Coping with Incomplete Information in Scheduling

Stochastic and Online Models

vorgelegt von
Dipl.-Math. oec. Nicole Megow

Von der Fakultät II – Mathematik und Naturwissenschaften
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften
– Dr. rer. nat. –

genehmigte Dissertation

Berichter:	Prof. Dr. Rolf H. Möhring
	Prof. Dr. Martin Skutella
Vorsitzender:	Prof. Dr. Fredi Tröltzsch

Tag der wissenschaftlichen Aussprache: 18. Dezember 2006

Berlin 2007
D 83

Acknowledgements

This thesis would not have been possible without the help of a number of people. I wish to thank all those who have been involved directly or indirectly. In particular, I thank my advisor Rolf Möhring for his support and trust, and for letting me pursue independent research on topics that I am interested in. I benefited a lot from the opportunity to join the academic research community by visiting workshops, international conferences, and other research groups.

Also I wish to thank Martin Skutella for taking the second assessment of this thesis. His enthusiasm and his stimulating ideas in any discussion are amazing and keep infecting me.

Furthermore, I am greatly indebted to my collaborators on material in this thesis, Sven Krumke, Andreas Schulz, Marc Uetz, and Tjark Vredeveld. I enjoyed the discussions, and definitely learned a lot from all of them.

It was certainly a very fortunate circumstance to meet Tjark in the beginning of my time as a PhD student. It was a pleasure working with him on different problems, convincing each other of new ideas or doubting those of the other one, and sharing moments of cheering as well as disappointment – although, I hardly remember the latter. I thank him for being such a good research sparring partner and a great friend.

I also want to thank Ekkehard (Ekki) Köhler, Guido Schäfer, Björn Stenzel, Tjark, Axel Werner, and Gregor Wünsch for reading different parts of this thesis.

Besides the people that where directly involved in preparing this thesis, several people contributed indirectly. Thanks to the members of the research groups of Rolf Möhring, Günter Ziegler, and Stefan Felsner for the wonderful working atmosphere. Moreover, I would like to thank Ekki, Stefan Heinz, Jörg Rambau, René Sitters, Björn Stenzel, Leen Stougie, and Andreas Tuchscherer for pleasant joint work, fruitful discussions and motivation in various ways.

I am extremely grateful to my parents and my sister for their unlimited support, their unshakeable believe in me, and for their interest in my work. Finally, I wish to thank Matthias Habenicht for his constant support and encouragement, and always patiently listening to me in moments of frustration and euphoria.

Berlin, October 2006 *Nicole Megow*

Contents

Introduction

Incomplete information is an omnipresent issue when dealing with real-world optimization problems. Typically, such limitations concern the uncertainty of given data or the complete lack of knowledge about future parts of a problem instance. This thesis is devoted to investigations on how to cope with incomplete information when solving scheduling problems.

To cope with scenarios where there is uncertainty about the problem data or parts of the instance are not known at all, there are two major frameworks in the theory of optimization. One deals with *stochastic information*, the other with *online information*. In the stochastic information environment, probabilistic assumptions are made on the data set or parts of it. Usually, the distribution functions of the respective random variables are assumed to be known in advance before executing an optimization algorithm. The consideration of stochastic data is a major step in terms of bridging the gap from theory to practice. However, the decisive assumption on the a priori knowledge of the entire instance and the distributions appears quite strong for certain applications. In contrast, in an online information environment, the assumption is that a problem instance is presented to an algorithm only piecewise. The actual problem data are usually disclosed upon arrival of a request, and decisions must be made based on partial knowledge of the instance. This second model appears more restrictive in the informational sense since it models the complete lack of knowledge about the future. On the other hand, once the data are revealed, they are usually completely certain.

Both frameworks have their legitimacy depending on the actual application. Nevertheless, problem settings are conceivable that comprise both, uncertain information about the data set and the complete lack of knowledge about the future. This rouses the need for a generalized model that integrates both traditional information environments.

In this thesis, we deal with all three of the mentioned frameworks, the classic ones with stochastic and online information, respectively, as well as a generalized model that combines both traditional models in a joint framework.

The particular problem class we consider is the class of scheduling problems which plays an important role within combinatorial optimization. These problems involve the temporal allocation of limited resources for executing activities so as to optimize some objective. Scheduling problems are apparent in many applications including, for example, manufacturing and service industries but also compiler optimization and parallel computing.

Practitioners and researchers are naturally interested in objective functions that are related to the minimization of completion times. The two classic types to mention here are the makespan, which is the completion time of the final job in the system, and the total (weighted) completion time of all jobs. Clearly, both types provide satisfying information on the scheduling performance for different applications. However, the individual job related objective function – the sum of all completion times – captures important details on the operations efficiency which are not reflected in the makespan objective. Therefore, we focus in this thesis mainly on problems to minimize the sum of weighted completion times. Only in the last chapter we deviate from this procedure. We not only consider a deadline based objective, we also leave the standard scheduling framework and add a locality component; that means, scarce resource must be allocated over time and, in addition, their position in a metric space must be respected.

These problems are $\mathcal{NP}$-hard, in general. Thus, by the current state-of-the-art, we cannot hope to find efficient algorithms that solve these problems to optimality. Therefore, we are restricted to find near-optimal solutions efficiently. A widely accepted measure for the performance of an algorithm is the approximation ratio, that is, the ratio of the outcome of an algorithm over the optimum value. In the past three decades, approximation algorithms for many $\mathcal{NP}$-hard optimization problems were designed and analyzed. Interestingly, one of the first approximation algorithms with a formalized analysis is due to Graham [Gra69] and was designed for a machine scheduling problem.

Certainly, these problems do not lose their intractability when information about the problem data is limited. Performance measures comparable to the approximation ratio, exist for both traditional restricted information environments, the online and the stochastic model – even though there is less agreement on the "right" measure. Within these frameworks, we design algorithms for certain scheduling problems. Thereby we provide first constant performance guarantees or improve previously best known results.

A general model that combines stochastic and online information is certainly of interest and can be designed as a natural extension. But the challenging question is whether there exists any algorithm that can perform well in such a restricted information environment. More precisely, is there an algorithm that yields a constant performance guarantee? We successfully treat this intriguing question and give a positive answer by providing such algorithms for machine scheduling problems. In fact, our results are competitive with the performance guarantees best known in the traditional settings of stochastic and online scheduling. Thus, they do not only justify the generalized model but also imply – at least in the considered problem settings – that optimization in the general model with incomplete information does not necessarily mean to give up performance.

Outline of the Thesis

Chapter 1. In this preliminary chapter we introduce basic concepts for modeling incomplete information on problem data. For both standard models that we deal with, the online and the stochastic model, we give details on terminology and typical (as well as alternative) performance measures. Furthermore, we introduce the problem classes that we focus on in this thesis; this includes the diverse class of *scheduling problems* as well as a brief overview on classic *traveling salesman problems.*

Chapter 2. The second chapter concerns online scheduling problems. We investigate algorithms for scheduling with the objective to minimize the total weighted completion time on single as well as on parallel machines. We consider both, a setting with independent jobs and one where jobs must obey precedence relations.

For independent jobs arriving online, we design and analyze algorithms for both, the preemptive and the non-preemptive setting. These online algorithms are extensions of the classical Smith rule (Wspt algorithm) and yield performance guarantees that are improving on the previously best known ones. A natural extension of Smith's rule to the preemptive setting is 2-competitive. Nevertheless, we propose a modified, more dynamic variant of this classic algorithm for the single-machine problem which achieves the same competitive ratio. While the performance analysis of the first algorithm is proven to be tight, the lower bound of our dynamic rule leaves a gap that raises hope that its true performance is much better. For the non-preemptive variant of the multiple-machine scheduling problem, we derive a 3.281-competitive algorithm that combines a processing time dependent waiting strategy with Smith's rule.

For the scenario in which precedence constraints among jobs are given, we discuss a reasonable online model and give lower and upper bounds on the competitive ratio for scheduling without job preemptions. In this context, previous work on the offline problem of scheduling jobs with generalized precedence constraints, the so called And/Or-precedence relations, appears to be adoptable to a certain extent; we discuss the relevance.

CHAPTER 3. The third chapter is devoted to the stochastic scheduling model. After reviewing (approximation) results in this area, we focus on preemptive stochastic scheduling. We present a first constant approximation for a preemptive problem to minimize the sum of weighted completion times. For scheduling jobs with release dates on identical parallel machines we derive a policy with a guaranteed performance ratio of 2 which matches the currently best known result for the corresponding deterministic online problem (derived in the previous chapter).

In contrast to the previously known results in the non-preemptive setting, our preemptive policy extensively utilizes information on processing time distributions other than the first (and second) moments. In order to derive our result we introduce a new non-trivial lower bound on the expected value of an unknown optimal policy. This bound is derived by borrowing ideas for a *fast single-machine relaxation* known from deterministic online scheduling. The crucial ingredient to our result is then the application of a Gittins index priority policy which is optimal to a relaxed version of our fast single-machine relaxation. This priority index also inspires the design of our policies.

CHAPTER 4. In this chapter, we consider a model for scheduling with incomplete information which combines the main characteristics of online and stochastic scheduling, as considered in the previous two chapters, in a simple and natural way. Job processing times are assumed to be stochastic, but in contrast to the traditional stochastic scheduling model, we assume that jobs arrive online, and there is no knowledge about the jobs that will arrive in the future. We name this model the *stochastic online scheduling model.* Indeed, it incorporates both, stochastic scheduling and online scheduling as a special case – also with respect to the performance measure.

The particular problems we consider are preemptive and non-preemptive parallel-machine scheduling with the objective to minimize the total weighted completion times of jobs. For the problem where jobs must run until completion without interruption, we analyze simple, combinatorial online scheduling policies and derive performance guarantees that match the currently best known performance guarantees for stochastic and online parallel-machine scheduling. For processing times that follow NBUE distributions, we improve upon previously best known performance bounds from stochastic scheduling, even though we consider a more general setting.

Finally, we argue that the results on preemptive stochastic (offline) scheduling presented in the previous chapter also apply in this more general model because the preemptive policy is feasible in an online setting as well.

Chapter 5. In the final chapter, we turn back to the traditional online optimization model. However, we deviate slightly from our pure scheduling focus and consider a variant of an online *traveling salesman problem* (Tsp). In this problem, requests with deadlines arrive online over time at points of a metric space. One or more servers move in a metric space at constant speed and serve requests by reaching the corresponding position before the deadline. The goal is to guide the server through the metric space such that the number of served requests is maximized. Thus, we have a Tsp-like problem with a maximization objective.

Applying standard competitive analysis, we study the problem on restricted metric spaces, namely the real line and the uniform metric space. While on the line no deterministic algorithm can achieve a constant competitive ratio, we provide competitive algorithms for the uniform metric space. Our online investigations are complemented by complexity results for the offline problem.

Even though this problem is not a standard scheduling problem, we associate it with this problem class in a broader sense. In a way, Tsp-like problems can be interpreted as scheduling problems with an additional locality component: an algorithm decides about the timing of the server moves, and in addition, it determines the direction of moves. However, the locality aspect becomes less important on the restricted spaces that we consider.

Chapter 1

Preliminaries

We investigate optimization problems in different information environments. They all have in common that a solution must be found based on incomplete information on the problem input. The classical ways to model such limitations of knowledge is to assume *online* or *stochastic* information.

In this chapter, we give details on both standard information environments and introduce the optimization problems that we consider in this thesis. After separate, detailed examinations on scheduling problems in both, an online and stochastic setting, in the following two chapters, we will discuss a combined model of *stochastic online information* and successfully analyze scheduling problems within this model in Chapter 4.

1.1 Introduction of Problems

In the major part of this thesis, we focus on a certain class of scheduling problems. In the following Section 1.1.1 we give an overview including a classification scheme and some definitions. In Section 1.1.2 we introduce a second class of well-known optimization problems, namely the traveling salesman problem with different objective functions.

1.1.1 Scheduling Problems and Standard Notation

Scheduling problems have been studied extensively for decades and are still of ongoing interest. The universal problem of assigning tasks temporally to scarce resources occurs in endless variations in the real world. Therefore, a remarkable amount of different scheduling models has been introduced in the literature. We refer to a recent and quite exhaustive collection of surveys on various models and problem settings edited by Leung [Leu04].

In this thesis, we consider certain problems within the class of so-called *machine scheduling problems*. In general, such problems can be described as follows: There is given a set $\mathcal{J} = \{1, 2, \ldots, n\}$ of jobs which must be scheduled on one or more machines. Each job $j \in \mathcal{J}$ has associated a positive processing time $p_j \geq 0$ and a non-negative weight $w_j \geq 0$. The goal is to find an assignment of jobs to time slots and machines – which we call a *schedule* – such that certain problem specific constraints are met and an objective function is minimized.

Typically, (machine) scheduling problems are described using a standard classification scheme that was initially introduced by Graham, Lawler, Lenstra, and Rinnooy Kan [GLLR79], which is also called the three-field notation $\alpha \,|\, \beta \,|\, \gamma$. For the sake of completeness we review this standard notation but focus on scheduling problems that we consider in this thesis.

The parameter α in the first field describes the machine environment. We consider the following two choices:

- 1 *(single machine).* Jobs must be scheduled on a single processor which can process only one job at a time.
- P *(parallel identical machines).* Each job can be processed on any of the identical parallel machines. The number of available machines is denoted by m. Each of the m machines can handle only one job at a time.

The β field represents job characteristics that are given to each job j in addition to its processing time p_j and weight w_j.

- r_j *(release date).* Each job has associated a release date; without this parameter, we assume $r_j = 0$ for all jobs j. The release date defines the earliest point in time when a job is available for processing. In our online models, this will also be the time when the scheduler learns about the existence of the job.

  If not stated differently, we assume for convenience that jobs are indexed such that $r_j \leq r_k$ for $j < k$.

- *pmtn (preemption).* Including this parameter indicates that job preemption is allowed. That means that the processing of a job may be suspended and resumed later on any machine at no extra cost. In contrast, if preemption is not permitted (that is, if the parameter is not set) then a job that has started must run until its completion on the same machine without any interruption.

- *prec (precedence constraints).* Precedence constraints define a partial order $(\mathcal{J}, \prec)$ on the set of jobs $\mathcal{J}$. Jobs have to be scheduled in compliance with these constraints, where $j \prec k$ implies that job k must not start processing before j has completed.

If no precedence constraints are given, then we call the jobs independent.

– d_j *(deadline).* The deadline of a job represents the date by which a job must be completed. If a job has not finished until its deadline, then it is called *late*. If this parameter is not included in the β-field, then all deadlines are assumed to be infinite, that means, they are non-existent.[1]

Finally, the third field, γ, indicates the objective function. We mainly consider one objective function in different models for scheduling with limited information.

– $\sum w_j C_j$ (*minimize the sum of weighted completion times*). Here, C_j denotes the completion time of a job j. If all job weights are equal, then this objective is equivalent to minimizing the average completion time.

– $\mathbb{E}\left[\sum w_j C_j\right]$ (*minimize the sum of weighted completion times in expectation*). In the stochastic scheduling environment (in Chapters 3 and 4) our goal is to minimize the objective function above in expectation. By having an expected objective value in the γ field we also denote that the jobs have stochastic processing times which is in some literature denoted by $p_j \sim$ *stoch* in the β field. Since the only stochastic problem characteristics in this work are stochastic processing times, we omit this truly exact – but in our case redundant – description.

Clearly, the relevance of certain objective functions is discussible and depends on the particular application. An objective that has gained a fair amount of attention recently is to *minimize the sum of (weighted) flow times*, $\sum(w_j)\,F_j$, where the flow time is defined as $F_j = C_j - r_j$. Problems of this type seem best to reflect the goal of certain scheduling problems; on the other hand, they are extremely intractable. The probably most widely studied objective is to minimize the latest completion time among all jobs, $C_{\max} := \max_{j\in\mathcal{J}} C_j$, also called the *makespan* of a schedule. Finally, we mention the objective to *minimize the number of late jobs*, $\sum U_j$, which turns out to be relevant for a discussion in Chapter 5; here, $U_j = 1$ indicates that the job j is late, that is, $C_j > d_j$, and otherwise we have $U_j = 0$. Obviously, this objective coincides with aiming for maximizing the number of jobs that are completed on time, that is, when $C_j \leq d_j$. (While in the minimization problem all jobs must

[1]We do not explicitly investigate scheduling problems with deadlines. However, we will discuss the close relation between such problems and the Tsp-related problem considered in Chapter 5.

be executed as we generally require, this is not the case in the maximization problem.)

As pointed out above, scheduling problems have attracted researchers for a long time period. The fruitful and ongoing research is documented in a vast amount of literature and it is beyond the scope of this thesis to give an exhaustive overview of previous work in this area. Therefore, we will limit the review to relevant work on several scheduling models throughout the chapters. For a general overview we refer the interested reader to the textbooks of Brucker [Bru04] and Pinedo [Pin02], and the surveys in [Leu04].

1.1.2 Traveling Salesman Problems

One of the fundamental problems in combinatorial optimization is the *traveling salesman problem* (Tsp). Generally speaking, the problem consists of finding a tour through every one of a given set of points (cities). Whereas in its original version the objective is to minimize the total length of the tour through all nodes, a number of variations has been considered in the literature. One well-studied example is the *traveling repairman problem* (Trp) where a tour must be found with minimum (weighted) sum of waiting times of the cities. This problem is also known as the *delivery man problem* or the *minimum latency problem*. In the standard model, waiting times are the arrival dates in the respective city. However, if there are release dates, that is, if a city can be visited only at some time later than a given date, then the waiting time is the time difference between the actual arrival date of the repairman in the city and the earliest possible date of arrival, the release date.

