arrives.

Once the algorithm starts a job at time $\frac{2-R}{R-1}$ or later, it runs LW without interruptions until all jobs have finished.

Whenever it starts a job at time $t < \frac{2-R}{R-1} \approx 2.2279$, it runs LW with interruptions until time

$$t' = \frac{t+2}{R} - 1. \quad (5.1)$$



Note that for all $t < \frac{2-R}{R-1}$ it holds that $t < t'$ since

$$t < \frac{2-R}{R-1} \Leftrightarrow tR - t < 2 - R \Leftrightarrow t < \frac{t+2}{R} - 1.$$

We also have that t' never exceeds the boundary $(2-R)/(R-1)$. During this time, if an interruption occurs at time t_i , then we update $t := t_i$ and set t' according to (5.1). If time t' is ever reached without an interruption, the running job is allowed to continue until completion without any interruptions. After this, the algorithm resumes running LW either with or without interruptions, depending on the current time as described above.

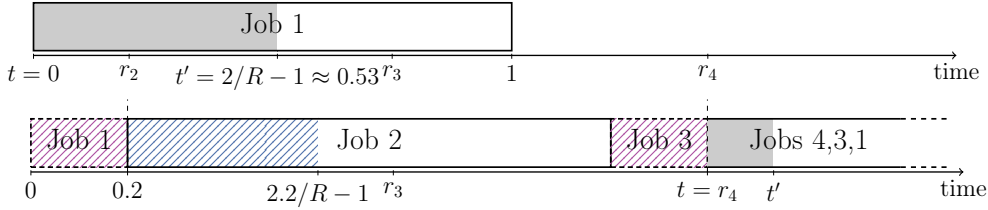


Figure 5.1: This example shows the behavior of our algorithm.

For any job that starts at some time $t < \frac{2-R}{R-1}$, the interval (t, t') is called an LW-phase; however, if an interruption occurs in such an LW-phase, the LW-phase ends at that time, and the next LW-phase starts immediately (because this only happens before time $\frac{2-R}{R-1}$). In Fig. 5.1, we sketched in an example the behavior of the algorithm for the following job sequence: $((0, 1), (0.2, 1.1), (0.7, 1.6), (1.4, 2.3))$. Here each pair specifies first a release date and then the weight of the job released at this time. The first figure shows the situation at time $t = 0$ and the first LW-phase with values t and t' . In this LW-phase job 1 gets interrupted because job 2 is heavier. The second figure shows the schedule of the algorithm at time $t = r_4$. Job 2 is not interrupted during its LW-phase, so it continues without interruption. At its completion time the algorithm starts the waiting job 3. Job 3 gets interrupted in its LW-phase and after that the remaining jobs are

scheduled with LLW in the order 4, 3, 1.

5.4 Competitive analysis of LLW

We now prove that LLW is R -competitive. Recall that R is the positive real root of $3R^3 - 2R^2 - R - 2 = 0$.

We call a job *critical* for an input I if its weighted completion time is equal to $\text{LLW}(I)$. That is, no other job in the schedule of LLW for input I has a higher weighted completion time. An input I may contain several critical jobs.

We assume the existence of a *minimal counterexample* I . A minimal counterexample has ratio strictly larger than R and consists of the smallest number of jobs. We show by contradiction that such a counterexample does not exist. Without loss of generality, we can assume that no two jobs arrive at the same time, as otherwise, we could postpone the lighter job(s) by an arbitrarily small amount.

Consider an arbitrary critical job in I and let S be the maximal set of jobs that run consecutively without interruptions and that ends with this critical job. We scale the input such that this critical job has weight 1. Thus, $\text{LLW}(I)$ is simply the completion time of the critical job.

We now derive a sequence of properties that a counterexample must have, which will eventually lead to the conclusion that no counterexample exists. We begin by showing that the critical job is the lightest job in S (Property 5.1), and that OPT starts running its first job in S relatively early (Property 5.2). LLW starts its first job in S when it arrives (Property 5.3), and the arrival of this job actually caused the interruption of another job $h \in S$ (Property 5.4). We then show that LLW does not complete any job before job 1 (Property 5.7). This requires some preparation: we first show that LLW does not complete two or more jobs before job 1 (Property 5.5) and then give a structural property for the case where LLW completes one job before job 1 (Property 5.6).

Property 5.1. *The critical job is the lightest job in S .*

Proof. Suppose not. Let a job be *heavy* if its weight is at least that of the critical job. We renumber the jobs according to their order in S and starting from 0 for the last non-heavy job in S (any jobs in S before this job have negative indices). Then after job 0, all jobs in S are heavy up to and including the critical job. No later job can be heavy because then the critical job would not be critical. Let H

be the set of these heavy jobs in S , and let $k = |H|$. Then

$$\text{LLW}(I) = s_0 + k + 1.$$

All jobs in H arrive after time s_0 , and they are all at least as heavy as the critical job. We have

$$\text{OPT}(I) \geq s_0 + k.$$

The competitive ratio is at most R as long as $s_0 + k \geq 1/(R - 1)$, so the only problematic case is $s_0 < \frac{2-R}{R-1}$. By the rules of the algorithm, this means no jobs from H arrive before time $(s_0 + 2)/R - 1$, so we also have the lower bound

$$\text{OPT}(I) \geq \frac{s_0 + 2}{R} + k - 1.$$

Therefore

$$\frac{\text{LLW}(I)}{\text{OPT}(I)} \leq \frac{s_0 + k + 1}{(s_0 + 2)/R + k - 1}.$$

Setting $s_0 = 0$ and $k = 1$ we get $2/(2/R) = R$. This shows that the competitive ratio is at most R for all values of k and s_0 . $\square$

Given Property 1, from this point on we simply number the jobs in S according to their order in S , with the first job in S having index 1 and starting (for the last time) at time s_1 . We define $k = |S|$, so we have

$$\text{LLW}(I) = s_1 + k.$$



Figure 5.2: This figure illustrates the set S with jobs 1 up to k . Job k is the critical job, and the jobs after k are not in S and are lighter.

Note that there may be additional jobs that are run consecutively without interruption immediately after S , but not immediately before S . If OPT runs the jobs in S in (essentially) the same order as LLW and does not start them too much earlier than LLW , then the competitive ratio is maintained. We formalize this intuition in the next definition.

Definition 5.1 (Good set). *Suppose OPT starts processing a job in S at time*

$t^* \geq (s_1 + k)/R - k$ and completes a job which is at least as heavy as the critical job at time $t^* + k$ or later. Then S is good.

Property 5.2. S is not good. OPT starts running its first job in S before time $(s_1 + k)/R - k$.

Proof. Suppose S is good. Then $\text{OPT}(I) \geq t^* + k \geq (s_1 + k)/R$, so I is not a counterexample. Second, suppose OPT starts running the first job in S at time $(s_1 + k)/R - k$ or later. Then again $\text{OPT}(I) \geq (s_1 + k)/R$. $\square$

Property 5.3. No job completes at time s_1 . Job 1 in S arrives at time s_1 . If a job in S arrives before s_1 , it is not heavier than job 1.

Proof. The first statement follows from the definition of S . It means that job 1 did not wait for another job to complete before starting. Since LLW also does not idle the machine if there is a job available, it starts job 1 immediately when it arrives, at time s_1 .

Suppose some job $j \in S$ arrives at time $r_j < s_1$ and is heavier than job 1. When j arrives, LLW may continue the job it is running, but LLW will not start any new job that is lighter than j until it has completed j .

By the definition of S , LLW does not complete any job in S before time s_1 . So it completes j at some time $c_j > s_1$. In the interval $[r_j, c_j]$, LLW only starts jobs of weight at least w_j . Since $s_1 \in (r_j, c_j)$, this contradicts the fact that LLW starts job 1 at time s_1 . This proves the last statement. $\square$

Property 5.4. A job $h \in S$ was interrupted and then job 1 starts at time

$$R - 1 < s_1 < \frac{2 - R}{R - 1}.$$

We have $s_1 > k(R - 1)$ and $2 \leq k \leq 7$.

Proof. If $s_1 \leq R - 1$, OPT starts processing the first job in S at the earliest at time $0 = \frac{R-1+1}{R} - 1 \geq \frac{s_1+1}{R} - 1$. By Definition 5.1 and because the critical job is the lightest job in S , S is therefore good. We conclude that $s_1 > R - 1$ in I . More generally, if OPT starts its first job from S at time $\frac{s_1}{R} - k(1 - \frac{1}{R})$ or later, then S is by Definition 5.1 good, as $|S| = k$.

Hence, OPT starts at least one job in S at some time $s' < (s_1 + k)/R - k < s_1$, implying that at least one job in S arrives at or before time s' . From

$(s_1 + k)/R - k > s' \geq 0$ we get $s_1 > k(R - 1)$.

Since LLW does not complete any job in S before time s_1 but also does not idle if at least one job is available, LLW does not idle in the interval $(s', s_1]$. By Property 5.3, LLW does not complete a job at time s_1 . Since LLW was not idle at time s_1 , LLW interrupted some job $h \neq 1$ at time s_1 .

If $h \notin S$, then h is lighter than the critical job, because otherwise h would be started not later than s_k by the rules of the algorithm, implying that $h \in S$ after all: a contradiction. Since h is lighter than the critical job which is the lightest job in S , all jobs in S are heavier than h . Then all jobs in S arrived at or after time s_1 because otherwise LLW would not still have been running h at time s_1 , contradicting $s' < s_1$.

Thus, $h \in S$ and with this, we have $k \geq 2$. Moreover, since LLW interrupted a job h at time s_1 , we get $s_1 < \frac{2-R}{R-1}$ by the rules of the algorithm. From $(R - 1)k < s_1 < \frac{2-R}{R-1}$ we get $k \leq 7$. $\square$

Property 5.5. LLW does not complete two jobs before time s_1 .

Proof. We prove this by contradiction. Suppose LLW completes two jobs before time s_1 , called q_1 and q_2 in order. Then $s_1 \in [2, \frac{2-R}{R-1})$. By Property 5.4, OPT starts running its first job j in S (say $j \neq 1$) before time

$$\frac{s_1 + k}{R} - k < \frac{\frac{2-R}{R-1} + 2}{R} - 2 < 1.23.$$

Note that for the starting time of job q_2 we have $s_{q_2} \geq 1$ and with this, we get that the LW-phase of job q_2 ends not before

$$\frac{s_{q_2} + 2}{R} - 1 \geq \frac{1 + 2}{R} - 1 = \frac{3}{R} - 1 > 1.29.$$

Suppose j is heavier than q_2 . Then either j is started before q_2 or q_2 is interrupted, as j arrives before or during its LW-phase. In both scenarios LLW completes j before q_2 , which means q_2 does not complete before time 3. This contradicts $s_1 < \frac{2-R}{R-1} \approx 2.2279$, since q_2 completes before time s_1 . Therefore, j is lighter than q_2 .

Now, if j is not heavier than q_1 , OPT needs to complete $k + 2$ jobs that are at least as heavy as the critical job, so $\text{OPT}(I) \geq k + 2$ whereas $\text{LLW}(I) < \frac{2-R}{R-1} + k \approx 2.2279 + k$. Therefore, the competitive ratio is less than R for $k \geq 2$. Hence, j is heavier than q_1 .

Job j must arrive after time $2/R - 1$ as j is heavier than q_1 but does not cause q_1 to be interrupted. This implies $\text{OPT}(I) \geq 2/R - 1 + k + 1 \approx 1.5270 + k$ as OPT must also complete q_2 which is heavier than the critical job and heavier than q_1 and can therefore also not arrive before time $2/R - 1$. Since $\text{LLW}(I) < \frac{2-R}{R-1} + k \approx 2.2279 + k$, the competitive ratio is less than R for $k \geq 2$.

Consequently, in both cases the competitive ratio is less than R for $k \geq 2$, which contradicts the assumption that I is a counterexample. $\square$

Property 5.6. *If LLW completes a job q before time s_1 , then q is lighter than the critical job, and OPT starts the first job in S at time $(s_q + 2)/R - 1$ or later.*

Proof. We prove this by contradiction. Suppose LLW completes a job q before time s_1 which is at least as heavy as the critical job. Since OPT must complete this job as well, OPT must start its first job in $S \cup \{q\}$ before time $(s_1 + k)/R - k - 1 \approx 0.764s_1 - 0.237k - 1$. Since $s_1 < \frac{2-R}{R-1} < 2.228$, this implies $k \leq 2$, so $k = 2$ by Property 5.4. It follows that after time $s_q + 1$, there is exactly one interruption, namely at time s_1 . Otherwise it would have to be the two most recent jobs (at time s_1) which are in S , since they are the heaviest, implying that S is good: let t be the last starting time before s_1 , then LLW starts running the jobs in S (without further interruptions) at time $s_1 < (t + 2)/R - 1$ and OPT starts running them at time t or later because the job that LLW started at time t arrived at that time. Since $t \geq s_q + 1$, this means S is good.

Because LLW does not complete another job at time s_q by Property 5.5, LLW starts job q when it arrives. Then, as we have seen, it interrupts the next job it starts at time s_1 , and then it completes two jobs. On the other hand, OPT must complete three jobs that are at least as heavy as the critical job starting from time s_q . After completing q , LLW does not idle, otherwise I would not be minimal. Therefore, we have $s_1 < (s_q + 3)/R - 1$ and this means that I is not a counterexample in this case: the competitive ratio is at most

$$\frac{(s_q + 3)/R + 1}{s_q + 3} \leq \frac{3/R + 1}{3} < R.$$

The first statement of the lemma follows, and it implies that all jobs in S are heavier than q . Such a job would cause q to be interrupted if it arrived before time $(s_q + 2)/R - 1$. $\square$

Property 5.7. *LLW does not complete a job before time s_1 .*

Proof. By definition, LLW does not complete a job at time s_1 . Let q be the last job that LLW completes before time s_1 . By Property 5.6, q is lighter than the critical job. After completing q , LLW does not idle, otherwise I would not be minimal. Any job started after q but before time s_1 is interrupted. If there is exactly one interruption after time $s_q + 1$ (namely at time s_1), then $\text{LLW}(I) \leq (s_q + 3)/R + k - 1$ by (5.1) and $\text{OPT}(I) \geq (s_q + 2)/R + k - 1$ by Property 5.6; the ratio is

$$\frac{\text{LLW}(I)}{\text{OPT}(I)} \leq \frac{(s_q + 3)/R + k - 1}{(s_q + 2)/R + k - 1} \leq \frac{(0 + 3)/R + 2 - 1}{(0 + 2)/R + 2 - 1} < 1.303,$$

for $k \geq 2$ and $s_q \geq 0$.

If there are additional interruptions, these are all of jobs that arrived after time $s_q + 1$. Let t be the arrival time of the first job in S . In this case, as in the previous proof, the k most recently arrived jobs at time s_1 are the heaviest k jobs that exist at that time. Hence, S does not contain any job that arrived before the first of those k jobs. Put another way, t is not smaller than the arrival time of the first of those k jobs.

If $k = 2$, we find that I is not a counterexample: we have $\text{OPT}(I) \geq t + 2$ and $\text{LLW}(I) \leq (t + 2)/R + 1$. The ratio is less than R . For the remaining bounds we always either use the bound $\text{OPT}(I) \geq t + k$ or $\text{OPT}(I) \geq (s_q + 2)/R + k - 1$.

If there are exactly two restarts, then

$$\text{LLW}(I) \leq \frac{(s_q + 3)/R + 1}{R} - 1 + k,$$

and the ratio is less than $4.513/3.526 \approx 1.280$ for $k \geq 3$ and $s_q \geq 0$.

Else if $k = 3$ and t is the arrival time of the first job in S we have $\text{OPT}(I) \geq t + 3$ and

$$\text{LLW}(I) \leq \frac{(t + 2)/R - 1 + 2}{R} - 1 + 3,$$

and the ratio is less than R .

Else if there are exactly three restarts, then

$$\text{LLW}(I) \leq \frac{\frac{(s_q + 3)/R + 1}{R} + 1}{R} - 1 + k,$$

and the ratio is less than $5.682/4.527 \approx 1.256$ for $k \geq 4$ and $s_q \geq 0$.

Finally, if $k = 4$ and t is the arrival time of the first job in S we have $\text{OPT} \geq t + 4$

and

$$\text{LLW}(I) \leq \frac{\frac{(t+2)/R-1+2}{R} - 1 + 2}{R} - 1 + 4,$$

and the ratio is less than R . In all cases, we now have $\text{LLW}(I) \leq \frac{2-R}{R-1} + k$ and $\text{OPT}(I) \geq \frac{s_q+2}{R+k-1}$. The ratio is less than $7.22815/5.52698 \approx 1.30776$ for $k \geq 5$ and $s_q \geq 0$. $\square$

By Property 5.7 and Property 5.4, in I it happens one or more times that a job is interrupted before any job completes. All interrupted jobs started when they arrived. The final interruption before the critical job completes is at time s_1 . Let m be the total number of interruptions. As in the proof of Property 5.7, at time s_1 , any job that arrived more than k jobs ago is not in S because it is too light. Recall that $2 \leq k \leq 7$ by Property 5.4. We also have $k \geq m + 1$ (if k were smaller, we could just remove previous jobs from I without affecting the objective value or increasing the optimal cost). As before $\text{LLW} = s_1 + k$. We now show for all values m by contradiction that I is not a counterexample.

If $m = 1$, then, since OPT starts its first job in S before time $(s_1 + k)/R - k$, we have $s_1 < ((s_1 + k)/R - k + 2)/R - 1$. For $k \geq 2$ this leads to $s_1 < 0.4$:

$$\begin{aligned} s_1 &< ((s_1 + 2)/R - 2 + 2)/R - 1 \\ \Leftrightarrow s_1 &< ((s_1 + 2)/R)/R - 1 \\ \Leftrightarrow s_1 &< \frac{s_1}{R^2} + \frac{2}{R^2} - 1 \\ \Leftrightarrow s_1(1 - \frac{1}{R^2}) &< \frac{2}{R^2} - 1 \\ \Leftrightarrow s_1 &< \frac{\frac{2}{R^2} - 1}{1 - \frac{1}{R^2}} \end{aligned}$$

contradicting the bound of Property 5.4 $s_1 \geq 2(R - 1)$.

If $m = 2$, $\text{OPT}(I) \geq \max(0, ((s_1 + 1)R - 1)R - 2) + k$ and $k \geq 3$. The ratio is at most R since $s_1 > 3(R - 1)$. It is exactly R for $s_1 = 3(R - 1)$.

It is easily checked that the ratio is at most R for larger values of m as well. In conclusion, we have proved the following theorem.

Theorem 5.3. *The LLW algorithm is R -competitive.*

5.5 Offline Setting

In this section, we consider the offline problem of scheduling for minimizing the weighted makespan $WC_{\max}$ in the presence of release dates on a single machine. Specifically, we study the problem $1|r_j|WC_{\max}$ and prove that the problem is NP-hard. Following the approach of Afrati et al. [3] by combining the dynamic programming approach with the vector trimming technique, we provide the first PTAS for this objective function on parallel machines that works in the setting with release dates.

NP-hardness

We prove strong NP-hardness of the problem $1|r_j|WC_{\max}$ via a reduction from 3Partition. Additionally, we show that the problem remains weakly NP-hard when only allowing ℓ many weight classes, where ℓ is a constant ≥ 2 , via a reduction from Partition.

Definition 5.2 (3Partition). *Given a set $A = \{a_1, \dots, a_{3m}\}$ of $3m$ integers and a positive integer K , such that $\frac{K}{4} < a_i < \frac{K}{2}, \forall i \in [3m]$ and $\sum_{a \in A} a = mK$. Is there a partition of the elements in A into m disjoint subsets $\{S_1, \dots, S_m\} \subseteq A$ such that $|S_i| = 3$ and $\sum_{a \in S_i} a = K$ hold for all $i \in [m]$?*

In 1975, Garey and Johnson [79] proved strong NP-hardness for 3Partition. This leads to the following.

Theorem 5.4. *The problem $1|r_j|WC_{\max}$ is strongly NP-hard.*

Proof. We reduce the 3Partition problem to the following decision problem: Does there exist a schedule σ for the problem $1|r_j|WC_{\max}$ such that $\max_j (w_j C_j)(\sigma) \leq y$ for given y . Given an instance I of 3Partition, we construct an instance for $1|r_j|WC_{\max}$ as follows: For each $i \in [3m]$, we create job i with release date $r_i = 0$, processing time $p_i = a_i$ and weight $w_i = 1$. Additionally, we create $m - 1$ control-jobs $3m + 1, \dots, 4m - 1$ as follows: For each $i \in [m - 1]$ create a job $j = 3m + i$ with release dates $r_j = iK$, processing time $p_j = 0$ and weight $w_j = \frac{m}{i}$. Set $y = mK$.

The described instance of the problem $1|r_j|WC_{\max}$ can be computed in polynomial time.

$\implies$: Assume that the given 3Partition instance is a yes-instance. Then there exists a partition of A into m disjoint sets $S_1, \dots, S_m$ such that

$|S_i| = 3$ and $\sum_{a \in S_i} a = K$ hold for all $i \in [m]$. W.l.o.g. assume that $\{a_{3(i-1)+1}, a_{3(i-1)+2}, a_{3i}\} = S_i$. Now, schedule the three jobs $3(i-1) + 1, 3(i-1) + 2$ and $3i$ in the time interval $[(i-1)K, iK]$. We know that they fit, as $p_{3(i-1)+1} + p_{3(i-1)+2} + p_{3i} = \sum_{a \in S_i} a = K$. As the final job finishes at time mK , we have for all jobs $j = 1, \dots, 3m$ that $w_j C_j \leq mK = y$. Also, note that there exists no job j that is being processed at any time before and after time iK .

Now, consider the control-jobs $j = 3m + i, i \in [m-1]$. Schedule those jobs at the time they arrive, i.e., at time iK . As they have processing time 0, they do not clash with any other job. By definition of those jobs, we have $w_j C_j = \frac{m}{i} \cdot iK = mK = y$.

Thus, the constructed $1|r_j|WC_{\max}$ instance is a yes-instance.

$\Leftarrow$: Assume that the constructed $1|r_j|WC_{\max}$ instance is a yes-instance. This means that there exists a schedule σ with $\max_j (w_j C_j)(\sigma) \leq y$. First, consider the control-jobs $j = 3m + i, i \in [m-1]$. By definition of the jobs, they are released at time $r_j = iK$ and have processing time $p_j = 0$. Thus, the earliest they can be completed is at time iK . Therefore, $w_j C_j \geq \frac{m}{i} \cdot iK = mK$. With $\max_j (w_j C_j)(\sigma) \leq y$, we know that they have to be scheduled the time they are released.