These problems (and objective functions) are certainly the most widely studied problems within the class of Tsp-like problems; see, for example, the book by Lawler, Lenstra, Rinnooy Kan, and Shmoys [LLKS85]. However, other objectives are conceivable as, for instance, the maximization of the number of visited cities given that each city has associated a deadline for being visited. Bansal, Blum, Chawla, and Meyerson [BBCM04] considered this problem recently as *deadline*-Tsp. If each city has associated not only a strict upper bound on the visiting time (deadline) but also a lower bound, a release date, then they call the problem *vehicle routing with time-windows*. In Chapter 5 we investigate an online variant of this problem class. Actually, our online problem has been considered earlier under the name *dynamic traveling repair problem* [ILR04] which – as we feel – is a somewhat unfortunate name choice because it is almost identical to the dynamic version of the Trp above and thus it suggests a different objective.

The importance of traveling salesman problems stems not only from its variety of obvious applications but also from its versatility: seemingly unrelated combinatorial optimization problems can be formulated in terms of the Tsp. As one example, we describe here the connection to *scheduling problems* in order to expose the strong relation between the two problem classes that we consider in this thesis. For more examples, see for instance the book chapter by Garfinkel [Gar85].

It is well-known that certain machine scheduling problems can be formulated as special cases of traveling salesman problems. In the problem of so-called *scheduling with sequence-dependent setup times* there is given a set of jobs, and for each job pair (i, j) there is a setup time s_{ij} which is incurred if job j is processed immediately after job i. The goal is to assign jobs temporally to a single processor as efficiently as possible according to some objective function. See reviews [AGA99, YL99] for an overview of common models and research in this field.

In case that all setup times are symmetric, this problem can be interpreted as an undirected graph problem in the graph that has a node for each job and an edge with cost s_{ij} between the nodes corresponding to jobs i and j. In addition, there is a dummy node d with an edge to any other node j with cost zero. The problem of finding the schedule with minimum makespan is then equivalent with finding a shortest tour in this graph – the classic Tsp. Moreover, the scheduling setting is fairly general and models many different problems. For instance, if deadlines are given then the scheduling problem with the goal of maximizing the number of jobs that complete before their specified deadlines is equivalent to the *deadline*-Tsp. Job processing times p_j can be incorporated also in this model by adding $p_j/2$ to all edges incident with the node corresponding to job j and subtracting the same value from the deadline.

We conclude this discussion of the relation between traveling salesman problems and scheduling problems with the observation that actually all cost functions used in the different Tsp-variants are similar to the most commonly employed performance measures in scheduling theory, as they are introduced in the previous section. The total travel time (Tsp) coincides with the makespan, $C_{\max}$, the sum of weighted waiting times (Trp) is known as the sum of weighted completion times, $\sum w_j C_j$, or weighted flow times if release dates are present, $\sum w_j F_j$, and the objective of maximizing the number of visited cities coincides with maximizing the number of jobs executed on time or minimizing the number of late jobs, $\sum U_j$, respectively.

1.2 Optimization with Incomplete Information

The focus of this thesis is on handling incomplete information. There are two major information concepts for modeling uncertainty in the data: One is the complete lack of information on (parts of) the input, this is called *online information*, the other one is the assumption of probabilistic information about (parts of) the data, which is called *stochastic information*. In the following subsections we introduce both general information environments and the models that are relevant for this thesis. This will prepare the ground not only for our investigations of online algorithms in Chapter 2 and 5 and stochastic policies in Section 3 but also for an integrated model that generalizes both models for partial knowledge of the input in Chapter 4.

1.2.1 Online Information

Information are given online if the problem input becomes known to the algorithm only piecewise while it is executed. The motivation is obvious since in many real-world problems decisions must be made based only on partial information of the problem instance.

An instance $\mathcal{I}$ of an online optimization problem contains a sequence of requests to be served by an algorithm. The requests are revealed to the online algorithm only step-by-step depending on the particular *online paradigm*. The two most common online paradigms are the *online-list model* and the *online-time model*.

In the *online-list model*, which is also known as the *sequence model*, requests are given to the algorithm in form of an ordered list of which it can see only the head. The algorithm learns about a request and has to service it before the next request is disclosed. The decisions it has made based on its current partial knowledge of the input are fixed and irrevocable.

The *online-time model* is also known as the *time stamp model*. In this information paradigm, requests are revealed to the algorithm online over time at their release date. Regarding the decision making process, an online algorithm has more freedom in this model. It is allowed to postpone decisions or even revoke them as long as they have not been executed.

These two general information paradigms are typically assumed in online optimization. Nevertheless, other online models are conceivable for special problem settings. In fact, in Section 2.3 we consider online scheduling of jobs with precedence constraints where the job arrival time depends on the completion time of all jobs that are predecessors in the partial order.

An additional feature that can be combined with any other online paradigm is *(non-)clairvoyance*. It describes the information that is revealed when a request becomes known. If we assume non-clairvoyance then parts of the data defining a request are not revealed at arrival but become known only by (completing) servicing the request. In scheduling this refers typically to the processing time which becomes known only by job completion. In contrast, in the clairvoyant setting all job characteristics (including the processing time for example) are revealed at request arrival. Most investigations on online problems concern clairvoyant online information. Likewise, the online scheduling problems we consider in Chapter 2 and the online deadline-Tsp in Chapter 5 will be restricted to clairvoyant knowledge. However, the stochastic online scheduling model in Chapter 4 is somewhat semi-clairvoyant in the sense that an algorithm does not learn the entire data defining the request, but for some data, namely the job processing time, only the probability distribution is revealed.

1.2.1.1 Competitive Analysis

The quality of online algorithms is typically assessed by their worst-case performance compared to an *optimal offline algorithm.* Thereby, the outcome of an online algorithm is compared to an optimal solution that can be obtained if one had complete knowledge of the whole input instance beforehand. Such an optimal offline algorithm does not need to be known; in fact, we (mainly) consider problems whose offline versions are $\mathcal{NP}$-hard problems. This technique has been introduced by Sleator and Tarjan [ST85] who analyzed paging and list update rules in an online environment. The actual term *competitive analysis* was coined by Karlin, Manasse, Rudolph, and Sleator [KMRS88].

Definition 1.1 (*Competitive algorithms*)**.** *A deterministic online algorithm* Alg *for a minimization problem is called* ρ-competitive *if, for any problem instance* $\mathcal{I}$*, it achieves a solution*

$$\textsc{Alg}(\mathcal{I}) \;\leq\; \rho \cdot \textsc{Opt}(\mathcal{I})\,,$$

where $\textsc{Opt}(\mathcal{I})$ *denotes the value of an optimal offline solution for the same instance* $\mathcal{I}$*. A randomized online algorithm takes certain decisions randomly. Such an algorithm is called* ρ*-competitive, if it achieves in expectation a solution of value* $\mathbb{E}\left[\,\textsc{Alg}(\mathcal{I})\,\right] \;\leq\; \rho \cdot \textsc{Opt}(\mathcal{I})$.

Considering maximization problems, we call an algorithm ρ*-competitive if it yields for any input instance* $\mathcal{I}$ *a solution (in expectation if randomized actions are taken) of*

$$\textsc{Alg}(\mathcal{I}) \;\geq\; \frac{1}{\rho} \cdot \textsc{Opt}(\mathcal{I})\,.$$

The competitive ratio *of* Alg *is the infimum over all* ρ *such that* Alg *is* ρ*-competitive.*

In the literature, competitiveness is sometimes defined allowing an additional additive constant whereas the definitions above imply that the algorithm ALG is *strictly* ρ-competitive. However, all our ρ-competitive algorithms are strictly ρ-competitive. And for most of our lower bounds, we can easily extend the constructions and therefore neglect this additive constant.

Note, that in the definition above there are no requirements on the computational complexity of competitive algorithms. In contrast to (offline) approximation algorithms,[2] an online algorithm does not need to solve the problem in time that is polynomial in the input data; we say, it does not have to run *efficiently*. Therefore, the performance guarantees of approximation and online algorithms are not actually comparable, even though they are closely related. In fact, a ρ-approximation algorithm that is working online is also ρ-competitive and, conversely, a ρ-competitive online algorithm is a ρ-approximation if it is running efficiently. Anyway, in practice one is primarily interested in efficient algorithms – and our online algorithms meet this demand.

Competitive analysis is often interpreted as a game between an *online player* and a "malicious all-knowing" *adversary*. The online player uses an online algorithm in order to serve a request instance generated by an adversary. The adversary knows the strategy of the online player and constructs an "evil" input instance that maximizes (resp. minimizes in a maximization problem) the ratio between the player's cost and the optimal offline cost.

Against a deterministic online player, it makes no difference for an adversary in which moment of the game it has to specify its "evil" instance. In terms of the information knowledge that means that we do not need to carefully define which information on the actual actions of the online algorithm is available to the adversary since they are deterministically predefined.

If the online player uses randomization then we must specify more precisely the information which an adversary can use. Intuitively, the omniscience of an adversary is broken since the actions of the online algorithm are not certain anymore. This results into different adversary models defining the power of an adversary. Ben-David et al. suggested in [BDBK+94] three types of adversaries. Regarding the availability of information, we will only use the most common and weakest type of adversary – the *oblivious adversary*.

Definition 1.2 (Oblivious adversary)**.** *An* oblivious adversary *must construct the entire request sequence in advance based only on the description of the online algorithm but without knowing the actual, randomly chosen actions that will be taken.*

[2] An algorithm with running time that is polynomial in the input size is a $\rho-$approximation algorithm for a minimization (maximization) problem, if it is guaranteed to find a solution with a value that is no more (less) than ρ times the optimal value.

Additionally, Ben-David et al. [BDBK+94] proposed two different adversaries. Their power is limited by restricting the information they have on the outcome of random choices of the randomized algorithm on the previous input. Several examples in the literature show that restricting the omniscience of the adversary may lead indeed to improved performance guarantees for online algorithms. For an extensive introduction and discussion of adversary models we refer to the textbooks by Motwani and Raghavan on randomized algorithms [MR95] and Borodin and El-Yaniv on competitive analysis [BEY98].

More problem-specific adversaries will be introduced in Chapter 5 for Tsp-like problems. The power of these adversaries is not only restricted by limiting their information but also by disallowing moves into a "direction" where currently no unserved request is available.

Finally, we refer to [BEY98] for an in-depth treatment of online optimization, in general, and competitive analysis. Furthermore, we point at a collection of surveys on various online optimization problems and state-of-the-art techniques for their solution and analysis, edited by Fiat and Woeginger [FW98b].

1.2.1.2 Lower Bounds on the Competitive Ratio – Yao's Minimax Principle

Lower bounds on the competitive ratio are typically derived by providing a class of input instances on which no online algorithm can perform well when compared to an optimal offline algorithm. Finding appropriate instances is often comparatively easy for deterministic algorithms. This changes when considering randomized algorithms because their solution value is a random variable and its expected value on a certain instance is usually hard to bound.

For such cases *Yao's Minimax Principle* [Yao77] has proven a useful technique for deriving lower bounds.[3] Generally speaking, it states, that a lower bound on the expected competitive ratio of any deterministic online algorithm on an appropriate input distribution also lower bounds the competitive ratio of any randomized online algorithm against an oblivious adversary. (Since this is the weakest type of adversaries, these lower bounds hold against any other adversary with more power as well.) More formally, we use the following version of Yao's Theorem.

[3]Actually, Yao [Yao77] was first to observe that using the von Neumann minimax theorem for two-person games in game theory, a lower bound on the performance of randomized algorithms in these models can be obtained by choosing any probability distribution on the input and proving a lower bound on the expected cost of any deterministic algorithm on this input. The first application in the context of competitive analysis of online algorithms was by Borodin, Linial, and Saks [BLS92] for metrical task systems.

Theorem 1.3 (Yao's Minimax Principle). *Let $\mathcal{A} = \{\textsc{Alg}_1, \textsc{Alg}_2, \ldots\}$ denote the set of all deterministic online algorithms for a minimization problem and let $\mathcal{D}$ be a distribution over inputs of this problem. If c is a positive constant such that*

$$\inf_{\textsc{Alg}_i \in \mathcal{A}} \mathbb{E}_{\mathcal{D}}\left[\textsc{Alg}_i(\mathcal{I})\right] \geq c \cdot \mathbb{E}_{\mathcal{D}}\left[\textsc{Opt}(\mathcal{I})\right]$$

and $\mathbb{E}_{\mathcal{D}}\left[\textsc{Opt}(\mathcal{I})\right] > 0$ then c is a lower bound on the competitive ratio for any randomized online algorithm against an oblivious adversary.

The power of this technique lies in reducing the problem of proving a lower bound c for any randomized algorithm (and any distribution for its random choices) to the problem of finding *one* distribution $\mathcal{D}$ over input instances on which any deterministic algorithm is in expectation not c-competitive. This seems much easier – in particular, the distribution $\mathcal{D}$ can be chosen freely and is known to the deterministic algorithm.

This theorem can be reformulated for maximization problems in the obvious way; see [MR95, BEY98] for details on the derivation of the theorem and the background in game theory.

1.2.1.3 Alternatives for Competitive Analysis

Competitive analysis has often been criticized for being overly pessimistic. Indeed, it measures the quality of an algorithm by its performance on a worst-case input instance which is typically some pathological construction that hardly occurs in practice. Thus, it is an arguable means for comparing the performance of algorithms.

Two algorithms A and B may have the same "bad" competitive ratio while A is performing on real-world instances superior to algorithm B. The reason is that in competitive analysis, there is no option of weighting the amount of instances on which an algorithm performs well or not. Algorithm A might fail on just a few worst-case instances whereas algorithm B performs bad in general. In an extreme scenario, we may find a lower bound on the competitive ratio of any online algorithm which matches the upper bound of some completely trivial algorithm. In this unfortunate case, algorithms are indistinguishable by their worst-case performance even though in practice they may perform very differently. Fiat and Woeginger [FW98a] call this phenomena "hitting the triviality barrier". Nevertheless, the true intention of any performance analysis is to compare online algorithms with each other – not against some adversary with magical powers. If worst-case analysis does not give appropriate performance measures to distinguish algorithms, it is reasonable to search for alternatives.

Diffuse adversary. As we have mentioned in the previous section, the power of an online algorithm can be increased by restricting the power of the adversary. Koutsoupias and Papadimitriou [KP00b] suggest a certain way of limiting the adversarial power in the so-called *diffuse adversary model*. Here, an adversary generates an input sequence according to a probability distribution that belongs to a certain class of possible distributions which is known to the online algorithm. As a consequence, it is not one single worst-case instance that is responsible for the performance of the algorithm.

Probabilistic analysis or (distributional) average case analysis. Probabilistic analysis determines the average case performance of an algorithm on a given probability distribution of input instances. The obvious difficulty is to choose a certain probability distribution and argue convincingly that this distribution captures realistic instances. There are always other realistic instances for other scenarios and the real-world problems are changing dynamically. Moreover, an analysis of those realistic distribution functions seems often nearly impossible.

Smoothed analysis. Spielman and Teng [ST04] introduced smoothed analysis as an alternative to the standard worst-case and distributional average-case analyses in the context of measuring running time complexity. The central idea is that an input instance is slightly perturbed by random (property-preserving) noise. Then an algorithm is analyzed with respect to its expected performance on this modified instance. Due to the random perturbation of an adversarial instance, the performance of an algorithm cannot be decided directly by artificial counterexamples constructed by an adversary. The hope is that this noise turns a pathological worst-case instance into an instance which is "easy" to solve. An argument favoring this perturbation is that data in real-world applications are noisy anyway.

Smoothed analysis has been successfully applied to characterize the running time complexity of algorithms for several problems. However, the general idea of smoothening the input can be adopted to analyze the performance of online algorithms, as well; this has been done successfully by Becchetti, Leonardi, Marchetti-Spaccamela, Schäfer, and Vredeveld [BLMS+06]. They introduce the *smoothed competitive analysis* and examine a widely used algorithm for non-clairvoyant online scheduling to minimize total flow time, $1 \mid r_j, pmtn \mid \sum F_j$, that behaves poorly with respect to worst-case analysis but seems very efficient in practice.

Resource augmentation and Lookahead. Another tool that has proven successful is *resource augmentation*, introduced by Kalyanasundaram and Pruhs in [KP00a]. It again strengthens the power of an online algorithm against an offline optimum. The idea is to augment an online algorithm with more resources, e.g., in the form of faster machines. Then the goal is to find a constant-competitive algorithm for a modest amount of speed up. (In other models, the number of available resources is increased or their size.)

In a way, *lookahead* can be interpreted as augmenting the online algorithm's weakest resource – its knowledge about the future – by a certain amount of future information. Different models for lookahead are conceivable: as an example, the algorithm could foresee a certain restricted number of future requests or it can anticipate all future requests within a certain time window. Such algorithms are also known as *Rolling horizon algorithms*. Among the resource augmentation alternatives, the lookahead might be the most realistic variant since the assumption that an online algorithm has absolutely no information whatsoever about the upcoming sequence is indeed unnecessarily pessimistic in most applications. Research on lookahead for online algorithms has addressed different problems as, e. g., paging, metrical task systems, and online graph problems (a list of references can be found in [Alb98]) – and also scheduling problems. Mao and Kincaid [MK94] gave worst-case bounds for scheduling with lookahead on a single machine with release dates. Furthermore, Motwani, Saraswat, and Torng [MST98] consider a more abstract machine scheduling problem derived from scheduling requests on network reprographic machines (combined printer-copier-fax machines servicing a network) with the objective to minimize the makespan.