Now, we schedule the remaining jobs $j = 1, \dots, 3m$. Since they have weight $w_j = 1$, each job has to be scheduled before time mK . As $\sum_{j=1}^{3m} p_j = \sum_{a \in A} a = mK$, there is no idle time in the time interval $[1, mK]$. Remember that the control-jobs have fixed times at $iK, \forall i \in [m-1]$. Thus, the other jobs have to be scheduled in the time slot $[1, K]$, in a time slot between two control-jobs or in the time slot $[(m-1)K, mK]$. Since all remaining jobs have processing time $p_j \in (\frac{K}{4}, \frac{K}{2})$, exactly three jobs are scheduled in each time slot. To show this, first assume that there is a time slot with two or less jobs. Then, since the time slot has size K , we would have idle time. Second, assume that there is a time slot with four or more jobs. In that case, the total processing time of those jobs is strictly larger than K . Thus, there exists a partition of the jobs $[3m]$ into sets of three such that each set has a total processing time of K . Define the sets $S_1, \dots, S_m \subseteq A$ accordingly.

Therefore, the given 3Partition instance is a yes-instance. $\square$

Definition 5.3 (Partition). *Given a set $A = \{a_1, \dots, a_m\}$ of m integers with $\sum_{a \in A} a = 2K$. Is there a subset $S \subseteq A$ such that $\sum_{a \in S} a = K$?*

Theorem 5.5. *The problem $1|r_j|WC_{\max}$ with at least two weight classes is weakly NP-hard.*

Proof. We reduce the **Partition** problem to the following decision problem: Does there exist a schedule σ for problem $1|r_j|WC_{\max}$ such that $\max_j(w_j C_j)(\sigma) \leq y$ for given y . Given an instance I of **Partition**, we construct an instance for $1|r_j|WC_{\max}$ as follows: For each $i \in [m]$, we create job i with release date $r_i = 0$, processing time $p_i = a_i$ and weight $w_i = 1$. Additionally, we create job $m+1$ with release date $r_{m+1} = K$, processing time $p_{m+1} = 0$ and weight $w_{m+1} = 2$. Set $y = 2K$.

To obtain the result for more than two weight classes, we can add additional jobs with release date $r_i = 0$, processing time $p_i = 0$ and arbitrary weight w_i . The described instance of the problem $1|r_j|WC_{\max}$ can be computed in polynomial time.

We show that the **Partition** instance is a yes-instance if and only if the constructed $1|r_j|WC_{\max}$ instance is a yes-instance.

$\Rightarrow$: Assume that the given **Partition** instance is a yes-instance. Then there exists a partition of A into S_1 and S_2 such that $\sum_{a \in S_1} a = \sum_{a \in S_2} a = K$. W.l.o.g. assume that elements $a_1, \dots, a_{m'} \in S_1$ and $a_{m'+1}, \dots, a_m \in S_2$. We now schedule the jobs $1, \dots, m'$ in the time interval $[0, K]$ in arbitrary order. By definition of the jobs, we know $\sum_{j=1}^{m'} p_j = \sum_{a \in S_1} a = K$. Thus all jobs can be processed before time K . Next, schedule job $m+1$ at time K . It also finishes at time K . After that schedule the jobs $m'+1, \dots, m$ in arbitrary order. Since $\sum_{j=m'+1}^m p_j = \sum_{a \in S_2} a = K$, we know that all jobs have been processed at time $2K$. With the definition of the jobs $1, \dots, m$, we get $w_j C_j \leq 2K = y$ for all $j \in [m]$ and with the definition of the job $m+1$, we get $w_{m+1} C_{m+1} = 2K = y$.

When considering more than two weight classes, schedule all jobs with processing time 0 at time 0. For those jobs, we get $w_j C_j = w_j \cdot 0 = 0 \leq 2K = y$.

Thus, the constructed $1|r_j|WC_{\max}$ instance is a yes-instance.

$\Leftarrow$: Assume that the constructed $1|r_j|WC_{\max}$ instance is a yes-instance. This means that we have a schedule σ with $\max_j(w_j C_j)(\sigma) \leq y$. In this schedule, job $m+1$ has to be scheduled at time K . It cannot be processed earlier due to its release date and it cannot be processed later because then it

finishes at time $C_{m+1} > K$ which implies $w_{m+1}C_{m+1} > 2K = y$ which is a contradiction to $\max_j(w_jC_j)(\sigma) \leq y$. W.l.o.g. assume that jobs $1, \dots, m'$ are scheduled before time K and $m' + 1, \dots, m$ are scheduled after time K . Now, partition A into two subsets S_1, S_2 such that $a_1, \dots, a_{m'} \in S_1$ and $a_{m'+1}, \dots, a_m \in S_2$. Assume for contradiction $\sum_{a \in S_1} a \neq \sum_{a \in S_2} a$.

Case 1: $\sum_{a \in S_1} a > \sum_{a \in S_2} a$.

Then $\sum_{a \in S_1} a + \sum_{a \in S_2} a = 2K$ implies $\sum_{a \in S_1} a > K$. Since all jobs $1, \dots, m'$ are processed before job $m + 1$ and $\sum_{j=1}^{m'} p_j = \sum_{a \in S_1} a > K$, job $m + 1$ cannot start at time K , which is a contradiction.

Case 2: $\sum_{a \in S_1} a < \sum_{a \in S_2} a$.

Then $\sum_{a \in S_1} a + \sum_{a \in S_2} a = 2K$ implies $\sum_{a \in S_2} a > K$. Since all jobs $m' + 1, \dots, m$ are processed after job $m + 1$, job $m + 1$ finishes at time K and $\sum_{j=1}^{m'} p_j = \sum_{a \in S_1} a > K$ we get that there exists a job k with weight $w_k = 1$ and completion time $C_k > 2K$. Thus, $\max_j(w_jC_j)(\sigma) \geq w_kC_k > 2K = y$ which is a contradiction to the assumption that the constructed $1|r_j|WC_{\max}$ instance is a yes-instance.

Therefore, $\sum_{a \in S_1} a = \sum_{a \in S_2} a$ and the given **Partition** instance is a yes-instance. $\square$

PTAS

We present a polynomial-time approximation scheme (PTAS) for the problem $P|r_j|WC_{\max}$. It works analogously to the PTAS for the problem $1|r_j|\sum w_jC_j$ of Afrati et al. [3]. A difference to the previous PTAS is that we insert the small jobs according to the largest weight first rule (LW).

For a single machine, if there are no release dates, it is easy to see that it is optimal to schedule the job with largest weight at any time. If there are release dates but jobs can be preempted, we again get an optimal solution in this way. Our PTAS relies on analogous versions of the lemmas 2.1-4 and 4.1-4 of Afrati et al. [3], and we mostly follow the line of their proof. Some of our exposition is adapted from Chekuri and Khanna [47]; we fix some minor inaccuracies and expand on some explanations in what follows.

Theorem 5.6. *There exists a PTAS for the problem $P|r_j|WC_{\max}$.*

We begin by simplifying the input in several steps. Each transformation may increase the objective function value by $1 + O(\varepsilon)$. We write that a transformation produces $1 + O(\varepsilon)$ loss (in some cases it is just $1 + \varepsilon$). We can perform a constant

number of such transformations while staying within $1 + O(\varepsilon)$ of the original optimum. Let $\varepsilon < 1/2$.

Lemma 5.1 (Lemma 2.1 in [3]). *With $1 + \varepsilon$ loss, we can assume that all processing times and release dates are integer powers of $1 + \varepsilon$.*

Proof. Multiplying each release date and processing time by $1 + \varepsilon$ increases the objective by $1 + \varepsilon$ (we just scale time). Rounding down to the next lower integer power of $1 + \varepsilon$ does not increase the objective function. After these two steps, no number is smaller than it was in the original input: suppose $x \in ((1 + \varepsilon)^{k-1}, (1 + \varepsilon)^k]$ for some integer k . Then $(1 + \varepsilon)x \in ((1 + \varepsilon)^k, (1 + \varepsilon)^{k+1}]$ and the final value therefore becomes $(1 + \varepsilon)^k \geq x$. $\square$

We can thus assume all jobs arrive at integer powers of $1 + \varepsilon$. We will consider intervals of the form $I_k = [(1 + \varepsilon)^k, (1 + \varepsilon)^{k+1})$ where k is integer (possibly negative). With I_k we denote the interval itself as well as the size of the interval. We have

$$I_k = \varepsilon(1 + \varepsilon)^k$$

and we define $s := \lceil \log_{1+\varepsilon}(1 + \frac{1}{\varepsilon}) \rceil \in O(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon})$ (using $s = \lceil \frac{\log(1+1/\varepsilon)}{\log(1+\varepsilon)} \rceil$ and $\log(1 + \varepsilon) \geq \frac{\varepsilon}{1+\varepsilon}$).

In the interior of an interval I_k , no jobs arrive. For the next transformations, we fix some optimal solution.

Lemma 5.2 (Lemma 2.2 in [3]). *With $1 + \varepsilon$ loss, we can enforce $r_j \geq \varepsilon p_j$ for each job j .*

Proof. We multiply every completion time in the optimal schedule by $1 + \varepsilon$ and increase starting times to match. A job that previously completed at time $t \geq p_j$ now completes at time $(1 + \varepsilon)t$, so it starts not before time $\varepsilon t \geq \varepsilon p_j$. This means that its release date can be increased to εp_j if it was smaller without making this new schedule infeasible. We still have a schedule that is within $1 + \varepsilon$ of the optimal schedule. $\square$

Corollary 5.1. *For any k , each job j released in I_k has size at most I_k/ε^2 .*

Proof. For such a job we have $(1 + \varepsilon)^k \geq r_j \geq \varepsilon p_j$, so $I_k = \varepsilon(1 + \varepsilon)^k \geq \varepsilon^2 p_j$. $\square$

Lemma 5.3. *For any k , the total size of the intervals $I_{k+1}, \dots, I_{k+s}$ is more than I_k/ε^2 and the total size of $I_k, \dots, I_{k+s-1}$ is less than I_k/ε^3 .*

Proof. We have

$$\sum_{j=k+1}^{k+s} \varepsilon(1+\varepsilon)^j = \varepsilon(1+\varepsilon)^{k+1} \frac{(1+\varepsilon)^s - 1}{1+\varepsilon-1} \geq (1+\varepsilon)^{k+1} \left(1 + \frac{1}{\varepsilon} - 1\right) = \frac{1}{\varepsilon}(1+\varepsilon)^{k+1} > \frac{I_k}{\varepsilon^2}.$$

Because $(1+\varepsilon)^s < (1+\varepsilon)(1+\frac{1}{\varepsilon}) = 2+\varepsilon+\frac{1}{\varepsilon}$ and $1+\varepsilon+\frac{1}{\varepsilon} < \frac{1}{\varepsilon^2}$, we similarly get

$$\sum_{j=k}^{k+s-1} \varepsilon(1+\varepsilon)^j = \varepsilon(1+\varepsilon)^k \frac{(1+\varepsilon)^s - 1}{1+\varepsilon-1} \leq (1+\varepsilon)^k \left(1 + \frac{1}{\varepsilon} + \varepsilon\right) < \frac{\varepsilon(1+\varepsilon)^k}{\varepsilon^3} = \frac{I_k}{\varepsilon^3}.$$

□

Lemma 5.4 (Lemma 2.3 in [3]). *For any k , a job that starts in I_k finishes at the latest in I_{k+s} .*

Proof. This follows immediately from Corollary 5.1 and Lemma 5.3. □

A job is *small* if its size is less than ε^2 times the size of the interval in which it arrives.

Lemma 5.5 (Lemma 2.4 and 4.1 in [3]). *With $1 + O(\varepsilon)$ loss, each small job is fully contained in an interval, and in each interval the small jobs in it are executed contiguously. Two small jobs j and k for which $r_j \leq r_k$ and $w_j > w_k$ are run in the order j before k .*

Proof. Increase the size of each interval by a factor of $1 + \varepsilon^2$. Since for each interval and each machine there is at most one job in a fixed schedule that starts in one interval and finishes in the next interval, then if such a job has size at most $\varepsilon^2 I_k$, we can run it in the additional $\varepsilon^2 I_k$ space that was generated by the increase of the size of the intervals.

For each machine, consider the first small job j in I_k . By shifting all later small jobs on this machine in this interval so that they run immediately after j , the completion times of some large jobs may increase. However, all large jobs that previously completed in I_k still do so, so their completion time is bounded by $(1 + \varepsilon^2)(1 + \varepsilon)^{k+1}$ whereas initially it was more than $(1 + \varepsilon)^k$. This means that the value of the objective function increases by at most a factor of $1 + \varepsilon$ as the completion time of small jobs only improves. Hence, without loss of generality, the small jobs are executed contiguously within each interval.

Consider a fixed optimal schedule in which all release dates are of the form $(1 + \varepsilon)^k$. Let s_{ki} be the (contiguous) time allocated to this schedule for running small jobs in interval I_k on machine i . We allocate the small jobs to intervals in

order of increasing k , where for each k we consider all the machines one by one before moving to $k + 1$. To each interval I_k , we assign the small jobs that have already been released but that have not been assigned to previous intervals in order of decreasing weight, stopping on machine i when the total assigned size to interval I_k is at least s_{ki} (and less than $s_{ki} + \varepsilon^2 I_k$), or when we have assigned all available small jobs. By stretching each I_k by a factor of $1 + \varepsilon^2$, all small jobs can be completed in their assigned intervals. Compared to the optimal solution, the only change to the overall order of the small jobs is that we may sometimes run heavier jobs earlier in this way. Formally, for the k th interval treated in this way, the highest weight assigned so far is at least the highest weight of the first k intervals in the optimal solution. Within an interval, some small jobs may complete earlier in the optimal solution than in our solution, but not more than a factor of $1 + \varepsilon$ earlier.

For each interval I_k , the total size of small jobs that we assign until this interval is not less than the total size assigned so far in the optimal solution. The intervals have been stretched by a factor of $1 + \varepsilon^2$, and small jobs complete at most a factor of $1 + \varepsilon$ later in each interval. Hence the weighted makespan of the small jobs is at most $(1 + \varepsilon^2)(1 + \varepsilon) \leq 1 + 2\varepsilon$ of its value in the optimal schedule (for $\varepsilon < 1/2$). This also holds for the large jobs because their order has remained unchanged and the only change is that the intervals have been stretched. $\square$

As a result of Lemma 5.5 we can order all small jobs released at $(1 + \varepsilon)^k$ according to their weight and consider them for scheduling only in that order. Let T_k and H_k denote the small and large jobs released at $(1 + \varepsilon)^k$ (T for tiny and H for huge). Let $p(S)$ denote the sum of the processing times of the jobs in set S . The next lemma says that any input instance I can be modified with $1 + \varepsilon$ loss to an instance I' so that the total size of the small and large jobs released at any release date $(1 + \varepsilon)^k$ is $O(mI_k)$. This lemma is important for the implementation of our dynamic programming algorithm since it allows us to represent information about unfinished jobs compactly.

Lemma 5.6 (Lemma 11.18 in [47]). *With $1 + O(\varepsilon)$ loss, we can transform our problem instance I to an instance I' such that the following conditions hold.*

- $p(T'_k) \leq (1 + \varepsilon)mI_k$ for all k .
- The number of distinct job sizes in H'_k is at most $\lfloor 1 + 4\log_{1+\varepsilon} \frac{1}{\varepsilon} \rfloor \in O(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon})$.
- The number of jobs of each distinct size in H'_k is at most $\frac{m}{\varepsilon^2}$.

- $p(H'_k \cup T'_k) \leq \frac{1}{\varepsilon^4} m I_k$ for all k .

Proof. Consider an input instance I in which each release date is of the form $(1 + \varepsilon)^k$. For each integer value of k , starting with the smallest value of k at which small jobs arrive and ending when all small jobs have been assigned to intervals, we do the following. The processing time available in interval I_k is $m I_k$. Order the small jobs in T_k by decreasing weights and pick jobs according to this order until the processing time of jobs picked just exceeds $(1 + \varepsilon) m I_k$. Picking jobs according to this order is justified by Lemma 5.5. The remaining jobs, which are released at $(1 + \varepsilon)^k$ but cannot be processed in I_k , are moved to the next release date $(1 + \varepsilon)^{k+1}$, that is, to the set T_{k+1} . (Jobs may be moved more than once.)

For each job j in H_k , Lemma 5.2 yields $(1 + \varepsilon)^k \geq \varepsilon p_j$. On the other hand, since j is large we get $p_j \geq \varepsilon^2 I_k = \varepsilon^3 (1 + \varepsilon)^k$. Since all job sizes are powers of $1 + \varepsilon$, the number of distinct job sizes in H_k is as claimed. For each job size we order the jobs of this size by non-increasing weights. The number of large jobs of each size class that can be executed in the current interval is limited to $\frac{m}{\varepsilon^2}$. We can therefore treat each size class analogously to the small jobs, moving superfluous large jobs of each size to the next interval. For each size class larger than I_k , it is sufficient to keep m jobs. By Corollary 5.1, each job released at time $(1 + \varepsilon)^k$ has size at most I_k / ε^2 . Hence there are at most $\lceil 2 \log_{1+\varepsilon} \frac{1}{\varepsilon} \rceil$ such classes. For each smaller size class, the total size of jobs is at most $2m I_k$, and there are at most $\lceil 2 \log_{1+\varepsilon} \frac{1}{\varepsilon} \rceil$ such classes since jobs of size at most $\varepsilon^2 I_k$ are not large. This implies $p(H'_k \cup T'_k) \leq (2 + 2 \log_{1+\varepsilon} \frac{1}{\varepsilon}) \cdot 2m I_k + (1 + 2 \log_{1+\varepsilon} \frac{1}{\varepsilon}) m I_k / \varepsilon^2 = (\frac{1}{\varepsilon^2} (1 + 2 \log_{1+\varepsilon} \frac{1}{\varepsilon}) + 4 + 4 \log_{1+\varepsilon} \frac{1}{\varepsilon}) m I_k \leq \frac{1}{\varepsilon^4} m I_k$, for $\varepsilon < 1/4$. $\square$

As in Afrati et al. [3] we now decompose the time horizon into a sequence of *blocks*. A block is a set of $s = \lceil \log_{1+\varepsilon}(1 + \frac{1}{\varepsilon}) \rceil$ consecutive intervals. Let $B_0, B_1, \dots, B_\ell$ be the partition of the time interval $[\min_j r_j, D)$ into blocks where D is an upper bound on the schedule makespan (we can bound D crudely by $(\sum_j p_j + \max_j r_j)$). Our goal is to do dynamic programming with blocks as units. There is interaction between blocks since jobs from an earlier block can be partially scheduled into the current block. However, by the choice of the block size and Lemma 5.4, no job has parts of it scheduled in three or more different blocks. In other words, jobs that start in B_i finish either in B_i or B_{i+1} . A *frontier* describes the potential ways that jobs in one block finish in the next. An incoming frontier for a block B_i specifies the times at which the remaining unfinished jobs from B_{i-1} finish.

Lemma 5.7 (Lemma 4.3 in [3]). *With $1 + \varepsilon$ loss, the number of possible frontiers for block B_i is at most $(m + 1)^{s/\varepsilon}$.*

Proof. By Lemma 5.5 we only need to consider large jobs. Fix an optimal schedule and consider a block B_i . A large job j continuing from the preceding block finishes in one of the s intervals of block B_i which we denote by $I_{k(j)}$. We can round up C_j to $C'_j = (1 + \varepsilon)^{k(j)} + r \cdot \varepsilon I_{k(j)}$ for some integer $0 \leq r \leq \frac{1}{\varepsilon} - 1$. This discretizes the time inside interval $I_{k(j)}$ and will increase the schedule value by only a $1 + \varepsilon$ factor. Thus, we can restrict the completion times of unfinished jobs from the preceding block to s/ε discrete time instants. Each machine realizes one of these possibilities, and the machines are identical and interchangeable. A frontier can thus be described as a tuple $(m_1, \dots, m_{s/\varepsilon})$ where m_i is the number of machines with unfinished jobs from the preceding block finishing at the i th discrete time instant. This proves the lemma. $\square$

Lemma 5.8 (Lemma 11.20 in [47]). *With $1 + O(\varepsilon)$ loss, each job finishes within 8 blocks after it is released.*

Proof. Consider some fixed optimal schedule. For each block B_i , let U_i denote the set of jobs released in B_i which are not finished in the optimal schedule by the end of B_{i+7} . For each i , we alter the schedule by scheduling all the jobs in U_i in B_{i+8} . Let I_x be the smallest interval in block B_{i+8} . Using Lemma 5.6 the total volume of jobs in any interval I_k is at most $\frac{1}{\varepsilon^4} m I_k$, so the total volume released in B_i is at most $\frac{1}{\varepsilon^4} m B_i$. Since $B_{i+1} \geq (1 + \frac{1}{\varepsilon}) B_i > B_i/\varepsilon$ for all i , this volume is at most $\varepsilon^4 m B_{i+8} \leq \varepsilon m I_x$ by Lemma 5.3. Using Corollary 5.1, for every job $j \in U_i$, we have $p_j \leq I_y/\varepsilon^2$ where I_y is the first interval in B_{i+1} . We conclude that $p_j \leq \varepsilon^4 I_x$.

To schedule jobs of U_i in B_{i+8} we group jobs of U_i into units each with volume between εI_x and $(\varepsilon + \varepsilon^4) I_x$. From the bound on $p(U_i)$ the number of units is at most m . We assign each unit to an exclusive machine and schedule the unit on the machine as soon as the crossing job on that machine from B_{i+7} finishes. We shift the jobs on the machine to accommodate the extra volume. Clearly, the completion times of jobs in U_i have only decreased. It is easy to see that expanding intervals by a $(1 + O(\varepsilon))$ -factor can accommodate the increase in the completion times of jobs in B_i for all i . The new schedule satisfies the desired property. $\square$

Number of blocks. Given Lemma 5.8, a very simple upper bound for the number

of blocks that have to be considered in our dynamic program is $8s$.

Large jobs. Consider the large jobs released in interval I_k . By Lemma 5.6, there are $O(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon})$ size classes in H_k . Within each size class, we sort and schedule the jobs in decreasing order of weight, which is optimal. Let U_{ki} be the set of large jobs released in I_k that remain unscheduled after block B_i . Such a set can be specified by indicating for each size class the smallest indexed job that remains to be scheduled. The index of that job is between 1 and $m/\varepsilon^2 + 1$ (if no job remains unscheduled), and we need to specify $O(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon})$ indices. The number of possibilities for one set U_{ki} is therefore (m/ε^2) . By Lemma 5.8, at most $O(s)$ sets U_{ki} are nonempty in any block, and we know $s \in O(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon})$. The number of options for all U_{ki} sets together for a single block is therefore $(m/\varepsilon^2)^{O(\frac{1}{\varepsilon^2} \log^2 \frac{1}{\varepsilon})}$.

Small jobs. The small jobs arriving at any given release date are executed in order of decreasing weight. For each interval I_k , we partition this *ordered* list into $\lceil (1 + \varepsilon)m/\varepsilon^2 \rceil$ pieces by repeatedly removing a prefix of jobs with total size at least $\varepsilon^2 I_k$ and at most $2\varepsilon^2 I_k$. Call the i th set constructed out of T_k in this way $J_i(k)$. We show using Lemma 5.9 below that time stretching lets us schedule an integral number of these pieces in each block.