Comparative analysis. Koutsoupias and Papadimitriou [KP00b] introduced *comparative analysis.* The main idea is to compare the performance of two classes of algorithms. Thus, it captures the intention of performance analysis of algorithms in general, quite well. The comparative analysis is probably best viewed as a game between two players, namely the classes of algorithms, $\mathcal{A}$ and $\mathcal{B}$, where typically (by not necessarily) $\mathcal{A} \subset \mathcal{B}$. In order to demonstrate its superiority, player $\mathcal{B}$

proposes an algorithm $B \in \mathcal{B}$. Player $\mathcal{A}$ replies with its own algorithm $A \in \mathcal{A}$. Finally, $\mathcal{B}$ chooses an input instance $\mathcal{I}$, and gets paid the ratio $A(\mathcal{I})/B(\mathcal{I})$. The maximum ratio (resp. minimum ratio for maximization problems) over all input instance is called the *comparative ratio*. Notice, that if the algorithm set $\mathcal{A}$ embraces all online algorithms and $\mathcal{B}$ denotes the set of all algorithms then the comparative ratio coincides with the standard competitive ratio.

1.2.2 Stochastic Information

The study of optimization problems with stochastic information was initiated in the 1950's with the work of Dantzig [Dan55] and Beale [Bea55]. Inspired by real-world applications, their idea was to model uncertainty in the data by assuming that (a part of) the input is specified by a probability distribution, instead of deterministic data which is given in advance. Since then many different models for (stochastic) uncertainty in the input have been investigated. It is way beyond the scope of this thesis to give a satisfying overview of this field. Our primary goal is to highlight a model which distinguishes significantly from other models in this field – that is the *stochastic scheduling model*. This is the model we focus on in main parts of this thesis, and therefore we give a detailed introduction below. For an introduction to models and algorithms in the domain of stochastic programming we refer to the textbook of Birge and Louveaux [BL97].

From the view point of computational complexity, all stochastic models share a dilemma: they are often quite difficult; even extremely specialized (sub)problems are $\#\mathcal{P}$-complete.[4]

For our investigations on scheduling problems with stochastic input, we need basic concepts and techniques of probability theory which we assume the reader is familiar with. Therefore, we omit an introduction and refer to standard textbooks as for instance [Ros03] by Ross.

1.2.2.1 Stochastic Scheduling

In stochastic scheduling we assume uncertainty about job processing times. Any job j must be processed for P_j units of time, where P_j is a random variable. By $\mathbb{E}[P_j]$ we denote the expected value of the processing time of

[4]The complexity class $\#\mathcal{P}$ is the set of counting (or enumeration) problems naturally associated with the decision problems in the set $\mathcal{NP}$. Informally speaking, $\mathcal{NP}$-problems typically ask if there is *any* solution that satisfies certain constraints, whereas the corresponding $\#\mathcal{P}$-problems ask *how many* solutions exist. For a precise definition of this complexity class, see e. g., the book by Garey and Johnson [GJ79].

job j and by p_j a particular realization of P_j. We assume that all random variables of processing times are stochastically independent. This restriction on the probability functions is not part of the stochastic scheduling model; still, the independence of random variables is crucial for our and previously known approximation results. More explicitly we say that a stochastic scheduling instance $\mathcal{I}$ contains of n jobs with certain deterministic job data and the random variable P_j with the corresponding probability distribution.

The solution of a stochastic scheduling problem is not a simple schedule, but a so-called *scheduling policy*. The notion of scheduling policies as proposed by Möhring, Radermacher, and Weiss [MRW84, MRW85] states: A scheduling policy specifies *actions* at *decision times* t. An action is a set of jobs that is started at time t, and a next decision time $t' > t$ at which the next action is taken, unless some job is released or ends at time $t'' < t'$. In that case, t'' becomes the next decision time. In order to decide, a policy may utilize the complete information contained in the partial schedule up to time t, as well as information about unscheduled jobs that have arrived before t. It must not anticipate information about the future, such as the actual realizations p_j of the processing times of the jobs that have not yet been completed by time t; we say a stochastic scheduling policy must be *non-anticipatory*.

Simple examples show that, in general, no scheduling policy can yield an optimal schedule for each possible realization of processing times. Therefore, an *optimal policy* is defined as a non-anticipatory scheduling policy that minimizes the objective function value in expectation. That means, a policy Π^* with expected objective value $\mathbb{E}\,[\,\Pi^*(\mathcal{I})\,]$ on instance $\mathcal{I}$ is optimal if

$$\mathbb{E}\,[\,\Pi^*(\mathcal{I})\,] \;=\; \inf\{\,\mathbb{E}\,[\,\Pi(\mathcal{I})\,] \;\mid\; \Pi \text{ policy}\,\}\,, \qquad \text{for any instance } \mathcal{I}\,.$$

Even though this notion of optimality is quite mild, only few special problem classes exist for which optimal scheduling policies are known. In Chapter 3 we study stochastic scheduling problems and give an overview of results in that area in Section 3.2. Our investigations, however, focus on approximative policies.

Definition 1.4 (Approximative policy). *A stochastic scheduling policy* Π *is a* ρ–approximation *if, for some* $\rho \geq 1$*, and all instances* $\mathcal{I}$ *of the given problem,*

$$\mathbb{E}\,[\,\Pi(\mathcal{I})\,] \leq \rho\,\mathbb{E}\,[\,\textsc{Opt}(\mathcal{I})\,]\,.$$

Here, $\textsc{Opt}(\mathcal{I})$ *denotes an optimal stochastic scheduling policy on the given instance* $\mathcal{I}$*, assuming a priori knowledge of the set of jobs* $\mathcal{J}$*, the corresponding processing time distributions* P_j *and some characteristic deterministic* *job data. The value* ρ *is called the* performance guarantee *or* approximation factor *of policy* Π. *The policy* Π *is also called* ρ–approximative policy.

We emphasize the crucial difference of this performance guarantee to other measures, as e. g., the competitive ratio or the expected competitive ratio (defined below): the expected outcome of a policy is compared against an optimal scheduling policy which must be non-anticipative. Thus, it also *underlies uncertainty* about processing times and is not an optimal offline solution based on complete knowledge of the instance. On that account, the performance of a policy is not determined by some algorithm with magical powers but under equal conditions.

Note also the conceptual difference to average case analysis as described in the previous section. In that model, the instance is also drawn from a probability distribution. But the algorithm gains substantially more information: it learns the realized instance completely before its execution. Thus, a non-anticipativity constraint does not exist, whereas it is decisive in the stochastic scheduling model.

1.2.2.2 Stochastic Scheduling – An Alternative Analysis

A different way of analyzing stochastic scheduling has been proposed by Coffman and Gilbert [CG85], Scharbrodt, Schickinger, and Steger [SSS06], and by Souza and Steger [SS06a]. They compare the outcome of a scheduling policy to the optimal offline solution *per realization*, and take the expectation of this ratio on the basis of the given processing time distributions. Their objective is to minimize this so-called *expected competitive ratio* of a policy over all distributions of processing times. Thus, their analysis is different from the traditional stochastic scheduling model introduced above; in particular it is not based upon a comparison to the optimal stochastic scheduling policy. The underlying adversary is stronger than in traditional stochastic scheduling, yet they derive constant bounds on the expected competitive ratio for certain probability distributions of processing times.

Souza and Steger call this analysis adequately a "hybrid between the stochastic scheduling model and competitive analysis". The similarities to stochastic scheduling lay evidently in the stochastic problem data that are realized only by execution. The important difference is the performance evaluation by relating the outcome of the algorithm to an optimal offline solution for each instance as in competitive analysis; see Section 1.2.1.1. However, instead of taking the maximum value, they take the average over all instances weighted with a distribution specified by a weakened diffuse adversary. Thus, their analysis follows the diffuse adversary model by Koutsoupias and Papadimitriou [KP00b] introduced in Section 1.2.1.3.

1.2.2.3 Stochastic Combinatorial Optimization

Stochastic linear programming has been widely studied since its introduction in 1955 [Dan55, Bea55]. On the other hand, there has been reported only moderate success on stochastic integer (or mixed-integer) programming. Only in the past few years, stochastic versions of $\mathcal{NP}$-hard optimization problems have received more attention. In particular, the design and analysis of approximation algorithms for *two-stage (multi-stage) stochastic optimization with recourse* is a relatively recent but successful research direction. The fundamental idea behind the concept of *recourse* is the ability to take correcting actions after a random event has taken place. In a first stage, an algorithm has given distributional information about (parts of) the problem input and it fixes a set of so-called *first-stage* decisions. Then, the actual data are realized according to the given distribution and further *recourse* or *second-stage* decisions can be taken to satisfy the requirements for a feasible solution when combined with the first-stage decisions. The goal is to take the initial decisions such that the expected total cost are minimized. Hereby, the second-stage cost are typically more expensive than the first-stage actions, but on the other hand, they can be taken based on complete knowledge.

Different models within this framework have been successfully studied on various versions of classical combinatorial optimization problems as the Minimum Spanning Tree Problem, Shortest Path Problems, Steiner Trees, Vertex and Set Cover Problems, and Network Design Problems. The first result (as a technical report in 1999) is due to Dye, Stougie, and Tomasgard [DST03] followed by a series of papers [IKMM04, SS04, RS04, GPRS04], just to name a few. For a survey on models and techniques of this rapidly ongoing research we refer to Swamy and Shmoys [SS06b].

The crucial difference to the stochastic scheduling problem as introduced above is the following: The gain of additional information is completely independent of the (first stage) decision. In stochastic scheduling instead, the policy influences this procedure by processing a job or not; if a job does not get started then the policy will never learn about the realized processing time of this particular job.

Chapter 2

Online Scheduling

In this chapter, we deal with online algorithms for scheduling problems with the objective to minimize the total weighted completion time on single and on parallel machines. For independent jobs arriving online, we design and analyze online algorithms for both, the preemptive and the non-preemptive setting, that are straightforward extensions of the classical Smith ratio rule (Wspt algorithm). We also propose a modified, more dynamic version of this algorithm for the single-machine problem. For the scenario, in which there are given precedence constraints among jobs, we discuss a reasonable online model and give lower and upper bounds on the competitive ratio for scheduling without job preemptions.

2.1 Model and Preliminaries

In online scheduling we assume that jobs (requests) and their characterizing data become known to the scheduler only piecewise. Thus, an online algorithm must take scheduling decisions based only on the partial knowledge of the instance as it is given so far. In the *online-time* model jobs become known upon their release dates r_j, whereas in the *online-list* model, jobs are presented one-by-one, all at time zero. In the latter case, a job has to be scheduled immediately upon presentation before the next job can be seen, whereas in the online-time model the moment of taking the decision is left to the scheduler; compare with the general introduction on online paradigms in Section 1.2.1. Furthermore, we refer to Pruhs, Sgall, and Torng [PST04] for a recent overview on these two online scheduling models and to the survey by Sgall [Sga98].

We consider scheduling on a single and on parallel identical machines in both, the preemptive and the non-preemptive machine environments. The objective in all our problems is to minimize the total (weighted) sum of completion times. For an instance $\mathcal{I}$, consisting of the number of machines m, the set of jobs $\mathcal{J}$ together with their release dates r_j, weights w_j and processing times p_j, let $C_j^{\text{ALG}}(\mathcal{I})$ denote the completion times of jobs in the schedule obtained by the online algorithm ALG. Mostly, we write C_j^{ALG}, for short, and if it is clear from the context which algorithm we refer to, then we omit ALG. We let

$$\text{ALG}(\mathcal{I}) = \sum\nolimits_{j \in \mathcal{J}} w_j \, C_j^{\text{ALG}}(\mathcal{I})$$

denote the outcome of the algorithm ALG on instance $\mathcal{I}$.

In the following two sections we analyze online scheduling algorithms using standard competitive analysis. In Section 2.2, we assume that jobs are independent, meaning that there are no precedence relations between any of them, whereas in Section 2.3 a schedule must respect a given partial order on jobs.

2.2 SCHEDULING INDEPENDENT JOBS

We consider the problem of scheduling independent jobs meaning that there are no precedence relations among them. Usually one assumes that jobs arrive online over time (online-time model) and we follow this model. The reason is that examinations of the problem in the online-list model led to quite discouraging results. In fact, Fiat and Woeginger [FW99] proved that the online single-machine problem $1 \,|\, r_j \,|\, \sum C_j$ with jobs arriving one-by-one, does not allow a deterministic or randomized online algorithm with a competitive ratio of $\log n$. In addition, they also give a class of algorithms that yield a competitive ratio of $(\log n)^{1+\varepsilon}$ for any $\varepsilon > 0$.

We show in Section 2.2.3 that a natural extension of the classical WSPT rule (detailed introduction follows) to preemptive scheduling on identical parallel machines with release dates is 2-competitive, and this bound is tight. The idea is to interrupt currently active jobs of lower priority whenever new high-priority jobs arrive and not enough machines are available to accommodate the arrivals. We also briefly discuss a different, more dynamic version of the preemptive WSPT rule on a single machine in Section 2.2.2.

When preemption is not allowed, a WSPT-based scheme or extensions do not lead directly to an optimal schedule. Therefore, we first modify the release date of each job such that it is equal to a certain fraction of its processing time, if necessary, and then we apply an extended version of the WSPT rule. In Section 2.2.4 we analyze a family of such algorithms and show that the best one is 3.281-competitive. In this case, we cannot show that our analysis is tight, but the remaining gap is at most 0.5.

Recently, this last result has been improved by Correa and Wagner [CW05]; they give an LP-based 2.62-competitive algorithm. We still present our excelled result here, since it will be reused in a more general model in Chapter 4.

2.2.1 Previous Work

The literature reflects an enormous amount of research on algorithms for deterministic (online) scheduling problems. We begin our review with one of the most famous algorithms which also serves as a basis for many online and offline algorithms in various ways. This is also true for the algorithms we consider.

Smith's Wspt *Rule and Extensions*

One of the first articles on algorithms solving scheduling problems dates back to 1956; that is when Smith [Smi56] the probably best known and widely used *Weighted Shortest Processing Time* (Wspt) rule, which is also known as *Smith's rule*.

Algorithm 1: Weighted Shortest Processing Time (Wspt)

Sequence jobs in non-increasing order of their ratios of weight over processing time, w_j/p_j; where $w_j/0 := \infty$.

We call the ratio w_j/p_j also Wspt ratio of a job j. For convenience and since our algorithms are based on the Wspt rule in different ways, we assume here and in the forthcoming subsections that the jobs are indexed in the order of their original Wspt ratios so that $w_1/p_1 \geq w_2/p_2 \geq \cdots \geq w_n/p_n$. We also say that a job with a smaller index has higher priority than one with a larger index. Moreover, we define $w_j/0 := \infty$ throughout the thesis.

Smith's Wspt rule [Smi56] solves instances of the problem $1 \,|\, | \sum w_j C_j$ optimally. However, in an online setting this rule is not feasible anymore (if release dates are present). An online algorithm does not have the knowledge of future job arrivals and their ratios w_j/p_j. A natural online variant of the Wspt algorithm respects the availability of jobs in the online setting. To that end, let us call a job j *available* at time t if it is known to the online algorithm by time t and has not completed yet, that means $r_j \leq t$.

Algorithm 2: Online Wspt

Whenever the machine is idle, choose an available job with highest ratio of weight over processing time, w_j/p_j.

This "reformulation" of the classic Wspt rule as a feasible online algorithm is trivial and does not involve any change when applied in the offline setting. Therefore and in order to keep the presentation concise, we refer to the ONLINE WSPT algorithm also shortly as WSPT. Chou, Queyranne, and Simchi-Levi [CQSL06] have shown asymptotic optimality of this online algorithm for $1 \,|\, r_j \,|\, \sum w_j C_j$ under the assumptions that weights and processing times are bounded from above by a constant. They also extend this result to uniform parallel machines.

When preemption is allowed, the WSPT algorithm does not lead directly to a bounded competitive ratio since it does not preempt jobs; consider a simple example in which a job of high priority is released right after the start of a long lower-priority job. A natural extension of WSPT preempts a job j if a new job with a higher ratio w_j/p_j arrives.

Algorithm 3: PREEMPTIVE WSPT

At any point in time, schedule an available job with highest ratio of weight over processing time, w_j/p_j.

For $1 \,|\, r_j, pmtn \,|\, \sum w_j C_j$, Goemans, Wein, and Williamson noted that this algorithm is 2-competitive. (This note is cited as personal communication by Schulz and Skutella in [SS02a], complemented with a proof.) The preemptive parallel-machine algorithm we consider in Section 2.2.3 is a direct extension of this variation of Smith's rule to parallel machines and yields the same competitive ratio. Both results are tight.

Observe, that in case that all jobs are released at the same time, this algorithm reduces to Smith's WSPT rule and is optimal without preempting a job. If jobs have individual release dates but common weights, then the PREEMPTIVE WSPT algorithm can perform arbitrarily bad. Instead, Schrage's *Shortest Remaining Processing Time* (SRPT) rule solves the (online) problem $1 \,|\, r_j, pmtn \,|\, \sum C_j$ to optimality [Sch68]. In general, this latter algorithm distinguishes from the (PREEMPTIVE) WSPT rule significantly, in the sense, that it dynamically measures at any time t the actual "value" of a job depending of the progress it has made in processing up to time t, instead of considering its static processing time (or ratio w_j/p_j) as WSPT does.

In Section 2.2.2, we briefly discuss a natural algorithm that combines the ideas of PREEMPTIVE WSPT and SRPT; it schedules jobs in order of their dynamic ratios of weight over *remaining* processing time. We prove the same performance guarantee of 2 (as PREEMPTIVE WSPT yields) for the single-machine problem.