Lemma 5.9 (Lemma 11.21 in [47]). *For each set $J_i(k)$, with $1 + 3\varepsilon$ loss, all jobs in it are scheduled in the same interval.*

Proof. Consider an optimal schedule that does not satisfy the properties of the lemma. Fix a release date $(1 + \varepsilon)^k$ for which the conditions of the lemma are violated. Let i be an index such that some but not all jobs from $J_i(k)$ are scheduled in an interval I_y . We assign all jobs in $J_i(k)$ to I_y , on the same machine as and immediately following the last job of $J_i(k)$ that is already scheduled in I_y . By Lemma 5.5, for each $k \leq y$ there is at most one index i such that $J_i(k)$ is partially scheduled in I_y . The total size additionally assigned to I_y is therefore at most

$$\sum_{k \leq y} 2\varepsilon^2 I_k < 2\varepsilon^3 (1 + \varepsilon)^y \sum_{k=0}^{\infty} \left(\frac{1}{1 + \varepsilon} \right)^k = 2\varepsilon^2 I_y \frac{1}{1 - \frac{1}{1 + \varepsilon}} = 2(\varepsilon + \varepsilon^2) I_y. \quad (5.2)$$

For each interval I_y , we stretch I_y by a factor of $1 + 3\varepsilon$. This is sufficient to place all these jobs (even if they are all scheduled on the same machine). All other jobs including all large jobs are scheduled in the same order and on the same machines as before, and their completion times increase by a factor of at most $1 + 3\varepsilon$. $\square$

This means that instead of scheduling small jobs we now only need to schedule pieces $J_i(k)$. Within each piece, we sort and schedule the small jobs in decreasing order of weight, which is optimal. Let V_{ki} be the set of pieces released in I_k that remain unscheduled after block B_i . Such a set can be specified by indicating the smallest indexed piece that remains to be scheduled. The index of that piece is between 1 and $(1 + \varepsilon)m/\varepsilon^2 + 2$ (if no job remains unscheduled), and we need to specify $O(s) = O(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon})$ indices since pieces from at most 8 blocks (or $8s$ intervals) remain unscheduled after B_i by Lemma 5.8. The number of options for all V_{ki} sets together for a single block is therefore $((1 + \varepsilon)m/\varepsilon^2 + 1)^{O(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon})}$.

Dynamic program. Let $\mathcal{F}$ be the possible set of frontiers between blocks, where each frontier is a multiset. By Lemma 5.7, $|\mathcal{F}| = (m + 1)^{s/\varepsilon}$. The dynamic programming table entry $O(i, F, U)$ stores the minimum of the maximum weighted completion time ($WC_{\max}$) achievable by starting the set U of jobs before the end of block B_i while leaving a frontier of $F \in \mathcal{F}$ for block B_{i+1} , considering only the completion times of jobs in U for the determination of the maximum. (Here and below, infeasible inputs like incompatible F and U get a value of ∞ .) Given all the table entries for some i , the values for $i + 1$ can be computed as follows. Let $W(i, F_1, F_2, V)$ be the minimum of the maximum weighted completion time ($WC_{\max}$) achievable by scheduling the set of jobs V in block B_i , with F_1 as the incoming frontier from block B_{i-1} and F_2 the outgoing frontier to block B_{i+1} . We obtain the following equation:

$$O(i + 1, F, U) = \min_{F' \in \mathcal{F}, V \subset U} (\max(O(i, F', V), W(i + 1, F', F, U - V)))$$

We have $O(1, F, U) = W(1, 0, F, U)$. In total, we need to calculate at most $8s((1 + \varepsilon)(m + 1)/\varepsilon^2 + 1)^{O(\frac{1}{\varepsilon^2} \log^2 \frac{1}{\varepsilon})}$ values $W(i, F_1, F_2, V)$. Since determining such a value is an NP-hard problem, we present a $1 + \varepsilon$ decision procedure which outputs a schedule that is within $1 + \varepsilon$ of $W(i, F_1, F_2, V)$ and shifts the frontier F_2 by at most a factor of $1 + \varepsilon$. This is sufficient to calculate a $(1 + O(\varepsilon))$ -approximation of the solution to the dynamic program.

Given a block B_i and a set of jobs V to be scheduled in it, we reclassify the jobs as follows. Let I_x be the smallest interval in B_i . A job j is big if $p_j \geq \varepsilon I_x$, otherwise we call it tiny. We will enumerate schedules of the big jobs, restricting ourselves to schedules in which in each interval I_k big jobs start only at times of the form $(1 + \varepsilon)^x + i\varepsilon^2 I_k$. A schedule is specified by the size and the starting times of the big jobs; if there are several jobs of some fixed size available at some starting time, we always schedule the one with larger weight first. In each block,

there are $s/\varepsilon^2 \leq 1/\varepsilon^4$ allowed starting times and at most $\frac{4}{\varepsilon} \log \frac{1}{\varepsilon}$ possible job sizes of big jobs (they are between εI_x and $(1 + \frac{1}{\varepsilon})I_x/\varepsilon^2$, and they are powers of $1 + \varepsilon$). For each machine, this gives at most $N = (\frac{4}{\varepsilon} \log \frac{1}{\varepsilon})^{1/\varepsilon^4}$ possible schedules. Hence there are at most $(m + 1)^N$ schedules for the block B_i and the set of big jobs in V . Out of these we consider only the feasible schedules for the big jobs (that is, no two big jobs overlap, etc.) that are compatible with the incoming and outgoing frontiers F_1 and F_2 .

For each remaining schedule for all the machines, we insert the tiny jobs (which include all the small jobs pieces of the form $J_i(k)$) greedily. First, we increase the size of each interval by a factor of $1 + \varepsilon$ to ensure that each tiny job is fully contained in an interval and in each interval the tiny jobs in it are executed contiguously; the proof that this loses only a factor of $1 + O(\varepsilon)$ is completely analogous to the proof of Lemma 5.5. This means on each machine, we only need to consider schedules for the big jobs in which there is at most one period of idle time that is longer than $\varepsilon^2 I_k$ in each interval I_k . Given such a schedule, we add another εI_k to the end of the longest period of idle time in I_k (if any) for each I_k . This increases the weighted makespan of the large jobs on each machine by at most a(nother) factor of $1 + \varepsilon$. We now fill up these intervals using the tiny jobs exactly as in the proof of Lemma 5.5, filling up all machines in I_k before moving to the next interval and making sure that no job is scheduled before it is released. To do this, the weight of a piece $J_i(k)$ is considered to be the weight of the heaviest (small) job in it; recall that such pieces have size at most $2\varepsilon^2 I_k$, so this distorts the calculations by at most a factor of $1 + 2\varepsilon^2$. Recall further that each small job in a piece $J_i(k)$ is released at the same time. Because each period of idle time has size at least εI_k by the above, each such period will receive at least one job. Inserting the tiny jobs like this takes time linear in the number of tiny jobs.

Among all the schedules constructed like this which schedule the tiny jobs in the set V completely in B_i , we pick the one with the minimum weighted makespan. The incoming jobs into a block (that form the frontier) and all previous jobs are considered to have weight 0 for these calculations, as their weights are already considered in $O(i, F', V)$ in the dynamic program.

For fixed values i, F_1, F_2, V , consider the schedule σ that minimizes the weighted makespan. At some point our procedure above will consider the schedule σ , possibly with some large jobs completing a factor of $1 + O(\varepsilon)$ later as described above. Compared to σ , as in the proof of Lemma 5.5, each interval I_k receives at least the same total size of the tiny jobs assigned to I_k , and again the highest

weight of tiny jobs assigned up to and including I_k is at least as high as in σ for each I_k in B_i . This completes the proof that our procedure produces a $(1 + O(\varepsilon))$ -approximation to the dynamic program.

CHAPTER 6

Discussion and Open Problems

In this thesis, we explored a variety of topics in the area of offline and online scheduling. These problems are central to algorithm design and resource optimization with wide-ranging applications. We now summarize the most significant results of our work and pose several open questions that may inspire further research in these areas.

The following table summarizes the provided lower bounds in this thesis.

Problem	Lower Bound	Section
$1 \mid r_j, \text{online, k-restart} \mid \sum C_j$	2	Section 3.4
$1 \mid r_j, \text{online, restart, } p_j = 1 \mid \sum w_j C_j$	1.09725	Section 4.2
$1 \mid r_j, \text{online, 3-restart, } p_j = 1 \mid \sum w_j C_j$	1.3247	Section 4.4
$1 \mid r_j, \text{online, restart} \mid WC_{\max}$	1.4656	Section 5.2
$1 \mid r_j, \text{online, restart, } p_j = 1 \mid WC_{\max}$	1.2344	Section 5.2

Furthermore, next table summarizes the algorithmic results in this thesis.

Problem	Algorithm	Competitive Ratio	Section
$1 \mid r_j, \text{online, restart} \mid \sum C_j$	IRSPT	1.4568	Section 3.5
$1 \mid r_j, \text{online, } p_j = p, \text{restart} \mid \sum w_j C_j$	LW-3R	1.3247	Section 4.4
$1 \mid r_j, \text{online, restart, } p_j = p \mid WC_{\max}$	LLW	1.3098	Section 5.4

We began with the objective of minimizing the total completion time, as presented in Chapter 3, we considered the online problem of minimizing the total completion time on a single machine using restarts. We presented a simple online algorithm that achieves a competitive ratio of 1.4568. The algorithm schedules jobs in

order of increasing size and considers restarts. While our algorithm achieves a tight bound, obtaining a better competitive ratio seems non-trivial without incorporating the number of jobs that meet the interruption condition into the decision rule. Understanding whether such an extension can improve the competitive ratio is an interesting open question for future research.

In Chapter 4, we presented the first positive results demonstrating how restarts can be used effectively to minimize the total weighted completion time on a single machine in the case where all jobs have identical processing times. Our main contribution in this work is a deterministic online algorithm with a competitive ratio better than 1.325, achieved through a designed structure of decision intervals. The algorithm requires at most three restarts over the entire schedule, which significantly simplifies the analysis by reducing the number of possible scheduling scenarios. However, achieving a better competitive ratio would require allowing more restarts and developing new analytical techniques.

Finally, we studied the problem of minimizing the weighted makespan. We presented two new lower bounds on the competitive ratio for deterministic algorithms with restarts, one in the general setting and another in the case where all jobs have identical processing times. The main result in Chapter 5 is a deterministic online algorithm that achieves a competitive ratio better than 1.3098 for the case where all jobs have identical processing times. This leaves a small gap to our lower bound for further improvement. Closing this gap appears to be nontrivial and is an interesting topic for future research. Additionally, we addressed a gap in the literature by proving that this problem remains NP-hard even in the single-machine setting and we developed a polynomial-time approximation scheme (PTAS) for parallel machines and jobs with release dates.

References

- [1] Anders Aamand, Mikkel Abrahamsen, Lorenzo Beretta, and Linda Kleist. Online sorting and translational packing of convex polygons. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 1806–1833. SIAM, 2023.
- [2] Mikkel Abrahamsen, Ioana O. Bercea, Lorenzo Beretta, Jonas Klausen, and László Kozma. Online sorting and online TSP: randomized, stochastic, and high-dimensional. In Timothy Chan, Johannes Fischer, John Iacono, and Grzegorz Herman, editors, *32nd Annual European Symposium on Algorithms, ESA 2024, September 2-4, 2024, Royal Holloway, London, United Kingdom*, volume 308 of *LIPICs*, pages 5:1–5:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.
- [3] Foto N. Afrati, Evripidis Bampis, Chandra Chekuri, David R. Karger, Claire Kenyon, Sanjeev Khanna, Ioannis Milis, Maurice Queyranne, Martin Skutella, Clifford Stein, and Maxim Sviridenko. Approximation schemes for minimizing average weighted completion time with release dates. In *FOCS*, pages 32–44. IEEE Computer Society, 1999.
- [4] Sara Ahmadian, Hossein Esfandiari, Vahab Mirrokni, and Binghui Peng. Robust load balancing with machine learned advice. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 20–34. SIAM, 2022.
- [5] Miklos Ajtai, James Aspnes, Moni Naor, Yuval Rabani, Leonard J Schulman, and Orli Waarts. Fairness in scheduling. *Journal of Algorithms*, 29(2):306–357, 1998.
- [6] Saeed Alaei. Bayesian combinatorial auctions: Expanding single buyer mechanisms to many buyers. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 512–521. IEEE Computer Society, 2011.

- [7] Susanne Albers. Online scheduling. In *Introduction to scheduling*, pages 71–98. CRC Press, 2009.
- [8] Susanne Albers and Sebastian Schubert. Tight bounds for online matching in bounded-degree graphs with vertex capacities. In Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman, editors, *30th Annual European Symposium on Algorithms, ESA 2022, September 5-9, 2022, Berlin/Potsdam, Germany*, volume 244 of *LIPICs*, pages 4:1–4:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.
- [9] Noga Alon, Yossi Azar, Gerhard J Woeginger, and Tal Yadid. Approximation schemes for scheduling on parallel machines. *Journal of Scheduling*, 1(1):55–66, 1998.
- [10] Aflatoun Amouzandeh, Paul Hesse, Sören Schmitt, and Rob van Stee. SIGACT news online algorithms column 44: 2025 in review. *SIGACT News*, 2025. To appear.
- [11] Aflatoun Amouzandeh, Klaus Jansen, Lis Pirotton, Rob van Stee, and Corinna Wambsganz. Minimizing the weighted makespan with restarts on a single machine. *arXiv preprint arXiv:2510.09589*, 2025.
- [12] Aflatoun Amouzandeh, Sören Schmitt, and Rob van Stee. SIGACT news online algorithms column 40: 2022 in review. *SIGACT News*, 54(1):90–104, 2023.
- [13] Aflatoun Amouzandeh, Sören Schmitt, and Rob van Stee. SIGACT news online algorithms column 41: 2023 in review. *SIGACT News*, 54(4):108–125, 2023.
- [14] Aflatoun Amouzandeh, Sören Schmitt, and Rob van Stee. SIGACT news online algorithms column 43: 2024 in review. *SIGACT News*, 55(4):94–107, 2024.
- [15] Aflatoun Amouzandeh and Rob van Stee. Improved online scheduling with restarts on a single machine. In Marcin Bienkowski and Matthias Englert, editors, *Approximation and Online Algorithms - 22nd International Workshop, WAOA 2024, Egham, UK, September 5-6, 2024, Proceedings*, volume 15269 of *Lecture Notes in Computer Science*, pages 16–30. Springer, 2024.
- [16] Aflatoun Amouzandeh and Rob van Stee. Minimizing the total weighted completion time with restarts on a single machine. 2025. Under review.

- [17] S. Anand, Naveen Garg, and Amit Kumar. Resource augmentation for weighted flow-time explained by dual fitting. In *SODA*, pages 1228–1241, 2012.
- [18] Edward J. Anderson and Chris N. Potts. On-line scheduling of a single machine to minimize total weighted completion time. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 548–557, New York / Philadelphia, 2002. ACM / SIAM.
- [19] Edward J. Anderson and Chris N. Potts. Online scheduling of a single machine to minimize total weighted completion time. *Math. Oper. Res.*, 29(3):686–697, 2004.
- [20] CJ Argue, Anupam Gupta, Marco Molinaro, and Sahil Singla. Robust secretary and prophet algorithms for packing integer programs. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1273–1297. SIAM, 2022.
- [21] Chen Avin, Marcin Bienkowski, Andreas Loukas, Maciej Pacut, and Stefan Schmid. Dynamic balanced graph partitioning. *SIAM J. Discret. Math.*, 34(3):1791–1812, 2020.
- [22] Yossi Azar, Chay Machluf, Boaz Patt-Shamir, and Noam Touitou. Competitive vertex recoloring. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPICs*, pages 13:1–13:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.
- [23] Yossi Azar, Debmalya Panigrahi, and Noam Touitou. Online graph algorithms with predictions. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 35–66. SIAM, 2022.
- [24] Yossi Azar and Noam Touitou. Beyond tree embeddings—a deterministic framework for network design with deadlines or delay. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1368–1379. IEEE, 2020.
- [25] Étienne Bamas, Marina Drygala, and Andreas Maggiori. An improved analysis of greedy for online steiner forest. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on*

- Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 3202–3229. SIAM, 2022.
- [26] Antar Bandyopadhyay and David Gamarnik. Counting without sampling: Asymptotics of the log-partition function for certain statistical physics models. *Random Structures & Algorithms*, 33(4):452–479, 2008.
- [27] Siddhartha Banerjee, Vasilis Gkatzelis, Artur Gorokh, and Billy Jin. Online nash social welfare maximization with predictions. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1–19. SIAM, 2022.
- [28] Amotz Bar-Noy, Rajeev Motwani, and Joseph Naor. The greedy algorithm is optimal for on-line edge coloring. *Information Processing Letters*, 44(5):251–253, 1992.
- [29] Imreh Bárány and Victor Grinberg. On some combinatorial questions in finite-dimensional spaces. *Linear algebra and its applications*, pages 1–9, 1981.
- [30] Hocine Belouadah, Marc E. Posner, and Chris N. Potts. Scheduling with release dates on a single machine to minimize total weighted completion time. *Discret. Appl. Math.*, 36(3):213–231, 1992.
- [31] Aaron Bernstein, Yann Disser, Martin Groß, and Sandra Himburg. General bounds for incremental maximization. *Math. Program.*, 191(2):953–979, 2022.
- [32] Dimitris Bertsimas, David B. Brown, and Constantine Caramanis. Theory and applications of robust optimization. *SIAM Review*, 53(3):464–501, 2011.
- [33] Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. Dynamic algorithms for packing-covering lps via multiplicative weight updates. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1–47. SIAM, 2023.
- [34] Marcin Bienkowski, Martin Böhm, Martin Koutecký, Thomas Rothvoß, Jirí Sgall, and Pavel Veselý. Improved analysis of online balanced clustering. In Jochen Könemann and Britta Peis, editors, *Approximation and Online Algorithms - 19th International Workshop, WAOA 2021, Lisbon, Portugal, September 6-10, 2021, Revised Selected Papers*, volume 12982 of *Lecture Notes in Computer Science*, pages 224–233. Springer, 2021.

- [35] Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královic, Richard Královic, and Tobias Mömke. On the advice complexity of online problems. In Yingfei Dong, Ding-Zhu Du, and Oscar H. Ibarra, editors, *Algorithms and Computation, 20th International Symposium, ISAAC 2009, Honolulu, Hawaii, USA, December 16-18, 2009. Proceedings*, volume 5878 of *Lecture Notes in Computer Science*, pages 331–340. Springer, 2009.
- [36] Bartłomiej Bosek, Yann Disser, Andreas Emil Feldmann, Jakub Pawlewicz, and Anna Zych-Pawlewicz. Recoloring interval graphs with limited recourse budget. In Susanne Albers, editor, *17th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2020, June 22-24, 2020, Tórshavn, Faroe Islands*, volume 162 of *LIPIcs*, pages 17:1–17:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
- [37] Bartłomiej Bosek and Anna Zych-Pawlewicz. Dynamic coloring of unit interval graphs with limited recourse budget. In Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman, editors, *30th Annual European Symposium on Algorithms, ESA 2022, September 5-9, 2022, Berlin/Potsdam, Germany*, volume 244 of *LIPIcs*, pages 25:1–25:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.
- [38] Domagoj Bradac, Anupam Gupta, Sahil Singla, and Goran Zuzic. Robust algorithms for the secretary problem. *arXiv preprint arXiv:1911.07352*, 2019.
- [39] Brian Brubach, Karthik Abinav Sankararaman, Aravind Srinivasan, and Pan Xu. Online stochastic matching: New algorithms and bounds. *Algorithmica*, 82(10):2737–2783, 2020.
- [40] Niv Buchbinder, Moran Feldman, Joseph Naor, and Roy Schwartz. Submodular maximization with cardinality constraints. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1433–1452. SIAM, 2014.
- [41] Niv Buchbinder, Joseph (Seffi) Naor, and David Wajc. Lossless online rounding for online bipartite matching (despite its impossibility). In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 2030–2068. SIAM, 2023.

- [42] Martin Bullinger and René Romen. Online coalition formation under random arrival or coalition dissolution. In Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman, editors, *31st Annual European Symposium on Algorithms, ESA 2023, September 4-6, 2023, Amsterdam, The Netherlands*, volume 274 of *LIPIcs*, pages 27:1–27:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.
- [43] Jivitej S. Chadha, Naveen Garg, Amit Kumar, and V. N. Muralidhara. A competitive algorithm for minimizing weighted flow time on unrelated machines with speed augmentation. In *STOC*, pages 679–684, 2009.
- [44] Xing Chai, Lingfa Lu, Wenhua Li, and Liqi Zhang. Best-possible online algorithms for single machine scheduling to minimize the maximum weighted completion time. *Asia Pac. J. Oper. Res.*, 35(6):1850048:1–1850048:11, 2018.
- [45] Xing Chai, Lingfa Lu, Wenhua Li, and Liqi Zhang. Best-possible online algorithms for single machine scheduling to minimize the maximum weighted completion time. *Asia-Pacific Journal of Operational Research*, 35(06):1850048, 2018.
- [46] Suresh Chand, Rodney Traub, and Reha Uzsoy. Single-machine scheduling with dynamic arrivals: Decomposition results and an improved algorithm. *Naval Research Logistics (NRL)*, 43(5):709–719, 1996.
- [47] Chandra Chekuri and Sanjeev Khanna. Approximation algorithms for minimizing average weighted completion time. In Joseph Y.-T. Leung, editor, *Handbook of Scheduling - Algorithms, Models, and Performance Analysis*. Chapman and Hall/CRC, 2004.
- [48] Chandra Chekuri, Rajeev Motwani, B. Natarajan, and Clifford Stein. Approximation techniques for average completion time scheduling. In Michael E. Saks, editor, *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, 5-7 January 1997, New Orleans, Louisiana, USA*, pages 609–618. ACM/SIAM, 1997.
- [49] Chandra Chekuri, Rajeev Motwani, B. Natarajan, and Clifford Stein. Approximation techniques for average completion time scheduling. *SIAM J. Comput.*, 31(1):146–166, 2001.

- [50] B. Chen and A. Vestjens. Scheduling on identical machines: How good is LPT in an on-line setting? *Operations Research Letters*, 21(4):165–19, 1997.
- [51] Ryder Chen, Jahanvi Khatkar, and Seeun William Umboh. Online weighted cardinality joint replenishment problem with delay. In *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.
- [52] Marek Chrobak, Wojciech Jawor, Jirí Sgall, and Tomáš Tichý. Online scheduling of equal-length jobs: Randomization and restarts help. *SIAM J. Comput.*, 36(6):1709–1728, 2007.
- [53] Chengbin Chu. Efficient heuristics to minimize total flow time with release dates. *Oper. Res. Lett.*, 12(5):321–330, 1992.
- [54] Ilan Reuven Cohen and Debmalya Panigrahi. A general framework for learning-augmented online allocation. *arXiv preprint arXiv:2305.18861*, 2023.
- [55] Ilan Reuven Cohen and Binghui Peng. Primal-dual schemes for online matching in bounded degree graphs. In Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman, editors, *31st Annual European Symposium on Algorithms, ESA 2023, September 4-6, 2023, Amsterdam, The Netherlands*, volume 274 of *LIPIcs*, pages 35:1–35:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.
- [56] José R. Correa and Andrés Cristi. A constant factor prophet inequality for online combinatorial auctions. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 686–697. ACM, 2023.
- [57] Andrés Cristi and Bruno Ziliotto. Prophet inequalities require only a constant number of samples. In Bojan Mohar, Igor Shinkar, and Ryan O’Donnell, editors, *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024*, pages 491–502. ACM, 2024.
- [58] Nico P Dellaert and M Teresa Melo. Heuristic procedures for a stochastic lot-sizing problem in make-to-order manufacturing. *Annals of Operations Research*, 59(1):227–258, 1995.