More Previous Work

In non-preemptive scheduling, every reasonable online algorithm has to make use of some kind of waiting strategy. Hoogeveen and Vestjens [HV96] yield a 2-competitive algorithm for the online problem $1 \,|\, r_j \,|\, \sum C_j$ by delaying the release date of each job j until time $\max\{r_j, p_j\}$. See also Vestjens PhD thesis [Ves97, Chap. 2] for a discussion of several delaying approaches for the single machine. Finally, Lu, Sitters, and Stougie [LSS03] assimilate these waiting strategies in a joint class of algorithms. Anderson and Potts [AP04] extended both, Hoogeveen and Vestjens' algorithm and its competitive ratio of 2, to the setting of arbitrary non-negative job weights. These results are best possible since Hoogeveen and Vestjens also proved that no non-preemptive deterministic online algorithm can achieve a competitive ratio better than 2.

Phillips, Stein, and Wein [PSW98] presented another online algorithm for the non-preemptive problem $1 \,|\, r_j \,|\, \sum C_j$, which converts a preemptive schedule into a non-preemptive one of objective value at most twice the value of the preemptive schedule. Since Schrage's Srpt rule [Sch68] works online and produces an optimal preemptive schedule for the single-machine problem, it follows that Phillips, Stein and Wein's algorithm has competitive ratio 2 as well. The conversion factor is $3-1/m$ if applied to identical parallel machines, but the corresponding preemptive problem is $\mathcal{NP}$-hard in this case. However, Chekuri, Motwani, Natarajan, and Stein [CMNS01] noted that by sequencing jobs non-preemptively in the order of their completion times in the optimal preemptive schedule on a single machine of speed m times as fast as that of any one of the m parallel machines, one obtains a $(3 - 1/m)$-competitive algorithm for the online version of $\mathrm{P} \,|\, r_j \,|\, \sum C_j$. For the same problem, Lu, Sitters, and Stougie [LSS03] gave a 2α-competitive algorithm, where α is the competitive ratio of the direct extension of the Srpt rule to identical parallel machines. Phillips, Stein, and Wein [PSW98] showed that this rule is 2-competitive, but a smaller value of α has not been ruled out.

For the corresponding preemptive and non-preemptive problems with arbitrary job weights, $\mathrm{P} \,|\, r_j, pmtn \,|\, \sum w_j C_j$ and $\mathrm{P} \,|\, r_j \,|\, \sum w_j C_j$, respectively, Hall, Schulz, Shmoys, and Wein [HSSW97] gave a $(4 + \varepsilon)$-competitive algorithm as part of a more general online framework. Recently, Correa and Wagner [CW05] introduced for $\mathrm{P} \,|\, r_j \,|\, \sum w_j C_j$ the currently best known non-preemptive algorithm which is 2.62-competitive. They apply the Preemptive Wspt algorithm to solve a subproblem and yield a solution which is known to be optimal [Goe96] for an LP-relaxation of the original problem. Based on the (partially) produced schedule, they apply list scheduling in order of α-points. This technique has been applied successfully in earlier work as, e. g., in [GQS$^+$02, SS02a, SS02b] and [CQSL06].

In [GQS$^+$02], Goemans et al. apply this approach in the non-preemptive single-machine setting and provide a deterministic 2.41-competitive and a randomized 1.69-competitive algorithm. Schulz and Skutella [SS02a] take a similar approach in order to achieve a randomized 4/3-competitive algorithm for the preemptive problem $1 \,|\, r_j, pmtn \,|\, \sum w_j C_j$. The currently best known deterministic online algorithm for this problem is due to Sitters [Sit04], who gave a 1.56-competitive algorithm. Let us finally mention that the currently best randomized online algorithms for the preemptive and non-preemptive scheduling problem on multiple machines as considered here have (in expectation) competitive ratio 2; see [SS02b].

On the side of negative results, Vestjens [Ves97] proved a universal lower bound of 1.309 for the competitive ratio of any deterministic online algorithm for $\mathrm{P} \,|\, r_j \,|\, \sum C_j$ which can be raised with decreasing number of machines such that it reaches 2 for $m = 1$ [HV96]. Any randomized algorithm in this setting has a competitive ratio of at least 1.157 on multiple machines as shown by Seiden [Sei00]; whereas, on a single machine, there is an improved lower bound of $e/(e-1) \approx 1.58$ by Stougie and Vestjens [SV97]. The only lower bound for preemptive scheduling that we are aware of is 1.073 by Epstein and van Stee [EvS03] derived for the problem $1 \,|\, r_j, pmtn \,|\, \sum w_j C_j$.

Finally, let us annotate that the off-line problems, $\mathrm{P} \,|\, r_j, pmtn \,|\, \sum w_j C_j$ and $\mathrm{P} \,|\, r_j \,|\, \sum w_j C_j$, including their corresponding single-machine versions are are well understood; both problems have a polynomial-time approximation scheme [ABC+99] and they are known to be $\mathcal{NP}$-hard even if there is only a single machine available [LLLR84, LRB77].

2.2.2 An Alternative Preemptive Single-Machine Algorithm

The Preemptive Wspt rule schedules jobs preemptively in non-increasing order of their static ratios w_j/p_j. Thus, it considers solely those information about a job which were available from the release date on, and it completely ignores the progress of jobs. In fact, jobs that have completed most of their processing requirement get interrupted if their original w_j/p_j-ratio is rather small compared to the ratios of newly arriving jobs.

We propose a dynamic rule which respects the current state of jobs in the system in the spirit of Schrage's Srpt rule [Sch68] for the problem without weights, which considers the *remaining processing time* of a job. Our algorithm respects the ratio of weight to remaining processing time of jobs and we name it *Weighted Shortest Remaining Processing Time Rule* (Wsrpt).

We prove a competitive ratio of 2 which is of minor significance as it does not improve the performance guarantee of Preemptive Wspt. Moreover, Sitters gave a 1.56-approximation for the same online problem in his PhD thesis [Sit04]. Nevertheless, we introduce this algorithm for two reasons: One is, that ideas on the design and analysis as well as the coherence of Preemptive Wspt and Wsrpt will recur in a way in the stochastic scheduling environment in Chapter 3. The second reason is that we conjecture that the true performance of Wsrpt is much better than we actually guarantee. Indeed, the best lower bound in the performance of Wsrpt leaves a quite large gap to the upper bound given above.

Merging ideas of the Srpt algorithm and the Preemptive Wspt algorithm, we define the Wsrpt rule as follows,

Algorithm 4: Wsrpt

```
Schedule at any time $t$ an uncompleted job $j$ with the highest
ratio $w_j/p_j(t)$ among the available jobs, where $p_j(t)$ denotes the
remaining processing time of job $j$ at time $t$. Ties are broken by
preferring the job with the smallest index.
```

The algorithm takes at any time only decisions that are available in this moment in the online environment. Notice, that in the case of trivial release dates, our algorithm sequences jobs in Wspt order; hence, it is optimal for the problem $1 \,|\, \,|\, \sum w_j C_j$. If there are non-trivial release dates, then Wsrpt performs optimally, again, with respect to the sum of completion times, i. e., the online version of the problem $1 \,|\, r_j, pmtn \,|\, \sum C_j$, because it behaves exactly as the optimal Srpt rule.

Nevertheless, simple examples show that on general problem instances, Wsrpt does not yield an optimal schedule. The following performance bound is due to Megow, Sitters, and Stougie [MSS03].

Theorem 2.1. *The competitive ratio of the online Algorithm* Wsrpt *is at most* 2 *for the problem* $1 \,|\, r_j, pmtn \,|\, \sum w_j C_j$.

Proof. Consider a job j and the time interval $(r_j, C_j]$ in which it is being processed – possibly preempted by jobs with higher priority, that is in this proof, jobs with a higher dynamic ratio of $w_j/p_j(t)$ at time t.

Recall that jobs are indexed in non-increasing order of their static Wspt ratios. Let $H(j)$ denote the set of jobs with index at most j and that are being processed in the time interval $(r_j, C_j]$. Such a job $h \in H(j)$, for $h \neq j$, preempts or postpones job j and we say, that it *disturbs* the processing of j. Furthermore, let $L(j)$ denote the set of jobs that have a larger job index than j but, due to processing before j is available, their dynamic ratio of weight over remaining processing time has increased such that it is at time r_j larger than j's ratio; more formally, $L(j) := \{\ell \in \mathcal{J} \,|\, \ell > j \wedge w_\ell/p_\ell(r_j) > w_j/p_j\}$. Such a job $\ell \in L(j)$ also disturbs the processing of job j if it is available in the time interval $(r_j, C_j]$.

Observe, that all jobs that disturb the processing of job j are contained in the two disjunctive sets $H(j)$ and $L(j)$, respectively. The amount by which a job $k \in H(j) \cup L(j)$ postpones the completion of job j is its remaining processing time $p_k(r_j)$ at the release date of j. Hence, we can express the value of the schedule obtained by the WSRPT algorithm as

$$\sum_j w_j C_j \;=\; \sum_j w_j \big(r_j + \sum_{h \in H(j)} p_h(r_j) + \sum_{\ell \in L(j)} p_\ell(r_j) \big) \,. \tag{2.1}$$

Since jobs are indexed in non-increasing order of their WSPT-ratios, we have that

$$\sum_{h \in H(j)} p_h(r_j) \;\leq\; \sum_{k \leq j} p_k \,, \qquad \text{for all jobs } j \,. \tag{2.2}$$

Using this bound, we increase the sum by all jobs that have smaller index than j (and thus a higher WSPT-ratio) but do not actually disturb its processing. Before applying this bound, observe, that the ratio $w_j/p_j(t)$ of any job j is non-decreasing with time t. Therefore, on a single machine, the priority order of two jobs will not change with time, that means if a job j has a higher ratio than another job at some time t, then this will be the case for any time $t' \geq t$ until j has completed. Thus, for any job $\ell \in L(j)$ which has by definition a larger index than j, we have $j \notin H(\ell)$. Thus, in summation we yield a tighter version of the bound in (2.2),

$$\sum_j w_j \sum_{h \in H(j)} p_h(r_j) \;\leq\; \sum_j w_j \sum_{k \leq j} p_k - \sum_j \sum_{\ell \in L(j)} w_\ell \, p_j \,. \tag{2.3}$$

Hence, we can bound the value of the WSRPT schedule in (2.1) using (2.3) in the following way,

$$\begin{aligned} \sum_j w_j C_j \;&\leq\; \sum_j w_j \,\big(\, r_j + \sum_{k \leq j} p_k \,\big) + \sum_j \sum_{\ell \in L(j)} \big(w_j \, p_\ell(r_j) \;-\; w_\ell \, p_j \,\big) \\ &<\; \sum_j w_j \,\big(\, r_j + \sum_{k \leq j} p_k \,\big) \\ &\leq\; 2\,\textsc{Opt} \,. \end{aligned}$$

The second inequality is due to the fact that by definition of the set $L(j)$ holds for any j and $\ell \in L(j)$ that $w_j \, p_\ell(r_j) - w_\ell \, p_j \;<\; 0$. The upper bound of twice an optimal value follows immediately: firstly, the weighted sum of release dates is a trivial lower bound on the optimum, and secondly, $\sum_j w_j \sum_{k \leq j} p_k$ is the objective function value of the optimal WSPT schedule on the corresponding instance of the relaxed problem $1 \,|\, |\sum w_j C_j$. Since this is indeed a relaxation of our preemptive problem with release dates, the theorem follows. $\square$

As mentioned earlier, we strongly conjecture that the true competitive ratio of Wsrpt is much better than the previous theorem guarantees. The following lower bound on the performance guarantee deviates remarkably from the proven upper bound.

Theorem 2.2. *The Algorithm* Wsrpt *does not have a competitive ratio less than* 1.21057 *for the scheduling problem* $1 \,|\, r_j, pmtn \,|\, \sum w_j C_j$.

Proof. The *bad instance* consists of $k+2$ jobs: a high priority job h with unit weight and unit processing requirement, a low priority job ℓ of length p_ℓ and unit weight and k small jobs of length ε. The job ℓ and the first small job are released at time 0 followed by the remaining small jobs at times $r_j = (j-1)\varepsilon$ for $j = 2, \dots, k$ whereas the high priority job is released at time $r_h = p_\ell - 1$. The weights of the small jobs are $w_j = \varepsilon/(p_\ell - (j-1)\varepsilon)$ for $j = 1, \dots, k$. We choose ε such that all small jobs could be completed until time r_h, i. e., $\varepsilon = r_h/k = (p_\ell - 1)/k$.

W.l.o.g. we can assume that Wsrpt starts processing job ℓ at time 0. Note that the weights of the small jobs are chosen such that the ratio of weight over remaining processing time of job ℓ at the release date of a small job equals the ratio of the newly released small job, and thus Wsrpt does not preempt ℓ until at time $r_h = p_\ell - 1$ when job h is released and starts processing. After its completion, job j is finished, followed by the small jobs $\ell, \ell-1, \dots, 1$. The value of the achieved schedule is

$$2p_\ell + 1 + \sum_{i=1}^{k} (p_\ell + 1 + i\varepsilon) \frac{\varepsilon}{p_\ell - (k-i)\varepsilon} \,.$$

An optimal policy, instead, processes first all small jobs followed by the high priority job and finishes with the job ℓ. The value of such a schedule is

$$\sum_{i=1}^{k} i\varepsilon \frac{\varepsilon}{p_\ell - (i-1)\varepsilon} + 3p_\ell \,.$$

If the number of small jobs, k, tends to infinity then the performance ratio of Wsrpt is no less than

$$\frac{p_\ell(3 - \log \frac{1}{p_\ell - 1} + \log \frac{p_\ell}{p_\ell - 1})}{1 + 2p_\ell + p_\ell \log p_\ell} \,,$$

which gives the lower bound of 1.21057 for $p_\ell \approx 5.75$. $\square$

2.2.3 Preemptive Parallel-Machine Scheduling

We consider the following natural extension of the single-machine Preemptive Wspt rule to identical parallel machines.

Algorithm 5: P-Wspt

At any point in time, schedule the m jobs with the highest Wspt ratios among the available, not yet completed jobs (or fewer if less than m incomplete jobs are available). Interrupt the processing of currently active jobs, if necessary. In case of ties prefer jobs with the smaller index.

The algorithm works online since the decision about which job to run at any given point in time t is just based on the set of available jobs at time t. In fact, it only depends on the priorities (Wspt ratios) of the available jobs. In particular, Algorithm P-Wspt also operates in an environment in which actual job weights and processing times may not become available before the completion of the jobs, as long as the jobs' priorities are known at their respective release dates.

Theorem 2.3. *The online Algorithm* P-Wspt *produces a solution with a value at most twice the optimal off-line value for* $\mathrm{P} \mid r_j, pmtn \mid \sum w_j C_j$.

Proof. Consider the time interval $(r_j, C_j]$ for an arbitrary but fixed job j. We partition this interval into two disjunctive sets of subintervals, which we call $T(j)$ and $\overline{T}(j)$, respectively. We let $T(j)$ contain the subintervals in which job j is being processed; $\overline{T}(j)$ denotes the set of remaining subintervals. Note that no machine can be idle during the subintervals belonging to $\overline{T}(j)$. Since the algorithm processes job j after its release date r_j whenever a machine is idle, we obtain

$$C_j \leq r_j + |T(j)| + |\overline{T}(j)| \ ,$$

where $|\cdot|$ denotes the sum of the lengths of the subintervals in the corresponding set.

The overall length of $T(j)$ is clearly p_j. Only jobs with a higher ratio of weight to processing time than j can be processed during the intervals of the set $\overline{T}(j)$, because the algorithm gives priority to j before scheduling jobs with lower ratio. In the worst case, that is when $|\overline{T}(j)|$ is maximal, all jobs with higher priority than j are being processed in the subintervals of this set. Then $|\overline{T}(j)| = (\sum_{k<j} p_k)/m$, and we can bound the value of the P-Wspt schedule as follows:

$$\sum_j w_j C_j \ \leq \ \sum_j w_j(r_j + p_j) + \sum_j w_j \sum_{k<j} \frac{p_k}{m} \, .$$

Since the completion time C_j of a job j is always at least as large as its release date plus its processing time, $\sum_j w_j(r_j + p_j)$ is obviously a lower bound on the value of an optimal schedule. Moreover, $\sum_j w_j \sum_{k \leq j} p_k/m$ is the objective function value of an optimal solution to the corresponding instance of the relaxed problem $1 \,|\, |\sum w_j C_j$ on a single machine with speed m times the speed of any of the identical parallel machines. As this problem is indeed a relaxation of the scheduling problem considered here, we can conclude that the P-Wspt algorithm is 2-competitive. $\square$

A family of instances provided by Schulz and Skutella [SS02a] shows that this result cannot be improved. In fact, for $m = 1$, P-Wspt coincides with the preemptive single-machine algorithm studied in their paper. Taking m copies of Schulz and Skutella's instance yields the following result.