- [59] Jitender S. Deogun. On scheduling with ready times to minimize mean flow time. *Comput. J.*, 26(4):320–328, 1983.
- [60] Lindsey Deryckere and Seeun William Umboh. Online matching with set and concave delays. In Nicole Megow and Adam D. Smith, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2023, September 11-13, 2023, Atlanta, Georgia, USA*, volume 275 of *LIPICs*, pages 17:1–17:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.
- [61] M. I. Dessouky and Jitender S. Deogun. Sequencing jobs with unequal ready times to minimize mean flow time. *SIAM J. Comput.*, 10(1):192–202, 1981.
- [62] Nikhil R. Devanur, Kamal Jain, and Robert D. Kleinberg. Randomized primal-dual analysis of RANKING for online bipartite matching. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 101–107. SIAM, 2013.
- [63] Yann Disser, Max Klimm, Kevin Schewior, and David Weckbecker. Incremental maximization via continuization. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, July 10-14, 2023, Paderborn, Germany*, volume 261 of *LIPICs*, pages 47:1–47:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.
- [64] Paul Dütting, Thomas Kesselheim, and Brendan Lucier. An $o(\log \log m)$ prophet inequality for subadditive combinatorial auctions. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 306–317. IEEE, 2020.
- [65] Martin E. Dyer and Laurence A. Wolsey. Formulating the single machine sequencing problem with release dates as a mixed integer program. *Discret. Appl. Math.*, 26(2-3):255–270, 1990.
- [66] Leah Epstein and Rob van Stee. Lower bounds for on-line single-machine scheduling. In Jirí Sgall, Ales Pultr, and Petr Kolman, editors, *Mathematical Foundations of Computer Science 2001, 26th International Symposium, MFCS 2001 Mariánské Lázně, Czech Republic, August 27-31, 2001, Proceedings*, volume 2136 of *Lecture Notes in Computer Science*, pages 338–350. Springer, 2001.

- [67] Leah Epstein and Rob van Stee. Lower bounds for on-line single-machine scheduling. *Theor. Comput. Sci.*, 299(1-3):439–450, 2003.
- [68] Ekin Ergen. Online makespan scheduling under scenarios. In Anne Benoit, Haim Kaplan, Sebastian Wild, and Grzegorz Herman, editors, *33rd Annual European Symposium on Algorithms, ESA 2025, September 15-17, 2025, Warsaw, Poland*, volume 351 of *LIPIcs*, pages 27:1–27:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025.
- [69] Tomer Ezra, Michal Feldman, Nick Gravin, and Zhihao Gavin Tang. "Who is next in line?" on the significance of knowing the arrival order in bayesian online settings. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 3759–3776. SIAM, 2023.
- [70] Ronald Fagin and John H Williams. A fair carpool scheduling algorithm. *IBM Journal of Research and development*, 27(2):133–139, 1983.
- [71] Jon Feldman, Aranyak Mehta, Vahab S. Mirrokni, and S. Muthukrishnan. Online stochastic matching: Beating $1-1/e$. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 117–126. IEEE Computer Society, 2009.
- [72] Michal Feldman, Federico Fusco, Simon Mauras, and Rebecca Reiffenhäuser. Truthful matching with online items and offline agents. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, July 10-14, 2023, Paderborn, Germany*, volume 261 of *LIPIcs*, pages 58:1–58:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.
- [73] Moran Feldman, Ola Svensson, and Rico Zenklusen. Online contention resolution schemes with applications to bayesian selection problems. *SIAM Journal on Computing*, 50(2):255–300, 2021.
- [74] Ruyan Fu, T. C. Edwin Cheng, Chi To Ng, and Jinjiang Yuan. Online scheduling on two parallel-batching machines with limited restarts to minimize the makespan. *Information Processing Letters*, 110(11):444–450, 2010.

- [75] Stanley P. Y. Fung, Feifeng Zheng, Wun-Tat Chan, Francis Y. L. Chin, Chung Keung Poon, and Prudence W. H. Wong. Improved on-line broadcast scheduling with deadlines. *J. Sched.*, 11(4):299–308, 2008.
- [76] Buddhima Gamlath, Michael Kapralov, Andreas Maggiori, Ola Svensson, and David Wajc. Online matching with general arrivals. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 26–37. IEEE Computer Society, 2019.
- [77] Amit Ganz, Pranav Nuti, and Roy Schwartz. A tight competitive ratio for online submodular welfare maximization. In Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman, editors, *31st Annual European Symposium on Algorithms, ESA 2023, September 4-6, 2023, Amsterdam, The Netherlands*, volume 274 of *LIPIcs*, pages 52:1–52:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.
- [78] Martin Gardner. *New mathematical diversions from Scientific American*, chapter 3. Simon and Schuster, 1966. Reprint of the original column published in February 1960 with additional comments.
- [79] M. R. Garey and David S. Johnson. Complexity results for multiprocessor scheduling under resource constraints. *SIAM Journal on Computing*, 4(4):397–411, 1975.
- [80] Pedro G. Gazmuri. Probabilistic analysis of a machine scheduling problem. *Math. Oper. Res.*, 10(2):328–339, 1985.
- [81] Michel X. Goemans, Maurice Queyranne, Andreas S. Schulz, Martin Skutella, and Yaoguang Wang. Single machine scheduling with release dates. *SIAM J. Discret. Math.*, 15(2):165–192, 2002.
- [82] Ronald L Graham. Bounds for certain multiprocessing anomalies. *Bell system technical journal*, 45(9):1563–1581, 1966.
- [83] Nick Gravin, Enze Sun, and Zhihao Gavin Tang. The cardinal complexity of online ordinal problems, 2023. FOCS.
- [84] Anupam Gupta, Vijaykrishna Gurunathan, Ravishankar Krishnaswamy, Amit Kumar, and Sahil Singla. Online discrepancy with recourse for vectors and graphs. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1356–1383. SIAM, 2022.

- [85] Anupam Gupta, Gregory Kehne, and Roie Levin. Random order online set cover is as easy as offline. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 1253–1264. IEEE, 2021.
- [86] Anupam Gupta, Gregory Kehne, and Roie Levin. Set covering with our eyes wide shut. In David P. Woodruff, editor, *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pages 4530–4553. SIAM, 2024.
- [87] Anupam Gupta, R Ravi, Kunal Talwar, and Seeun William Umboh. Last but not least: Online spanners for buy-at-bulk. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 589–599. SIAM, 2017.
- [88] Varun Gupta, Ravishankar Krishnaswamy, Sai Sandeep, and Janani Sundaresan. Look before, before you leap: Online vector load balancing with few reassignments. In Yael Tauman Kalai, editor, *14th Innovations in Theoretical Computer Science Conference, ITCS 2023, January 10-13, 2023, MIT, Cambridge, Massachusetts, USA*, volume 251 of *LIPIcs*, pages 65:1–65:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.
- [89] Bernhard Haeupler, Vahab S. Mirrokni, and Morteza Zadimoghaddam. Online stochastic weighted matching: Improved approximation algorithms. In Ning Chen, Edith Elkind, and Elias Koutsoupias, editors, *Internet and Network Economics - 7th International Workshop, WINE 2011, Singapore, December 11-14, 2011. Proceedings*, volume 7090 of *Lecture Notes in Computer Science*, pages 170–181. Springer, 2011.
- [90] Leslie A. Hall, Andreas S. Schulz, David B. Shmoys, and Joel Wein. Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Math. Oper. Res.*, 22(3):513–544, 1997.
- [91] Yijie Han. Deterministic sorting in $o(n \log \log n)$ time and linear space. *J. Algorithms*, 50(1):96–105, 2004.
- [92] Sarel Har-Peled, Elfarouk Harb, and Vasilis Livanos. Oracle-augmented prophet inequalities. In Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson, editors, *51st International Colloquium on Automata, Languages, and Programming, ICALP 2024, July 8-12, 2024, Tallinn, Estonia*, volume 297 of *LIPIcs*, pages 81:1–81:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

- [93] Dorit S. Hochbaum and David B. Shmoys. Using dual approximation algorithms for scheduling problems theoretical and practical results. *Journal of the ACM*, 34(1):144–162, 1987.
- [94] Martin Hoefer and Kevin Schewior. Threshold testing and semi-online prophet inequalities. In Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman, editors, *31st Annual European Symposium on Algorithms, ESA 2023, September 4-6, 2023, Amsterdam, The Netherlands*, volume 274 of *LIPICs*, pages 62:1–62:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.
- [95] Han Hoogeveen and Arjen P. A. Vestjens. Optimal on-line algorithms for single-machine scheduling. In William H. Cunningham, S. Thomas McCormick, and Maurice Queyranne, editors, *Integer Programming and Combinatorial Optimization, 5th International IPCO Conference, Vancouver, British Columbia, Canada, June 3-5, 1996, Proceedings*, volume 1084 of *Lecture Notes in Computer Science*, pages 404–414. Springer, 1996.
- [96] J. A. Hoogeveen and Arjen P. A. Vestjens. Optimal on-line algorithms for single-machine scheduling. In *Proc. 5th Conf. Integer Programming and Combinatorial Optimization (IPCO)*, volume 1084 of *Lecture Notes in Comput. Sci.*, pages 404–414. Springer, 1996.
- [97] Zhiyi Huang and Xinkai Shu. Online stochastic matching, Poisson arrivals, and the natural linear program. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 682–693. ACM, 2021.
- [98] Zhiyi Huang, Xinkai Shu, and Shuyi Yan. The power of multiple choices in online stochastic matching. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 91–103. ACM, 2022.
- [99] Zhiyi Huang, Xinkai Shu, and Shuyi Yan. The power of multiple choices in online stochastic matching. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 91–103. ACM, 2022.
- [100] Sungjin Im, Ravi Kumar, Shi Li, Aditya Petety, and Manish Purohit. On-line load and graph balancing for random order inputs. In Kunal Agrawal

- and Erez Petrank, editors, *Proceedings of the 36th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2024, Nantes, France, June 17-21, 2024*, pages 491–497. ACM, 2024.
- [101] Nicole Immorlica, Sahil Singla, and Bo Waggoner. Prophet inequalities with linear correlations and augmentations. In *Proceedings of the 21st ACM Conference on Economics and Computation*, pages 159–185, 2020.
- [102] Samin Jamalabadi and Uwe Schwiegelshohn. Wsrpt is 1.2259-competitive for weighted completion time scheduling. *arXiv preprint arXiv:2307.07739*, 2023.
- [103] Samin Jamalabadi and Uwe Schwiegelshohn. WSRPT is 1.2259-competitive for weighted completion time scheduling. *CoRR*, abs/2307.07739, 2023.
- [104] Klaus Jansen, Kim-Manuel Klein, and José Verschae. Closing the gap for makespan scheduling via sparsification techniques. *Mathematics of Operations Research*, 45(4):1371–1392, 2020.
- [105] Jiashuo Jiang, Will Ma, and Jiawei Zhang. Tight guarantees for multi-unit prophet inequalities and online stochastic knapsack. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 1221–1246. SIAM, 2022.
- [106] Bala Kalyanasundaram and Kirk Pruhs. An optimal deterministic algorithm for online b-matching. *Theor. Comput. Sci.*, 233(1-2):319–325, 2000.
- [107] Haim Kaplan, David Naori, and Danny Raz. Almost tight bounds for online facility location in the random-order model. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1523–1544. SIAM, 2023.
- [108] Richard M Karp, Umesh V Vazirani, and Vijay V Vazirani. An optimal algorithm for on-line bipartite matching. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 352–358, 1990.
- [109] Tsuyoshi Kawaguchi and Seiki Kyan. Worst case bound of an lrf schedule for the mean weighted flow-time problem. *SIAM Journal on Computing*, 15(4):1119–1129, 1986.