Corollary 2.4. *The competitive ratio of the Algorithm* P-Wspt *is not better than* 2 *for the online problem* $\mathrm{P} \,|\, r_j, pmtn \,|\, \sum w_j C_j$*, for any given number of machines.*

Proof. We include a (different) proof for the sake of completeness. We consider an instance that is slightly different from the one given in [SS02a]. It consists of m copies of $n+1$ jobs with $w_j = 1$, $p_j = n - j/n$ and $r_j = jn - j(j+1)/(2n)$ for all $0 \leq j \leq n$. Algorithm P-Wspt preempts any job when it has left just $1/n$ units of processing time and finishes it only after all jobs with a larger release date have been completed. The value of this schedule is $m\,(\sum_{j=0}^{n}(r_n + p_n + j/n))$. An optimal off-line algorithm does not preempt any job and yields a schedule of value $m\,(\sum_{j=0}^{n}(r_j + p_j + j/n))$. A simple calculation shows that the ratio of the values of the P-Wspt schedule and the optimal schedule goes to 2 when n goes to infinity. $\square$

Of course, Theorem 2.3 subsumes the scheduling problem with equal job weights, $\mathrm{P} \,|\, r_j, pmtn \,|\, \sum C_j$, as a special case. Corollary 2.4 holds also in the unweighted setting because its proof utilizes an instance with jobs that have equal weights. Thus, this extension of the 2-competitive single-machine Shortest Processing Time (Spt) rule to the parallel-machine setting has the same competitive ratio as the analogous extension of Schrage's optimal single-machine Srpt rule [PSW98].

2.2.4 Non-Preemptive Parallel-Machine Scheduling

When preemption is not allowed, a straightforward extension of the P-Wspt rule, as introduced in the previous section, is to start the currently available job of highest priority whenever a machine becomes idle. However, neither this rule nor its extensions lead directly to a bounded competitive ratio. In fact, consider a single-machine instance in which a job of high priority is released right after the start of a long lower-priority job, then the performance of such a non-preemptive Wspt algorithm cannot even be bounded by the number of jobs.

Example 2.5. *We have two jobs. The first job is released at time* 0 *and has processing time* $p_1 = k$ *and weight* $w_1 = 1$*. Only one time unit later, job* 2 *is released with unit processing time and weight* $w_2 = k$*.*

Wspt *schedules the jobs in order* $1 \prec 2$ *and yields an objective value of* $k^2 + 2k$*, whereas in an optimal schedule the machine is left idle for the first unit length time interval in order to start processing job* 2 *at its release date, followed by job* 1 *starting at time* 2*. The ratio of the values obtained by* Wspt *and an optimal algorithm is* $(k^2 + 2k)/(3k + 2)$*, which is unbounded if* k *tends to infinity.*

Every reasonable online algorithm for non-preemptive scheduling has to make use of some kind of waiting strategy. We refer the reader to Vestjens' PhD thesis [Ves97, Chap. 2] and the paper by Lu, Sitters and Stougie [LSS03] for discussions of related techniques for the single machine. Here, we apply the idea of delaying release dates to the parallel-machine problem.

First, we modify the release date of each job such that it is equal to a certain fraction of its processing time, if necessary. If we now start a long job j and a high-priority job becomes available shortly thereafter, the badly chosen time point for starting j can be accounted for by the fact that the high-priority job has a release date or processing time at least as large as a fraction of p_j. Therefore, its delay is bounded by its own contribution to the objective function in any feasible schedule.

Algorithm 6: α-Shift-Wspt

```
Modify the release date of every job $j$ to $r'_j$, where $r'_j$ is some value
between $\max\{r_j, \alpha\, p_j\}$ and $r_j + \alpha\, p_j$, for some $\alpha \in (0, 1]$.
Whenever a machine becomes idle, choose among the available jobs
a job $j$ with highest priority and schedule it on the idle machine.
```

Note that this is indeed an online algorithm; we will later choose α to minimize the corresponding competitive ratio. Moreover, for $m = 1$ and $\alpha = 1$, Algorithm α-Shift-Wspt is identical to the algorithm proposed by Lu, Sitters, and Stougie in [LSS03] for $1 \,|\, r_j \,|\, \sum C_j$. The idea of shifting release dates to ensure that the processing time of any one job is not too large compared to its arrival time is also present in the proposed approximation schemes for the offline variant of this problem [ABC+99].

In the analysis of the α-Shift-Wspt algorithm, we make use of the following lower bound on the optimal value of the relaxed problem with trivial release dates, which is due to Eastman, Even, and Isaacs [EEI64].

Lemma 2.6. *The value of an optimal schedule for an instance of the scheduling problem* $\mathrm{P}\,|\,|\sum w_jC_j$ *is bounded from below by*

$$\sum_{j=1}^{n} w_j \sum_{k\le j} \frac{p_k}{m} + \frac{m-1}{2m}\sum_{j=1}^{n} w_j p_j .$$

Let us now analyze the performance of the α-Shift-Wspt algorithm.

Theorem 2.7. *For the scheduling problem* $\mathrm{P}\,|\,r_j\,|\sum w_jC_j$*, the online Algorithm* α-Shift-Wspt *has a competitive ratio of less than*

$$2 \;+\; \max\{\frac{1}{\alpha},\, \alpha \;+\; \frac{m-1}{2m}\}\,.$$

Proof. The algorithm schedules a job j at time r'_j if a machine is idle and j is the job with the highest ratio of weight to processing time among all available jobs; otherwise, j has to wait for some time. The waiting time for j after r'_j is caused by two types of jobs: jobs with lower priority that started before time r'_j, and jobs with higher priority. Note that the algorithm does not insert idle time on any machine in the time interval between r'_j and the start of job j.

Clearly, every machine has at most one low-priority job $\ell > j$ that is running at time r'_j and before the start of job j. By construction, such a job ℓ satisfies $\alpha\, p_\ell \le r'_\ell < r'_j$. Thus, it is completed by time $(1+1/\alpha)r'_j$. Consequently, any job that is running between $(1+1/\alpha)r'_j$ and the start of job j must have a higher priority than j. The total processing time of these jobs is bounded by $\sum_{h<j} p_h$. As a result, one of the m parallel machines finishes processing jobs of this kind after at most $\sum_{h<j} p_h/m$ time units. Hence,

$$\begin{aligned}
C_j \;&<\; \Big(1+\frac{1}{\alpha}\Big)\, r'_j \;+\; \sum_{h<j}\frac{p_h}{m} + p_j\\
&\le\; (1+\frac{1}{\alpha})(r_j \;+\; \alpha p_j) \;+\; \frac{m-1}{2m}p_j + \sum_{h\le j}\frac{p_h}{m} + \frac{m-1}{2m}p_j\\
&=\; \Big(\;\frac{r_j}{\alpha} + \big(\alpha + \frac{m-1}{2m}\big)\; p_j\;\Big) \;+\; (r_j \;+\; p_j) \;+\; \sum_{h\le j}\frac{p_h}{m} \;+\; \frac{m-1}{2m}\; p_j\,.
\end{aligned}$$

Thus, Algorithm α-SHIFT-WSPT generates a schedule of value

$$\sum_j w_j C_j \;<\; \Big(1 + \max\big\{\frac{1}{\alpha}, \alpha + \frac{m-1}{2m}\big\}\Big)\sum_j w_j(r_j+p_j) \\ + \sum_j w_j\Big(\sum_{h\le j}\frac{p_h}{m} + \frac{m-1}{2m}\,p_j\Big).$$

The proof is completed by applying two lower bounds on the optimal value: first, the trivial lower bound $\sum_j w_j(r_j + p_j)$, and second, the lower bound presented in Lemma 2.6. □

A simple calculation shows that the minimum of the expression $\max\{1/\alpha, \alpha + (m-1)/(2m)\}$ is attained at

$$\alpha \;=\; \frac{1}{4m}\,(1 - m + \sqrt{16m^2 + (m-1)^2}\,) \;=:\; \alpha_m\,.$$

In particular, $\alpha_1 = 1$.

Corollary 2.8. *The Algorithm α-SHIFT-WSPT with $\alpha = \alpha_m$ is $(2 + 1/\alpha_m)$-competitive. The value of this increasing function of m is 3 for the single-machine case and has its limit at $(9+\sqrt{17})/4 < 3.281$ for $m \to \infty$.*

Lemma 2.9. *Algorithm α-SHIFT-WSPT cannot achieve a better competitive ratio than*

$$\max\{2+\alpha, 1+\frac{1}{\alpha}\} \;\ge\; 2 + \frac{\sqrt{5}-1}{2}$$

for $\alpha \in (0,1]$, on any number of machines.

Proof. We give two instances from which the lower bound follows. Consider $2m$ jobs released at time 0; half of the jobs are of type I and have unit processing times and weights ε, whereas the other half of the jobs, type II, have processing time $1+\varepsilon$ and unit weight. The algorithm modifies the release dates and schedules jobs of type I at time $t = \alpha$ first, one on each machine, followed by jobs of type II. The value of the schedule is $m(\alpha+1)\varepsilon + m(\alpha+2+\varepsilon)$. In the optimal schedule, jobs of type II start processing first, at time $t = 0$, such that the value of the schedule is $m(1+\varepsilon) + m(2+\varepsilon)\varepsilon$. For $\varepsilon \to 0$, the ratio between the value of the α-SHIFT-WSPT schedule and the optimal schedule goes to $2 + \alpha$.

The second instance consists again of $2m$ jobs: half of the jobs are of type I and have release dates $1+\varepsilon$, processing times ε and weights $1/m$, whereas the other half of the jobs, type II, are released at time 0 and have processing time $1/\alpha$ and zero weight. α-Shift-Wspt starts scheduling jobs at time 1 and obtains a solution with a value of at least $1 + 1/\alpha + \varepsilon$. The value of the optimal schedule is $1 + 2\varepsilon$. □

For the special choice $\alpha = \alpha_m$, the first lower bound is tighter and it follows more concretely.

Corollary 2.10. *The performance guarantee of* α-Shift-Wspt *with* $\alpha = \alpha_m$ *is not better than* $(1 + 7m + \sqrt{16m^2 + (m-1)^2})/(4m)$ *for instances with* m *machines. This means that the above analysis for this specific algorithm has a gap of at most* $(m-1)/2m < 0.5$*, and it is tight for the single-machine problem.*

2.3 Scheduling Jobs with Precedence Constraints

Consider the online problem of scheduling jobs with precedence constraints. Such job dependencies require a refinement of the online paradigms as introduced in Section 1.2.1. Recall, that in both (clairvoyant) online models, the online-time and the online-list model, we generally assume that all data about a request are revealed as soon as the request becomes known. Interpreted for the online scheduling problem with precedence constraints, this means that whenever a job arrives, a scheduler learns about its weight and processing time and – most importantly – about job dependencies. However, these dependencies occur between *two* jobs and it is not clear which job gets assigned the information about such a bilateral relation.

Certainly, there are various options to specify the information that should be revealed at job arrival. Nevertheless, we use a different model: a scheduler learns about the existence of a job when all its predecessors have completed their processing. Then, its weight, processing time and all precedence relations to predecessors become known. One of the first publications we know of which applies this model, is by Feldmann, Kao, Sgall, and Teng [FKST98]. In contrast to the standard online paradigms as introduced in Section 1.2.1, the moment of unveiling jobs and all their data is designated by other *job completions*, more precisely, by the completion times of jobs it depends on.

Our investigations on this online model focus on non-preemptive scheduling with the objective to minimize the sum of (weighted) completion times. We derive in Section 2.3.2 matching upper and lower bounds for the online problem on a single machine. We show that the basic Spt Algorithm achieves the best possible performance for this problem – albeit the competitive ratio in order of n is discouraging. In Section 2.3.3 we derive bounds for the corresponding problem on identical parallel machines.

	lower bound	upper bound	AND/OR-prec [EKM04]
$1 \mid prec \mid \sum C_j$	$\frac{2}{3}\sqrt{n} - 1$	$\sqrt{2}\sqrt{n}$	$2\sqrt{n}$
$1 \mid prec \mid \sum w_j C_j$	$n - 1$	n	n
$\mathrm{P} \mid prec \mid \sum C_j$	$\frac{2}{3}\sqrt{\frac{n}{m}} - 1$	$\sqrt{2m}\sqrt{n}$	(n)
$\mathrm{P} \mid prec \mid \sum w_j C_j$	$\frac{n-1}{m}$	n	(n)

Table 2.1: Bounds on the competitive ratio of any deterministic online algorithm (lower bounds) and on the competitive ratio of SPT (upper bounds) for scheduling jobs with precedence constraints online on single and parallel machines. Lower bounds for randomized algorithms can be derived, with value one half of the deterministic bound. Offline considerations in [EKM04] for problems with AND/OR-precedence relations transfer to our online setting and inspire our new results; approximation factors they proved (and suggested) are given in the third column.

In our examination of the online environment it appeared that there exists relevant work on the offline problem of scheduling jobs with generalized precedence constraints, the so called AND/OR-precedence relations. While ordinary precedence constraints force a job to wait for the completion of *all* its predecessors (represented as an AND-node in the corresponding precedence graph), there is an additional relaxed waiting condition that allows a job to start after at least one of its predecessors has completed (OR-node). Clearly, ordinary precedence constraints are contained as a special case in AND/OR-precedence constraints. Erlebach, Kääb, and Möhring [EKM04] derived approximation guarantees for the offline scheduling problem with generalized job dependencies which translate into competitiveness results for our online framework. Their work also inspires our extended results. Let us mention at this point, that the analysis of our online algorithms does not need significant new techniques since borrowing major ideas from [EKM04] turned out successful. We emphasize that the intention of this section is to give a first rounded off presentation of online results for scheduling jobs with precedence constraints. Table 2.1 gives a summary of our results. They are still valid when considering the more general AND/OR-precedence constraints although this is not focus of our work.

2.3.1 Previous Work

The problem of online scheduling to minimize the sum of (weighted) completion times is quite intensively studied; see an overview for scheduling independent jobs in Section 2.2.1. Nevertheless, we are not aware of any publication on this problem (under any online paradigm) that deals with additional job dependencies.

On the other hand, there has been done more work on this particular problem class with respective to a different objective, which is minimizing the makespan. Probably, one of the earliest publications using the model where jobs become known after all its predecessors have completed, is by Feldmann, Kao, Sgall, and Teng [FKST98]. They consider scheduling of parallel jobs that are processed by more than one machine at the same time (malleable jobs). Considering jobs that are processed by at most one machine at the time, Azar and Epstein [AE02] derived a lower bound of $\Omega(\sqrt{m})$ for the competitive ratio of any deterministic or random online algorithm that schedules jobs (preemptively or not) on m related machines.

The offline problem of scheduling jobs with precedence constraints to minimize the sum of (weighted) completion times has been shown to be $\mathcal{NP}$-hard [Law78, LR78] even if there is only a single processor. This classical problem, $1\,|\,prec\,|\sum w_jC_j$, has attracted research for more than thirty years and a vast amount of results has been obtained on this problem – just recently there has been made significant progress.

Several classes of scheduling algorithms are known that achieve an approximation ratio of 2 in polynomial time [Pis03, Sch96b, HSSW97, CH99, CM99, MQW03]; excluding the first reference, all of them are based on linear programming relaxations; we refer to [CS05] for a nice overview. Correa and Schulz recently showed in [CS05] that all known 2-approximations follow Sidney's decomposition [Sid75] from 1975, and thus belong to the class of algorithms described by Chekuri and Motwani [CM99] and Margot, Queyranne, and Wang [MQW03]. Very recently, Ambühl and Mastrolilli [AM06] proved the meaningful fact that the scheduling problem $1\,|\,prec\,|\sum w_jC_j$ is a special case of the classical vertex cover problem.

Despite the complexity of the general problem, special cases are known to be solvable in polynomial time. For a long time, the most important class of polynomially solvable problems consisted of series-parallel precedence constraints. First, Lawler [Law78] presented an optimal algorithm with running time $\mathcal{O}(n \log n)$. Later, Goemans and Williamson [GW00] gave an elegant alternative proof for the correctness of Lawler's algorithm using a two-dimensional Gantt chart. The recent work by Ambühl and Mastrolilli [AM06] combined with Correa and Schulz [CS05] also implies that instances with two-dimensional precedence constraints are optimally solvable in polynomial time. Let us solely emphasize that two-dimensional precedence constraints are a generalization of series-parallel precedence constraints and refer to [CS05, AM06] and to Möhring [Möh89] for a profound treatment.

2.3.2 SCHEDULING ON A SINGLE MACHINE

Consider scheduling of jobs with precedence constraints online and non-preemptively on a single processor. In the previous sections on scheduling of independent jobs, we have derived good performance guarantees for online version of the classic WSPT algorithm and various extensions. If jobs must obey precedence relations which are revealed after predecessor completion, no such variant of the ONLINE WSPT algorithm yields a bounded performance. Note, that on a single machine no waiting time will reveal new information on the online sequence and is therefore superfluous.

Example 2.11. *Consider an instance that consists of the following three jobs. The first job has processing time $p_1 = k$ and weight $w_1 = 1$. Jobs 2 and 3 must obey the precedence constraint $2 \prec 3$; they have processing times $p_2 = 1$ and $p_3 = \varepsilon$, respectively, and their weights are $w_2 = \varepsilon$ and $w_3 = k$. Let k and ε be such that $\varepsilon << k$ and the ratios of weight over processing time of the two independent jobs 1 and 2 fulfill $w_1/p_1 > w_2/p_2$, that means $\varepsilon < 1/k$.*

Then the ONLINE WSPT *algorithm schedules the jobs in increasing order of their indeces, $1, 2$, and 3 achieving an objective value of $k^2+2k+\varepsilon(2k+1)$. In contrast, an optimal schedule has job 2 being processed first, followed by 3 and 1, and yields thus a value of $2k+1+\varepsilon(k+2)$. If ε tends to 0, then the ratio of values of the* WSPT *schedule and an optimal schedule goes towards $\frac{k(k+2)}{2k+1}$ which is unbounded for increasing k.*

An even more basic online algorithm than WSPT is the *Online Shortest Processing Time* (SPT) rule; it is simply the ONLINE WSPT algorithm assuming equal job weights. Even though it seems counter intuitive to ignore known job weights, this algorithm yields a competitive ratio that matches the lower bound on the performance guarantee for any deterministic online algorithm on a single machine. In fact, Erlebach et al. [EKM04] analyze the performance of the same algorithm in an offline setting with generalized job dependencies, the AND/OR-precedence constraints. The offline version of our problem with standard precedence constraints is a special case of the problem they consider. Moreover, their variant of the SPT algorithm considers only jobs that are *available* for processing according to the precedence constraints. Thus, it coincides with the online SPT algorithm that knows of jobs only after all predecessors have finished. Therefore, the approximation results translate into competitiveness results in our online setting.

Theorem 2.12 (Erlebach, Kääb, and Möhring [EKM04])**.** *The online version of the* SPT *algorithm has a competitive ratio of n solving the scheduling* *problem* $1\,|\,prec\,|\sum w_jC_j$*. If all jobs have equal weights, then* Spt *is* $2\sqrt{n}$*-competitive. This result is tight up to the constant factor of* 2*.*

These bounds are not only tight for the performance of Spt, they are also best possible (up to a constant factor) for any deterministic or randomized online algorithm for the considered problems. In the following section, we complement the upper bounds by matching lower bounds on the competitive ratio (up to a constant factor) by considering the general setting of $m \geq 1$ machines. All our lower bounds hold against the oblivious adversary. Since this is the weakest adversary, they also hold for any stronger one. An introduction of the adversary model can be found in Section 1.2.1.

2.3.3 Scheduling on Parallel Machines – General Job Weights

Consider now the problem of scheduling jobs with precedence relations in an online fashion on multiple machines. An immediate extension of the Online Spt algorithm to parallel machines is such that at any time when a machine is idle a job is chosen with shortest processing time p_j among all available, unfinished jobs.

As mentioned in the previous section, Erlebach et al. [EKM04] analyzed the performance of the Spt algorithm in a more general environment where jobs have And/Or-precedence constraints. They claim that the general approximation guarantee of n for the Spt algorithm obtained on a single machine directly carries over to multiple machines. This performance guarantee would hold also in our online setting. However, simple examples show that the crucial property they use in the line of argumentation does not hold for the parallel-machine version of the Spt algorithm. On the other hand, Erlebach et al. [EKM04] do not specify how a generalization of the Spt rule and their proofs to multiple machines looks like. In case, the parallel-machine extension of Spt uses only one out of m available machines, then their crucial property still holds.

For the sake of completeness, we give an n-competitive online algorithm, including the full proof, for $P\,|\,prec\,|\sum w_jC_j$. At the same time this renders more precisely the claimed performance of the corresponding (offline) problem with generalized And/Or-precedence constraints.

Let C_j^S denote the completion time of a job j in some schedule S. Adopting the notation in [EKM04], we define the *threshold* ξ_j^S of a job j in schedule S as the maximum processing time of a job that finishes no later than j. More formally,

$$\xi_j^S \;=\; \max_{k\in\mathcal{J}}\{\, p_k \mid C_k^S \;\leq\; C_j^S \,\}\,.$$

These thresholds have a nice property that is crucial in the analysis of the SPT algorithm on a single machine by Erlebach et al. Their *Threshold-Lemma* [EKM04, Lemma 2] states that the processing time p_k of each job k with $C_k^{\text{SPT}} \leq C_j^{\text{SPT}}$ is bounded from above by j's threshold ξ_j^S in any schedule S, that means $p_k \leq \xi_j^S$. This is true for the single processor case, but a simple counter example proves it wrong already on two machines.

Example 2.13. *Consider four jobs: jobs* $1, 2,$ *and* 3 *have unit length and job* 4 *has processing time* $p_4 = 2$*; the given partial order is* $1 \prec 2$ *and* $1 \prec 3$*. In a* SPT *schedule on* $m = 2$ *machines, job* 4 *finishes earlier than job* 3 *does, i. e.,* $C_4^{\text{SPT}} \leq C_3^{\text{SPT}}$*.*

Now consider a schedule S *that keeps one machine idle for one time unit and schedules then jobs* 2 *and* 3 *simultaneously on both machines. Here, job* 3 *has a threshold* $\xi_3^S = 1$ *which leads to a contradiction to the claim* $p_4 \leq \xi_3^S$*.*

We can still take advantage of properties of the thresholds by considering the online algorithm that runs the SPT algorithm on only *one* machine. It simply ignores $m - 1$ remaining available machines, and therefore, it will leave a performance gap of a factor in the order of m.

In the remaining part of this section, we denote by SPT the algorithm that schedules all jobs on only one machine using the ONLINE SPT rule. We give a slightly different, strengthened version of the *Threshold-Lemma* [EKM04, Lemma 2] by Erlebach et al.

Lemma 2.14 (Threshold-Lemma)**.** *Let* S *be a feasible schedule for an instance of the scheduling problem* $\mathrm{P}\,|\,prec\,|\sum w_jC_j$*. Then for any job* $j \in \mathcal{J}$ *with threshold* ξ_j^S *in schedule* S *holds*

$$\xi_j^{\text{SPT}} \;\leq\; \xi_j^S\,.$$

Proof. For the sake of contradiction, assume that $\xi_j^{\text{SPT}} > \xi_j^S$. Clearly, it follows that $p_j < \xi_j^{\text{SPT}}$. Let x denote the job that "causes" the threshold, that means, $p_x = \xi_j^{\text{SPT}}$, and thus, $p_j < p_x$. Since SPT chooses by definition always the job with shortest processing time among the available ones, job j cannot have been available at the start date of job x, S_x^{SPT}. Hence, there must be a predecessor, say k, of j among the jobs that are available at time S_x^{SPT}. This job has processing time $p_k \geq p_x$ otherwise SPT would not have started x at S_x^{SPT}. This gives the contradiction $\xi_j^S \;\geq\; p_k \;\geq\; p_x \;=\; \xi_j^{\text{SPT}}$.

□

Now, we can establish a performance guarantee for the SPT algorithm for our online problem $\mathrm{P}\,|\,prec\,|\sum w_jC_j$, as well as, for the more general (offline) problem of scheduling jobs with AND/OR-precedence constraints.

Theorem 2.15. *The* Online Spt *algorithm that utilizes only one machine is n-competitive for the online scheduling problem* $\mathrm{P}\,|\,prec\,|\sum w_jC_j$.

Proof. Let jobs be indexed in their order in the Spt schedule. Since there is no idle time in the Spt schedule, the completion time C_j^{SPT} of a job j in the Spt schedule is

$$C_j^{\text{SPT}} = \sum_{k=1}^{j} p_k \;\leq\; n\,\xi_j^{\text{SPT}}\,. \tag{2.4}$$

With the Threshold-Lemma 2.14 and the fact that $\xi_j^S \leq C_j^S$ holds by definition for any schedule S – thus, also for an optimal schedule Opt – we conclude from inequality (2.4)

$$C_j^{\text{SPT}} \;\leq\; n\,\xi_j^{\text{OPT}} \;\leq\; n\,C_j^{\text{OPT}}\,.$$

Weighted summation over all jobs $j \in \mathcal{J}$ proves the theorem. □

We complement this upper bound on the competitive ratio by a lower bound which leaves a gap of factor at most $2m$.

Theorem 2.16. *No deterministic online algorithm can achieve a competitive ratio less than $(n-1)/m$ for the scheduling problem* $\mathrm{P}\,|\,prec\,|\sum w_jC_j$ *on any number of machines.*

Proof. Consider the following instance that consists of n jobs and assume, w.l.o.g., that $n-1$ is a multiple of the number of machines m. We have $n-1$ independent jobs $1, 2, \ldots, n-1$ with weights $w_j = 0$ and unit processing time. Suppose, that the online algorithm chooses the job ℓ to be scheduled as one of the last jobs (with maximum completion time). Then, we have one final job n in the instance with ℓ as its predecessor and with processing time zero and weight $k > 0$.

Clearly, an online algorithm can schedule the highly weighted last job only as the final job, achieving a schedule with value $k\,(n-1)/m$. In contrast, an offline algorithm would choose job ℓ as one of the m first jobs to be processed, followed by the highly weighted job n. This yields value of k. Thus, the ratio between both value is $(n-1)/m$.

□

Adding a randomizing ingredient to the instance above, we extend the result to a lower bound for any randomized online algorithm. Here, we make use of Yao's principle [Yao77]; see Theorem 1.3 in Chapter 1.

Theorem 2.17. *The competitive ratio of any randomized online algorithm is bounded by $n/(2m)$ for the scheduling problem $\mathrm{P}\,|\,prec\,|\sum w_jC_j$ for any number of machines.*

Proof. Consider the instance in the previous proof. Against a randomized algorithm, the adversary does not know which will be the last scheduled job ℓ among the independent jobs. Therefore, we modify (and randomize) the instance by adding a random precedence relation between a job j, for $j = 1, \dots, n-1$ and the job n with probability $1/(n-1)$, for any job j.

Clearly, an offline solution yields a value of at most k as in the previous proof. Now, consider any deterministic online algorithm ALG. Let $\Pr[i \prec n]$ be the probability with that job i is the predecessor of job n. Any reasonable algorithm ALG schedules the highly weighted job n immediately after it became known. Thus, the expected completion time of job n in the schedule of ALG is

$$\mathbb{E}\left[C_n^{\textsc{Alg}}\right] \;\geq\; \sum_{i=1}^{n-1} \Pr[i \prec n] \cdot C_i \;=\; \frac{1}{n-1} \sum_{i=1}^{\frac{n-1}{m}} m \cdot i \;\geq\; \frac{n}{2m}\,.$$

Since all jobs $1, 2, \dots, n-1$ have zero weight, their completion times do not contribute to the value of the schedule. Thus, the expected value of the schedule is

$$\mathbb{E}\left[\sum_j w_j C_j^{\textsc{Alg}}\right] \;=\; k\,\mathbb{E}\left[C_n^{\textsc{Alg}}\right] \;\geq\; k\,\frac{n}{2\,m}\,.$$

Hence, the ratio of the expected values of any deterministic schedule to the value of an optimal schedule is $n/(2m)$. By Yao's principle (Theorem 1.3) this gives the lower bound for any online algorithm.

□

2.3.4 SCHEDULING ON PARALLEL MACHINES – EQUAL JOB WEIGHTS

Notice that the lower bounds in the previous section heavily depend on choosing adequate job weights. Hence, they do not transfer to the problem setting where jobs have equal weights. For this relaxed problem $\mathrm{P}\,|\,prec\,|\sum C_j$, we show the following weaker lower bound which we complement by a strengthened upper bound matching up to a constant factor.

Theorem 2.18. *The competitive ratio of any deterministic online algorithm for the scheduling problem $\mathrm{P}\,|\,prec\,|\sum C_j$ has a lower bound of $\frac{2}{3}\sqrt{n/m} - 1$. A lower bound for randomized online algorithms is $\frac{1}{3}\sqrt{n/m}$.*

Proof. First, we show the lower bound for deterministic online algorithms. We have mk independent jobs with processing times $p_j = 1$ for all $j = 1, \ldots, mk$ where $k >> m$. Moreover, there are $mk^2 - mk$ jobs that have length 0 and which must obey precedence constraints that form one long chain $mk+1 \prec mk+2 \prec \ldots$. Let $\ell \in \{1, \ldots, mk\}$ be the job to be scheduled last by the online algorithm among the independent jobs. This job ℓ is a predecessor of $mk+1$, the first job of the chain. Note that $C_j^{\text{ALG}} \geq k$. Thus, an online algorithm ALG cannot start the job chain with $mk^2 - mk$ jobs earlier than time k. ALG yields a schedule of value

$$\begin{aligned} \text{ALG} \quad &\geq \quad m \sum_{i=1}^{k} i + (mk^2 - mk)k \;=\; \frac{1}{2} mk \left(1 + (2k-1)\, k\right) \\ &\geq \quad \frac{1}{2} mk^2 (2k-1)\,. \end{aligned} \tag{2.5}$$

In contrast, an optimal offline algorithm knows the sequence in advance. By processing job ℓ at time 0 and starting the chain of zero length jobs at time 1, it can achieve an objective value

$$\begin{aligned} \text{OPT} \quad &\leq \quad m \sum_{i=1}^{k} i + mk^2 - mk \;=\; \frac{1}{2} mk \,(3k-1) \\ &\leq \quad \frac{3}{2} mk^2\,. \end{aligned} \tag{2.6}$$

The ratio of the bounds in (2.5) and (2.6) combined with the number of jobs, $n = mk^2$, gives the lower bound on the competitive ratio of any deterministic online algorithm.

$$\frac{\text{ALG}}{\text{OPT}} \;\geq\; \frac{2k-1}{3} \;>\; \frac{2}{3}\sqrt{\frac{n}{m}} - 1\,.$$

The proof that one half of this value is a lower bound for randomized online algorithms is similar to the proof of Theorem 2.17. We consider the instance above and replace the precedence constraint $\ell \prec mk+1$ by a randomized variant. That means, with probability $\Pr[j \prec k+1] = 1/(mk)$, a job $j = 1, \ldots, mk$ precedes job $mk+1$, the first job of the job chain.

The earliest possible completion time of any zero-length job $i = mk + 1, \ldots, mk^2$ is the expected time when the jobs $mk+1$ is revealed to an online algorithm ALG, that is, when its random predecessor completes. Thus,

$$\mathbb{E}\left[C_i^{\text{ALG}}\right] \;\geq\; \sum_{j=1}^{mk} \Pr\left[j \prec mk+1\right] \cdot \mathbb{E}\left[C_j^{\text{ALG}}\right] \;\geq\; m \sum_{j=1}^{k} \frac{1}{mk}\, j \;=\; \frac{k+1}{2}\,.$$

The expected total value of the online algorithm's schedule can be bounded by $\text{ALG} \geq mk^2(k+1)/2$. When comparing against an offline optimum value (as estimated above), we yield the bound of $k/3 = \frac{1}{3}\sqrt{n/m}$ on the competitive ratio. □

Clearly, the upper bound of n on the competitive ratio for the SPT algorithm in Theorem 2.15 holds in the unweighted setting. Here, we give an improved bound by extending ideas of Erlebach et al. [EKM04] for the single-machine setting.

Theorem 2.19. *The* SPT *algorithm that utilizes only one machine is* $\sqrt{2mn}$*-competitive for the online scheduling problem* $\mathrm{P}\,|\,prec\,|\sum C_j$.

Proof. Let $\alpha > 0$ be a parameter that will be specified later. Consider an SPT schedule and define C_x^{SPT} as the last completion time in this schedule such that all jobs in the SPT order completing before C_x^{SPT} have a processing time of at most $C_x^{\text{SPT}}/(\alpha\sqrt{n})$; that is, more formally

$$C_x^{\text{SPT}} \;=\; \max_{j\in\mathcal{J}}\{\, C_j^{\text{SPT}} \mid p_k \;\leq\; \frac{C_j^{\text{SPT}}}{\alpha\sqrt{n}} \text{ for all } k \in \mathcal{J} \text{ with } C_k^{\text{SPT}} \;\leq\; C_j^{\text{SPT}}\,\}\,.$$

This value, C_x^{SPT}, is used to partition the set of jobs into two disjunctive subsets: $\mathcal{J}^{\leq}$ denotes the set of jobs that have completed by time C_x in the SPT schedule, i. e., $\mathcal{J}^{\leq} = \{j \in \mathcal{J} \,|\, C_j^{\text{SPT}} \leq C_x^{\text{SPT}}\}$, and $\mathcal{J}^{>}$ consists of the remaining jobs $\mathcal{J} \setminus \mathcal{J}^{\leq}$. Obviously, the completion time of job x is $C_x^{\text{SPT}} = \sum_{j\in\mathcal{J}^{\leq}} p_j$. Now, the value of the SPT schedule can be expressed as

$$\sum_{j\in\mathcal{J}} C_j^{\text{SPT}} \;=\; \sum_{j\in\mathcal{J}^{\leq}} C_j^{\text{SPT}} \;+\; \sum_{j\in\mathcal{J}^{>}} C_j^{\text{SPT}}\,.$$

We bound the completion times of jobs of both job sets separately. We begin with job set $\mathcal{J}^{\leq} \neq \emptyset$ and bound the sum of optimal completion times of jobs under consideration.

Let OPT be an optimal schedule for all jobs $j \in \mathcal{J}$ and $\sum_{j\in\mathcal{J}^{\leq}} C_j^{\text{OPT}}$ be the portion of the objective value of OPT due to the considered job set. Consider now a schedule OPT' which optimally schedules the job set $\mathcal{J}^{\leq}$. Clearly,

$$\sum_{j\in\mathcal{J}^{\leq}} C_j^{\text{OPT}} \;\geq\; \sum_{j\in\mathcal{J}^{\leq}} C_j^{\text{OPT}'}\,. \tag{2.7}$$

We bound the value of OPT' from below by the minimum total completion time for the relaxed problem, where independent jobs must be scheduled on m identical parallel machines with the freedom to choose the non-negative processing times and with the constraints that all jobs have processing times of at most $C_x^{\text{SPT}}/(\alpha\sqrt{n})$ and the sum of all processing times is at least C_x^{SPT}.