- [110] Thomas Kesselheim and Marco Molinaro. Knapsack secretary with bursty adversary. *arXiv preprint arXiv:2006.11607*, 2020.
- [111] Ishai Kones and Asaf Levin. A unified framework for designing EPTAS for load balancing on parallel machines. *Algorithmica*, 81(7):3025–3046, 2019.
- [112] Janardhan Kulkarni, Yang P Liu, Ashwin Sah, Mehtaab Sawhney, and Jakub Tarnawski. Online edge coloring via tree recurrences and correlation decay. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 104–116, 2022.
- [113] Janardhan Kulkarni, Victor Reis, and Thomas Rothvoss. Optimal online discrepancy minimization. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, STOC 2024, page 1832–1840, New York, NY, USA, 2024. Association for Computing Machinery.
- [114] Tung-Wei Kuo. Online deterministic minimum cost bipartite matching with delays on a line. *CoRR*, abs/2408.02526, 2024.
- [115] William Kuszmaul and Shyam Narayanan. Optimal time-backlog tradeoffs for the variable-processor cup game. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPICs*, pages 85:1–85:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.
- [116] Jacques Labetoulle, Eugene L Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. Preemptive scheduling of uniform machines subject to release dates. In *Progress in combinatorial optimization*, pages 245–261. Elsevier, 1984.
- [117] Eugene L. Lawler, Jan Karel Lenstra, Alexander H. G. Rinnooy Kan, and David B. Shmoys. Chapter 9 sequencing and scheduling: Algorithms and complexity. In Stephen C. Graves, Alexander H. G. Rinnooy Kan, and Paul Herbert Zipkin, editors, *Logistics of Production and Inventory*, volume 4 of *Handbooks in Operations Research and Management Science*, pages 445–522. North-Holland, 1993.
- [118] J.K. Lenstra, A.H.G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. In P.L. Hammer, E.L. Johnson, B.H. Korte, and G.L. Nemhauser, editors, *Studies in Integer Programming*, volume 1 of *Annals of Discrete Mathematics*, pages 343–362. Elsevier, 1977.

- [119] Wenjie Li. A best possible online algorithm for the parallel-machine scheduling to minimize the maximum weighted completion time. *Asia-Pacific Journal of Operational Research*, 32(04):1550030, 2015.
- [120] Jingxun Liang, Zhihao Gavin Tang, Yixuan Even Xu, Yuhao Zhang, and Renfei Zhou. On the perturbation function of ranking and balance for weighted online bipartite matching, 2023.
- [121] Xiaoxiao Liang, Lingfa Lu, Xueke Sun, Xue Yu, and Lili Zuo. Online scheduling on a single machine with one restart for all jobs to minimize the weighted makespan. *AIMS Mathematics*, 9(1):2518–2529, 2024.
- [122] Hai-Ling Liu and Xi-Wen Lu. Online scheduling on a parallel batch machine with delivery times and limited restarts. *Journal of the Operations Research Society of China*, pages 1–19, 2022.
- [123] Ming Liu, Feifeng Zheng, Shijin Wang, and Jiazhen Huo. Optimal algorithms for online single machine scheduling with deteriorating jobs. *Theor. Comput. Sci.*, 445:75–81, 2012.
- [124] Peihai Liu and Xiwen Lu. On-line scheduling of parallel machines to minimize total completion times. *Comput. Oper. Res.*, 36(9):2647–2652, 2009.
- [125] Vasilis Livanos and Ruta Mehta. Minimization is harder in the prophet world. In David P. Woodruff, editor, *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pages 424–461. SIAM, 2024.
- [126] Lingfa Lu and Liqi Zhang. A PTAS for single-machine scheduling with release dates and job delivery to minimize makespan. *RAIRO Operations Research*, 53(4):1261–1266, 2019.
- [127] Lingfa Lu, Liqi Zhang, and Jinwen Ou. Single machine scheduling with rejection to minimize the weighted makespan. In *Algorithmic Aspects in Information and Management: 15th International Conference, AAIM 2021, Virtual Event, December 20–22, 2021, Proceedings 15*, pages 96–110. Springer, 2021.
- [128] Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. *CoRR*, abs/1802.05399, 2018.

- [129] Ran Ma, Jiping Tao, and Jinjiang Yuan. Online scheduling with linear deteriorating jobs to minimize the total weighted completion time. *Appl. Math. Comput.*, 273:570–583, 2016.
- [130] Calum MacRury, Will Ma, and Nathaniel Grammel. On (random-order) on-line contention resolution schemes for the matching polytope of (bipartite) graphs. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 1995–2014. SIAM, 2023.
- [131] Adam Meyerson. Online facility location. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 426–431. IEEE Computer Society, 2001.
- [132] Benjamin Moseley, Heather Newman, and Kirk Pruhs. Online k-median with consistent clusters. In Amit Kumar and Noga Ron-Zewi, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2024, August 28-30, 2024, London School of Economics, London, UK*, volume 317 of *LIPICs*, pages 20:1–20:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.
- [133] Aniket Murhekar, David Arbour, Tung Mai, and Anup B. Rao. Brief announcement: Dynamic vector bin packing for online resource allocation in the cloud. In Kunal Agrawal and Julian Shun, editors, *Proceedings of the 35th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2023, Orlando, FL, USA, June 17-19, 2023*, pages 307–310. ACM, 2023.
- [134] Joseph (Seffi) Naor and David Wajc. Near-optimum online ad allocation for targeted advertising. *ACM Trans. Economics and Comput.*, 6(3-4):16:1–16:20, 2018.
- [135] Cynthia A. Phillips, Clifford Stein, and Joel Wein. Scheduling jobs that arrive over time (extended abstract). In Selim G. Akl, Frank K. H. A. Dehne, Jörg-Rüdiger Sack, and Nicola Santoro, editors, *Algorithms and Data Structures, 4th International Workshop, WADS '95, Kingston, Ontario, Canada, August 16-18, 1995, Proceedings*, volume 955 of *Lecture Notes in Computer Science*, pages 86–97. Springer, 1995.
- [136] Marc E. Posner. The deadline constrained weighted completion time problem: Analysis of a heuristic. *Oper. Res.*, 36(5):742–746, 1988.

- [137] Feng Qi and Yuan Jinjiang. NP-hardness of a multicriteria scheduling on two families of jobs. *Operations Research Transactions*, pages 121–126, 2007.
- [138] Maurice Queyranne. Structure of a simple scheduling polyhedron. *Math. Program.*, 58:263–285, 1993.
- [139] Sharath Raghvendra. A robust and optimal online algorithm for minimum metric bipartite matching. In Klaus Jansen, Claire Mathieu, José D. P. Rolim, and Chris Umans, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2016, September 7-9, 2016, Paris, France*, volume 60 of *LIPICs*, pages 18:1–18:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.
- [140] Rajmohan Rajaraman and Omer Wasim. Improved bounds for online balanced graph re-partitioning. In Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman, editors, *30th Annual European Symposium on Algorithms, ESA 2022, September 5-9, 2022, Berlin/Potsdam, Germany*, volume 244 of *LIPICs*, pages 83:1–83:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.
- [141] Rajmohan Rajaraman and Omer Wasim. Competitive capacitated online recoloring. In Timothy Chan, Johannes Fischer, John Iacono, and Grzegorz Herman, editors, *32nd Annual European Symposium on Algorithms, ESA 2024, September 2-4, 2024, Royal Holloway, London, United Kingdom*, volume 308 of *LIPICs*, pages 95:1–95:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.
- [142] David B. Shmoys, Joel Wein, and David P. Williamson. Scheduling parallel machines on-line. In Lyle A. McGeoch and Daniel Dominic Sleator, editors, *On-Line Algorithms, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, February 11-13, 1991*, volume 7 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 163–166. DIMACS/AMS, 1991.
- [143] René Sitters. Efficient algorithms for average completion time scheduling. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 411–423. Springer, 2010.
- [144] Allan Sly. Uniqueness thresholds on trees versus graphs. *The Annals of Applied Probability*, 18(5):1897–1909, 2008.

- [145] Wayne E Smith et al. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2):59–66, 1956.
- [146] W.E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3:59–66, 1956.
- [147] Leen Stougie and Arjen P. A. Vestjens. Randomized algorithms for on-line scheduling problems: how low can’t you go? *Oper. Res. Lett.*, 30(2):89–96, 2002.
- [148] Ruiqing Sun. Randomized approximation schemes for minimizing the weighted makespan on identical parallel machines. *Journal of Combinatorial Optimization*, 47(3):34, 2024.
- [149] Jiping Tao, Zhijun Chao, Yugeng Xi, and Ye Tao. An optimal semi-online algorithm for a single machine scheduling problem with bounded processing time. *Inf. Process. Lett.*, 110(8-9):325–330, 2010.
- [150] Mikkel Thorup. Randomized sorting in $o(n \log \log n)$ time and linear space using addition, shift, and bit-wise boolean operations. *J. Algorithms*, 42(2):205–230, 2002.
- [151] Ji Tian, Ruyan Fu, and Jinjiang Yuan. Online over time scheduling on parallel-batch machines: a survey. *Journal of the Operations Research Society of China*, 2(4):445–454, 2014.
- [152] Noam Touitou. Nearly-tight lower bounds for set cover and network design with deadlines/delay. In Hee-Kap Ahn and Kunihiro Sadakane, editors, *32nd International Symposium on Algorithms and Computation, ISAAC 2021, December 6-8, 2021, Fukuoka, Japan*, volume 212 of *LIPIcs*, pages 53:1–53:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.
- [153] Marjan van den Akker, Han Hoogeveen, and Nodari Vakhania. Restarts can help in the on-line minimization of the maximum delivery time on a single machine. In Mike Paterson, editor, *Algorithms - ESA 2000, 8th Annual European Symposium, Saarbrücken, Germany, September 5-8, 2000, Proceedings*, volume 1879 of *Lecture Notes in Computer Science*, pages 427–436. Springer, 2000.
- [154] Rob van Stee and Johannes A. La Poutré. Minimizing the total completion time on-line on a single machine, using restarts. *Journal of Algorithms*, 57(2):95–129, 2005.

-
- [155] Dror Weitz. Counting independent sets up to the tree threshold. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 140–149, 2006.
 - [156] Philip Whittington. Online string attractors. *CoRR*, abs/2407.15599, 2024.
 - [157] Shuyi Yan. Edge-weighted online stochastic matching: Beating $1 - \frac{1}{e}$. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 4631–4640, 2024.
 - [158] Lili Zuo, Jing Zhang, Zhan Shi, and Bingbing Fan. Single machine scheduling with a restricted rate-modifying activity to minimize the weighted makespan. *Discrete Applied Mathematics*, 380:89–100, 2026.