Let p_{ki} denote the processing time of the k-th last job on machine i; if there are less than k jobs assigned to machine i, then $p_{ki} := 0$. The total completion time of all jobs on machine i is $\sum_k k \cdot p_{ki}$. Thus, we can formulate the relaxed problem as

$$\begin{aligned} \min \quad & \sum_{i=1}^{m} \sum_{k} k \cdot p_{ki} \\ \text{s.t.} \quad & \sum_{i=1}^{m} \sum_{k} p_{ki} \;\geq\; C_x^{\text{SPT}} && (2.8) \\ & 0 \;\leq\; p_{ki} \;\leq\; \frac{C_x^{\text{SPT}}}{\alpha\sqrt{n}}, \qquad \forall k, i\,. && (2.9) \end{aligned}$$

The objective function, rewritten as $\sum_{i=1}^{m}\sum_k k \cdot p_{ki} = \sum_k k \sum_{i=1}^{m} p_{ki}$, is minimized if the "last" jobs (small k) on each machine have maximum length. Let $b := \lfloor \alpha\sqrt{n}/m \rfloor$. Then the optimal solution of the relaxed problem is given by $p_{ki} = C_x^{\text{SPT}}/(\alpha\sqrt{n})$ for $k = 1, \ldots, b$ on each machine i. In total, mb jobs have maximum processing time. To fulfill condition 2.8, we distribute the remaining processing time, $C_x^{\text{SPT}} - mbC_x^{\text{SPT}}/(\alpha\sqrt{n})$, over the $(b+1)$-st jobs on all machines, i. e., $p_{b+1,i} = C_x^{\text{SPT}}(1 - mb/(\alpha\sqrt{n})/m$, which also meets condition 2.9. All remaining jobs have processing time 0.

The objective value of Opt' is no less than the optimal value of the relaxed problem.

$$\begin{aligned} \sum_{j \in \mathcal{J}^{\leq}} C_j^{\text{OPT}'} &\geq \sum_{i=1}^{m}\sum_{k=1}^{b} k \cdot p_{ki} + \sum_{i=1}^{m} (b+1)\, p_{b+1,i} \\ &= \sum_{i=1}^{m}\sum_{k=1}^{b} k \cdot \frac{C_x^{\text{SPT}}}{\alpha\sqrt{n}} + (b+1)\, C_x^{\text{SPT}}(1 - \frac{bm}{\alpha\sqrt{n}}) \\ &= \frac{b+1}{2}\, C_x^{\text{SPT}}(2 - \frac{bm}{\alpha\sqrt{n}}) = \frac{\lfloor \frac{\alpha\sqrt{n}}{m} \rfloor + 1}{2}\, C_x^{\text{SPT}}(2 - \frac{\lfloor \frac{\alpha\sqrt{n}}{m} \rfloor\, m}{\alpha\sqrt{n}}) \\ &\geq \frac{\alpha\sqrt{n}}{2m}\, C_x^{\text{SPT}}\,. \end{aligned}$$

With this estimate of the relevant portion of the optimal value we can bound the value of the Spt schedule.

$$\sum_{j \in \mathcal{J}^{\leq}} C_j^{\text{SPT}} \;\leq\; n\, C_x^{\text{SPT}} \;\leq\; \frac{2m}{\alpha}\sqrt{n} \sum_{j \in \mathcal{J}^{\leq}} C_j^{\text{OPT}}\,. \tag{2.10}$$

Consider now jobs in the remaining set $\mathcal{J}^>$; by definition, there exists for each job $j \in \mathcal{J}^>$ a job k that completes in SPT earlier than j and has processing time $p_k > C_x^{\text{SPT}}/(\alpha\sqrt{n})$. We conclude from this fact and the Threshold-Lemma 2.14 (including the notion of the threshold ξ_j^S) that for all $j \in \mathcal{J}^>$ holds

$$C_j^{\text{OPT}} \;\geq\; \xi_j^{\text{OPT}} \;\geq\; \xi_j^{\text{SPT}} \;\geq\; p_k \;>\; \frac{C_j^{\text{SPT}}}{\alpha\sqrt{n}}\,.$$

Summation over all jobs $j \in \mathcal{J}^>$ yields a bound on the completion times in the SPT schedule,

$$\sum_{j\in\mathcal{J}^>} C_j^{\text{SPT}} \;\leq\; \alpha\sqrt{n} \sum_{j\in\mathcal{J}^>} C_j^{\text{OPT}}\,.$$

Finally, combination with Equality (2.10) yields the desired bound

$$\sum_{j\in\mathcal{J}} C_j^{\text{SPT}} \;\leq\; \max\{\frac{2m}{\alpha},\, \alpha\}\,\sqrt{n} \sum_{j\in\mathcal{J}} C_j^{\text{OPT}}\,,$$

which is minimized if the parameter α is set to $\alpha = \sqrt{2m}$. $\square$

This analysis holds also if AND/OR-precedence constraints are present. Thus, we improve the approximation factor of n suggested in [EKM04] for the (offline) scheduling problem where jobs have equal weights. Finally, this result also improves the performance guarantee for the single-machine problem ([EKM04] and Theorem 2.12) by a negligible factor.

Chapter 3

Stochastic Scheduling

In this chapter we consider the stochastic scheduling model and present first constant approximation guarantees for *preemptive stochastic scheduling* to minimize the sum of weighted completion times on parallel identical machines. For scheduling jobs with release dates, we derive policies with the performance ratio of 2. This matches the currently best known result for the corresponding deterministic problem in an online environment presented in Section 2.2.3. In comparison to previously known results for the stochastic scheduling model in a non-preemptive setting, our performance guarantee stands out by being constant and independent of the probability distribution of processing times.

3.1 Model and Preliminaries

The stochastic scheduling model features incomplete information about the problem instance in a different way than the online model in the previous chapter. It is assumed that there is no uncertainty about the set of jobs and their release dates. Instead, the uncertain component appears in the form of stochastic processing times given as random variables. We refer to Section 1.2.2.1 for a detailed introduction of the stochastic scheduling model and the notion of scheduling policies.

The particular problems we consider are preemptive scheduling on a single as well as on parallel machines to minimize the sum of weighted completion times in expectation, denoted by $1 \,|\, r_j, pmtn \,|\, \mathbb{E}\left[\sum w_j C_j\right]$ and $\mathrm{P} \,|\, r_j, pmtn \,|\, \mathbb{E}\left[\sum w_j C_j\right]$, respectively. For an instance $\mathcal{I}$, consisting of the number of machines, m, the set of jobs, $\mathcal{J}$, together with their release dates r_j, weights w_j and processing time distributions P_j, let $C_j^{\Pi}(\mathcal{I})$ denote the random variables for completion times of jobs under policy Π. We also write C_j^{Π}, for short. Moreover, we let

$$\mathbb{E}\left[\,\Pi(\mathcal{I})\,\right] = \mathbb{E}\left[\sum\nolimits_{j\in\mathcal{J}} w_j\, C_j^{\Pi}(\mathcal{I})\right] = \sum\nolimits_{j\in\mathcal{J}} w_j\, \mathbb{E}\left[\,C_j^{\Pi}(\mathcal{I})\,\right]$$

denote the expected outcome of a scheduling policy Π on instance $\mathcal{I}$.

3.2 PREVIOUS WORK

Stochastic scheduling has been considered for about forty years. Various research has been published concerning criteria that guarantee the optimality of simple policies for rather special, restricted scheduling problems. Since we focus in this thesis on problems with the objective to minimize the sum of (weighted) completion times, we review previous work on such problems.

The WSEPT *Policy*

One of the first papers reporting results on stochastic scheduling to minimize the total weighted sum of completion times is by Rothkopf [Rot66]. For the single-machine problem $1 \mid \mid \mathbb{E}\left[\sum w_j C_j\right]$, he showed optimality of todays probably best known and most often applied policy in stochastic scheduling, the *Weighted Shortest Expected Processing Time* (WSEPT) rule.

Algorithm 7: WSEPT

Schedule jobs in non-increasing order of their ratios of weight over expected processing time, $w_j / \mathbb{E}[P_j]$.

This policy is a direct extension of Smith's WSPT rule (see Section 2.2.1) to the stochastic setting. In fact, even the interchange argument proving WSPT's optimality for the corresponding deterministic problem carries over to the stochastic setting in a straightforward way.

On parallel machines, it has been shown by Bruno, Downey, and Frederickson [BDF81] that WSEPT is optimal if all weights are equal. In that case, the policy reduces to the so called *Shortest Expected Processing Time* (SEPT) rule. More general, Kämpke showed in [Käm89] that this classical policy is optimal if the weights are *agreeable*, where jobs i, j are defined as *agreeable* if $\mathbb{E}[P_i] < \mathbb{E}[P_j]$ implies $w_i \leq w_j$. Later, Weiss [Wei90, Wei92] derived additive performance bounds for the WSEPT rule on the same problem class, $P \mid \mid \mathbb{E}\left[\sum w_j C_j\right]$; his bounds prove asymptotic optimality of this rule for a certain class of processing time distributions.

Consider the more general case, where jobs have non-trivial release dates. WSEPT inserts a large amount of unnecessary idle time and is, in general, not optimal anymore. A straightforward extension of the policy consists of a refinement of the set of jobs that is considered for scheduling: to this end, call a job j *available* at a given time t if it has not yet been scheduled, and if its release date has passed, that is, if $r_j \leq t$.

Algorithm 8: ONLINE WSEPT

At any time when the machine is idle schedule a job with largest ratio of weight over expected processing time, $w_j/\mathbb{E}[P_j]$, among the available jobs.

This extension corresponds to the generalization of Smith's classic WSPT rule to the setting with non-trivial release dates in the deterministic (online) scheduling environment; see Section 2.2.1. Therefore we refer to this generalization also as ONLINE WSEPT. Notice that this policy utilizes at any time t only knowledge about jobs released at time $r_j \leq t$; thus, it runs indeed online. Chou, Liu, Queyranne, and Simchi-Levi [CLQSL06] proved asymptotic optimality of this generalized WSEPT rule for the single machine problem with more general probability distributions of processing times, $1 \,|\, r_j \,|\, \mathbb{E}[\sum w_j C_j]$, assuming that the weights w_j and processing times p_j can be bounded from above and below by constants. Whenever release dates are absent, this policy reduces to the traditional WSEPT rule.

Preemptive Stochastic Scheduling

Some of the first results on *preemptive scheduling* that can be found in the literature are by Chazan, Konheim, and Weiss [CKW68] and Konheim [Kon68]. They formulated sufficient and necessary conditions for a policy to solve the single-machine problem optimally where all jobs become available at the same time, $1 \,|\, pmtn \,|\, \mathbb{E}[\sum w_j C_j]$. Later Sevcik [Sev74] developed an intuitive method for creating optimal schedules. He introduced a priority policy that relies on an index which can be computed for each job based on the properties of the job itself but independent of other jobs.

Gittins [Git79] showed that this priority index is a special case of his Gittins index [Git79, Git89] originally derived for so-called multi-armed bandit problems. Later in 1995, Weiss [Wei95] formulated Sevcik's priority index again in terms of the Gittins index and named it a *Gittins index priority policy*. He also provided a different proof of the optimality of this priority policy, based on the work conservation invariance principle. Weiss covers a more general problem than the one considered here and in [CKW68, Kon68, Sev74]: The holding costs (weights) of a job are not deterministic constants, but may vary during the processing of a job. At each state these holding costs are random variables.

For more general scheduling problems with release dates and/or multiple machines, no optimal preemptive policies are known. Instead, literature reflects a variety of research on restricted problems as those with special probability distributions for processing times or job weights. Pinedo [Pin83] considered the single-machine problem in the presence of release dates but restricted the processing times to be drawn from exponential distributions. He showed the optimality of a preemptive version of the ONLINE WSEPT policy. If all jobs become available at the same time then preemption is not even necessary. Rothkopf [Rot66] showed for the more general distribution of increasing failure rates (monotone hazard rate) but without job weights, that no finite number of preemptions can improve the performance.

This result holds true also for multiple machines if all job weights are equal. Weber [Web82] showed for this setting that the *Shortest expected remaining processing time* (SERPT) policy is optimal. In case that processing times are not drawn from a distribution with decreasing failure rates (monotone hazard-rate), Coffman, Hofri, and Weiss [CHW89] showed that this policy is not optimal, even if processing times follow the same two-point distribution and even if we deal with only two processors. On the other hand, for such a special distribution, they show that the SERPT policy is asymptotically optimal and has a turnpike property: asymptotically, for large n, most of the optimal decisions will be according to this policy. In case of general probability distributions, Weiss [Wei95] showed that the Gittins index priority policy has an expected value that is only a constant away from the optimal value. Moreover, he shows that this policy has a turnpike property and is asymptotically optimal.

None of the results on multiple machines considers individual job weights and/or non-trivial release dates. Only for exponential processing times and with strong restrictions on weights, Kämpke [Käm89] can prove the optimality of the SEPT policy for the problem $\mathrm{P}\,|\,pmtn\,|\,\mathbb{E}\,[\,\sum w_j C_j\,]$; in this setting weights need to be *agreeable* as defined earlier, and therefore, this policy coincides with the WSEPT rule without preempting any job.

Approximation in Stochastic Scheduling

The literature review above reflects efforts to find optimal policies which have been successful only for very special cases of stochastic scheduling problems. Indeed, it might be quite hopeless to search for optimal policies in the general setting, since already the deterministic counterpart of the scheduling problem we consider, is well-known to be $\mathcal{NP}$-hard, even in the case that there is only a single processor or if all release dates are equal [LLLR84, LRB77]. Moreover, optimal policies may be extremely complicated to describe and to implement, as Weiss [Wei90] states – it may involve dynamic scheduling with inserted idle time, and it may depend on the entire processing time distribution of each job rather than on a few parameters; see also Uetz [Uet03] for a discussion of deliberate idle times in optimal policies.

Therefore, attempts have been made recently on obtaining approximation algorithms for stochastic scheduling problems. Going further than showing asymptotic optimality as Coffman et al. [CHW89] and Weiss [Wei90, Wei92], Möhring, Schulz, and Uetz [MSU99] derived first constant-factor approximations for the *non-preemptive* problem with and without release dates. Their results are based on a linear programming relaxation – that includes the design as well as the analysis of policies. In principle, Möhring et al. [MSU99] generalized a class of valid inequalities known in the deterministic scheduling environment, see, e. g., Queyranne [Que93] and Schulz [Sch96a], to the case with stochastic processing times. This LP relaxation and the lower bound it yields as well as the given performance guarantees for non-preemptive policies depend on a parameter Δ defined by expected values of processing times and the coefficients of variation, such that $\Delta \geq \text{Var}[P_j]/\mathbb{E}\,[\,P_j\,]^2$ for all jobs j.

For the scheduling problem $\text{P}\,|\,r_j\,|\,\mathbb{E}\,[\sum w_j C_j\,]$, Möhring et al. [MSU99] provide a policy with an approximation guarantee of $3 - \frac{1}{m} + \max\{1, \frac{m-1}{m}\,\Delta\,\}$ which reduces to $4 - \frac{1}{m}$ for $\Delta \leq 1$; this is the case for many common probability distributions as the exponential, uniform and Erlang distributions. Moreover, for the setting in which all jobs become available at the same release date then they show that WSEPT yields a performance guarantee of $1 + \frac{(m-1)(\Delta+1)}{2m}$ which is $2 - \frac{1}{m}$ for $\Delta \leq 1$. Megow, Uetz, and Vredeveld [MUV06] and Schulz [Sch05] partially improved the results in a more general setting; see Chapter 4.

Skutella and Uetz [SU05] complemented the approximation results by first constant-approximative policies for scheduling jobs that must obey precedence constraints. The approximation ratio also depends on the parameter Δ; for an upper bound $\Delta \leq 1$ they derive a linear programming based $(3 + 2\sqrt{2})$–approximation for the problem $\text{P}\,|\,r_j, prec\,|\,\mathbb{E}\,[\sum w_j C_j\,]$.

This concludes the overview on approximative policies for stochastic scheduling to minimize the expected total (weighted) completion time. We are not aware of any other work on approximations in this stochastic model for any particular problem class except for a paper by Goel and Indyk [GI99] and the recent PhD thesis by Dean [Dea05]. Goel and Indyk consider the problem of minimizing the makespan on identical parallel machines and derive constant approximations and a PTAS for special probability distributions of processing times. In contrast, Deans examinations focus on problems with hard deadlines for which he derives first constant approximation guarantees.

3.3 APPROXIMATIVE POLICIES FOR PREEMPTIVE STOCHASTIC SCHEDULING

Only recently research interest addressed also approximative policies. While all of the results reviewed in the previous section hold for non-preemptive scheduling, we are not aware of any approximation results for problems that allow job preemption. Previous results on minimizing the expected sum of completion times [MSU99, SU05, MUV06, Sch05] are based on linear programming relaxations (at least in their analysis) which do not seem to carry over to the preemptive setting.

In this section, we give first approximation results for preemptive policies for stochastic scheduling on identical parallel machines. We provide two policies for the problem $\mathrm{P}\,|\,r_j, pmtn\,|\,\mathbb{E}\,[\sum w_j C_j\,]$ with a proven approximation guarantee of 2 which matches exactly the currently best known approximation result for the deterministic online version of this problem in Chapter 2. The policy in Section 3.3.3 can be interpreted as a generalization of the P-WSPT algorithm introduced in the deterministic online scheduling framework in Section 2.2.3. Our policy in Section 3.3.4 is a randomized extension of a single-machine policy that can be seen as a stochastic generalization of the deterministic online algorithm WSRPT introduced in Section 2.2.2.

In order to derive our results we introduce a new non-trivial lower bound on the expected value of an unknown optimal policy. This bound is derived by borrowing ideas for a *fast single-machine relaxation* from Chekuri, Motwani, Natarjan, and Stein [CMNS01]. The crucial ingredient to our investigations is then the application of a Gittins index priority policy which is known to solve optimally a relaxed version of our fast single-machine relaxation [Sev74, Wei95]. The priority index used in this optimal policy also inspires the design of our policies. Thereby, our preemptive policies extensively utilize information on processing time distributions other than the first (and second) moments, which distinguishes them significantly from approximative policies known in the non-preemptive setting.

In general our policies are not optimal. However, on restricted problem instances they coincide with policies whose optimality is well known. If processing times are exponentially distributed and release dates are absent, our parallel-machine policy in Section 3.3.3 coincides with the ONLINE WSEPT rule discussed in the previous section. Furthermore, the single-machine policy derived in Section 3.3.4 solves deterministic instances of $1\,|\,r_j, pmtn\,|\,\sum C_j$ optimally since it achieves the same schedule as Schrage's *Shortest remaining processing time* (SRPT) rule [Sch68]. This performance is achieved in the weighted setting as well, if all release dates are equal. In that case our policy coincides with Smith's WSPT rule [Smi56]; see Section 2.2.1.

Our policies are not restricted to special probability distributions of processing times. However, in the following sections we consider only problems where job processing times follow discrete probability distributions. With the restriction on discrete probability functions and a standard scaling argument, we may assume w.l.o.g. that P_j attains integral values in the set $\Omega_j \subseteq \{1, 2, \ldots, M_j\}$ and that all release dates are integral. The sample space of all processing times is denoted by $\Omega = \Omega_1 \times \cdots \times \Omega_n$.

3.3.1 A Gittins Index Priority Policy

As mentioned in the introduction, a Gittins index priority policy solves the single-machine problem with trivial release dates, $1 \,|\, pmtn \,|\, \mathbb{E}\left[\sum w_j C_j\right]$, to optimality; see [Kon68, Sev74, Wei95]. This result is crucial for the approximation results we give in the following sections; it inspires the design of our policies and it serves as a tool for bounding the expected value of an unknown optimal policy for the more general problem that we consider. Therefore, we introduce in this section the Gittins index priority rule and some useful notation.

Given that a job j has been processed for y time units and it has not completed, we define the *expected investment* of processing this job for q time units or up to completion, which ever comes first, as

$$I_j(q, y) = \mathbb{E}\left[\min\{P_j - y, q\} \,|\, P_j > y\right].$$

The ratio of the weighted probability that this job is completed within the next q time units over the expected investment, is the basis of the Gittins index priority rule. We define it as the *rank* of a sub-job of length q of job j, after it has completed y units of processing:

$$R_j(q, y) = \frac{w_j \Pr\left[P_j - y \leq q \,|\, P_j > y\right]}{I_j(q, y)}.$$

This ratio is well defined if we assume that we compute the rank only for $q > 0$ and $P_j > y$, in which case the investment $I_j(q, y)$ has a value greater than zero.

For a given (unfinished) job j and attained processing time y, we are interested in the maximal rank it can achieve. We call this the Gittins index, or rank, of job j, after it has been processed for y time units.

$$R_j(y) = \max_{q \in \mathbb{R}^+} R_j(q, y).$$

The length of the sub-job achieving the maximal rank is denoted as

$$q_j(y) = \max\{\, q \in \mathbb{R}^+ \;:\; R_j(q,y) = R_j(y)\,\}.$$

With the definitions above, we define the Gittins index priority policy.

Algorithm 9: Gittins index priority policy (GIPP)

At any time t, process an unfinished job j with currently highest rank $R_j(y_j(t))$, where $y_j(t)$ denotes the amount of processing that has been done on job j by time t. Break ties by choosing the job with the smallest job index.

Theorem 3.1 ([Kon68, Sev74, Wei95]). *The Gittins index priority policy* (GIPP) *solves the stochastic scheduling problem* $1\,|\,pmtn\,|\,\mathbb{E}\,[\,\sum w_jC_j\,]$ *optimally.*

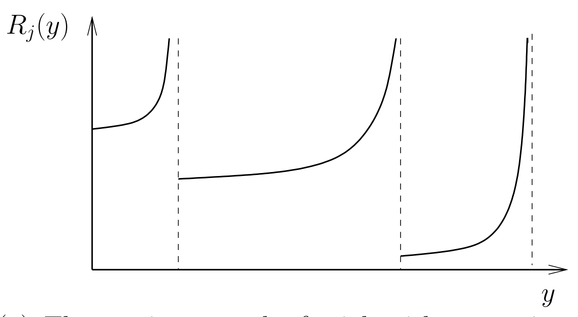
The following properties of the Gittins indices and the lengths of sub-jobs achieving the Gittins index are well known; see [Git89, Wei95]. In parts, they have been derived earlier in the scheduling context by Konheim [Kon68] and Sevcik [Sev74]. They prove useful to analyze GIPP as well as the more general policies in the following sections.

Proposition 3.2 ([Git89, Wei95]). *Consider a job j that has been processed for y time units. Then, for any* $0 < \gamma < q_j(y)$ *it holds*

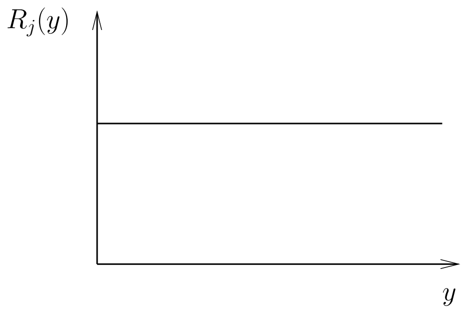
$$\begin{aligned} R_j(y) &\le R_j(y+\gamma)\,, && \text{(a)}\\ q_j(y+\gamma) &\le q_j(y)-\gamma\,, && \text{(b)}\\ R_j(y+q_j(y)) &\le R_j(y)\,. && \text{(c)} \end{aligned}$$

These properties help to understand GIPP better. Let us denote the sub-job of length $q_j(y)$ that causes the maximal rank $R_j(y)$, a *quantum* of job j. We now split a job j into a set of n_j quanta, denoted by tuples (j,i), for $i = 1,\ldots,n_j$. The processing time y_{ji} that a job j has attained up to a quantum (j,i) and the length of each quantum, q_{ji}, are recursively defined as $y_{j1} = 0$, $q_{ji} = q_j(y_{ji})$, and $y_{j,i+1} = y_{j,i} + q_{ji}$.

By Proposition 3.2 (a), we know that, while processing a quantum, the rank of the job does not decrease, whereas Proposition 3.2 (c) and the definition of $q_j(y)$ tell us that the rank is strictly lower at the beginning of the next quantum. Hence, once a quantum has been started, GIPP will process it for its complete length or up to the completion of the job, whatever



(a) The maximum rank of a job with processing time that follows a three-point distribution.



(b) The maximum rank of a job with exponential processing time.

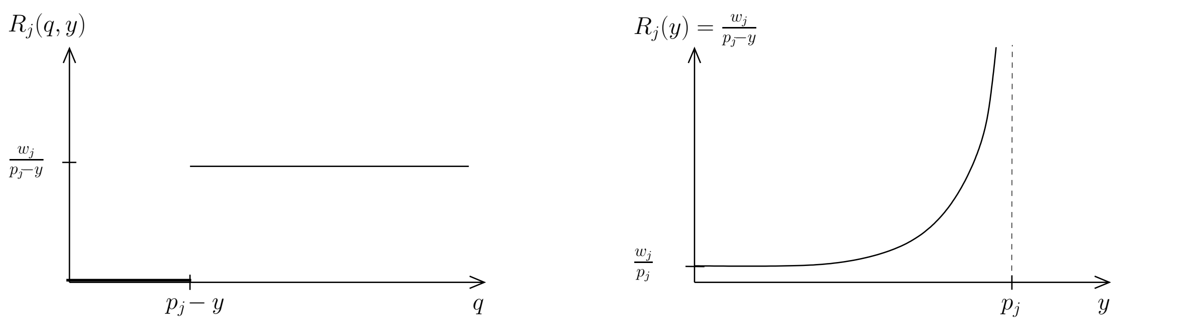
Figure 3.1: The maximum rank of jobs with certain processing time distributions depending on the amount of time, y, that the job has been processing.

comes first; that means, Gipp preempts a job only at the end of a quantum. Obviously, the policy Gipp processes job quanta non-preemptively in non-increasing order of their ranks. In particular, Gipp does not need to recompute the maximum rank of a running job until the completion of the current quantum.

Before proceeding with a deeper analysis of the policy Gipp, let us have a closer look at the rank of a (sub-)job and the implications of the properties observed above. Figure 3.1 illustrates the maximum rank of two jobs with particular processing time distributions (three-point and exponential distribution) as a function of the amount of time that the job has been processing.

For the analysis of our policies in the following sections also Proposition 3.2 (b) becomes important. It bounds the length of a new quantum that causes maximum rank if a previous quantum got preempted. Suppose, at some time t, a quantum of length q that maximizes the job rank R begins processing. Now, consider some time $t' < t + q$. Gipp does not recompute the rank and the quantum until the completion of q but in a more complex problem setting where jobs arrive at their individual release dates this might become essential. At time t', the new maximum job rank R' is by Proposition 3.2 (a) at least as large as R and, as Proposition 3.2 (b) states, the new quantum that causes the new rank R' has length q' which is not greater than the remaining part of quantum q, that is, $q' \leq q - (t' - t)$.

The general assumption of stochastic job processing times subsumes deterministic processing times as a special case. Consider an incomplete job j with deterministic processing time p_j of which y units have already finished. The rank and the quantum lengths are deterministically predetermined by



(a) The rank of a sub-job of length q after y units of processing.

(b) The maximum rank of a job after it has been processing for y units of time.

Figure 3.2: Rank functions in the special case of *deterministic processing times.*

their definition.

$$\begin{aligned} R_j(q,y) &= \frac{w_j \Pr[P_j - y \le q \mid P_j > y]}{I_j(q,y)} \\ &= \begin{cases} 0 & : \text{ iff } q < p_j - y \\ \frac{w_j}{p_j - y} & : \text{ otherwise.} \end{cases} \end{aligned}$$

The behavior of the rank function for deterministic job processing times is illustrated in Figure 3.2. The quantum length is infinite which does not harm the policy since it processes only unfinished jobs.
Let us turn back to the Policy GIPP. Recall that it runs job quanta in non-increasing order of their ranks. We assume that quanta $(j,1)$, $(j,2), \ldots (j, n_j)$ are naturally indexed in their order of occurrence. Now, we define the set $H(j,i)$ of all quanta that preceed quantum (j,i) in the GIPP order. Let $\mathcal{Q}$ be the set of all quanta, i. e., $\mathcal{Q} = \{(k,\ell) \mid k = 1,\ldots,n,\ l = 1,\ldots,n_k\}$, then

$$\begin{aligned} H(j,i) &= \{\,(k,\ell) \in \mathcal{Q} \mid R_k(y_{k\ell}) > R_j(y_{ji})\,\} \\ &\quad \cup \{\,(k,\ell) \in \mathcal{Q} \mid R_k(y_{k\ell}) = R_j(y_{ji}) \ \wedge\ k \le j\,\}\,. \end{aligned}$$

As the Gittins index of a job is decreasing with every finished quantum 3.2 (c), we know that $H(j,h) \subseteq H(j,i)$, for $h \le i$. In order to uniquely relate higher priority quanta to exactly one quantum of a job, we introduce the notation $H'(j,i) = H(j,i) \setminus H(j,i-1)$, where we define $H(j,0) = \emptyset$. Note that the quantum (j,i) is also contained in the set of its higher priority quanta $H'(j,i)$. In the same manner, we define the set of lower priority quanta as $L(j,i) = \mathcal{Q} \setminus H(j,i)$.

With these definitions and the observations above, we can give a closed formula for the expected objective value of GIPP.

Lemma 3.3. *The optimal policy for* $1\,|\,pmtn\,|\,\mathbb{E}\,[\sum w_j C_j\,]$, Gipp, *achieves an expected objective value of*

$$\mathbb{E}\,[\,\text{Gipp}\,] = \sum_j w_j \sum_{i=1}^{n_j} \sum_{(k,\ell)\in H'(j,i)} \Pr\,[P_j > y_{ji} \wedge P_k > y_{k\ell}] \cdot I_k(q_{k\ell}, y_{k\ell}).$$

Proof. Consider a realization of processing times $p \in \Omega$ and a job j. Let i_p be the index of the quantum in which job j finishes, that is, $y_{ji_p} < p_j \leq y_{ji_p} + q_{ji_p}$. The policy Gipp processes quanta of jobs that have not completed non-preemptively in non-increasing order of their ranks. Hence,

$$C_j(p) \;=\; \sum_{(k,\ell)\in H(j,i_p)\,:\,p_k > y_{k\ell}} \min\{q_{k\ell}, p_k - y_{k\ell}\}\,. \tag{3.1}$$

For an event $\mathcal{E}$, let $\chi(\mathcal{E})$ be an indicator random variable which equals 1 if and only if the event $\mathcal{E}$ occurs. The expected value of $\chi(\mathcal{E})$ equals then the probability with that the event $\mathcal{E}$ occurs, i. e., $\mathbb{E}\,[\,\chi(\mathcal{E})\,] = \Pr\,[\mathcal{E}]$. Additionally, we denote by $\xi_{k\ell}$ the special indicator random variable for the event $P_k > y_{k\ell}$.

We take expectations on both sides of equation (3.1) over all realizations. This yields

$$\begin{aligned}
\mathbb{E}\,[\,C_j\,] \;&=\; \mathbb{E}\left[\sum_{h: y_{jh} < P_j \leq y_{j,h+1}} \; \sum_{\substack{(k,\ell)\in H(j,h):\\ P_k > y_{k\ell}}} \min\{q_{k\ell}, P_k - y_{k\ell}\}\right] \\
&=\; \mathbb{E}\left[\sum_{h=1}^{n_j} \chi(y_{jh} < P_j \leq y_{j,h+1}) \sum_{(k,\ell)\in H(j,h)} \xi_{k\ell}\cdot \min\{q_{k\ell}, P_k - y_{k\ell}\}\right] \\
&=\; \mathbb{E}\left[\sum_{h=1}^{n_j} \chi(y_{jh} < P_j \leq y_{j,h+1}) \sum_{i=1}^{h} \sum_{(k,\ell)\in H'(j,i)} \xi_{k\ell}\cdot \min\{q_{k\ell}, P_k - y_{k\ell}\}\right] \\
&=\; \mathbb{E}\left[\sum_{i=1}^{n_j} \sum_{h=i}^{n_j} \chi(y_{jh} < P_j \leq y_{j,h+1}) \sum_{(k,\ell)\in H'(j,i)} \xi_{k\ell}\cdot \min\{q_{k\ell}, P_k - y_{k\ell}\}\right] \\
&=\; \mathbb{E}\left[\sum_{i=1}^{n_j} \chi(y_{ji} < P_j) \sum_{(k,\ell)\in H'(j,i)} \xi_{k\ell}\cdot \min\{q_{k\ell}, P_k - y_{k\ell}\}\right] \\
&=\; \mathbb{E}\left[\sum_{i=1}^{n_j} \xi_{ji} \sum_{(k,\ell)\in H'(j,i)} \xi_{k\ell}\cdot \min\{q_{k\ell}, P_k - y_{k\ell}\}\right]\,.
\end{aligned} \tag{3.2}$$

The equalities follow from an index rearrangement and the facts that by definition $H(j,h) = \cup_{i=1}^{h} H'(j,i)$ for any $h = 1, 2, \ldots, n_j$ and that n_j is an upper bound on the actual number of quanta of job j.

For jobs $k \neq j$, the processing times P_j and P_k are independent random variables and thus, the same holds for their indicator random variables ξ_{ji} and $\xi_{k\ell}$ for any i, ℓ. Using linearity of expectation, we rewrite (3.2) as

$$
\begin{aligned}
&= \sum_{i=1}^{n_j} \sum_{(k,\ell)\in H'(j,i)} \mathbb{E}\left[\, \xi_{ji} \cdot \xi_{k\ell} \cdot \min\{q_{k\ell}, P_k - y_{k\ell}\}\,\right] \\
&= \sum_{i=1}^{n_j} \sum_{(k,\ell)\in H'(j,i)} \sum_{x} x \cdot \Pr\left[\xi_{ji} = \xi_{k\ell} = 1 \wedge \min\{q_{k\ell}, P_k - y_{k\ell}\} = x\right] \\
&= \sum_{i=1}^{n_j} \sum_{(k,\ell)\in H'(j,i)} \sum_{x} x \cdot \Pr\left[\xi_{ji} = \xi_{k\ell} = 1\right] \cdot \Pr\left[\min\{q_{k\ell}, P_k - y_{k\ell}\} = x \,|\, \xi_{k\ell} = 1\right] \\
&= \sum_{i=1}^{n_j} \sum_{(k,\ell)\in H'(j,i)} \Pr\left[P_j > y_{ji} \,\wedge\, P_k > y_{k\ell}\right] \cdot \mathbb{E}\left[\,\min\{q_{k\ell}, P_k - y_{k\ell}\} \,|\, P_k > y_{k\ell}\,\right] \\
&= \sum_{i=1}^{n_j} \sum_{(k,\ell)\in H'(j,i)} \Pr\left[P_j > y_{ji} \,\wedge\, P_k > y_{k\ell}\right] \cdot I_k(q_{k\ell}, y_{k\ell}) \,,
\end{aligned}
$$

where the third equality follows from conditional probability and the fact that either $j \neq k$, thus ξ_{ji} and $\xi_{k\ell}$ are independent, or $(j,i) = (k,\ell)$ and thus the variables ξ_{ji} and $\xi_{k\ell}$ are the same. Weighted summation over all jobs concludes the proof. $\square$

3.3.2 A New Lower Bound on the Optimum

For the scheduling problem $\mathrm{P}\,|\, r_j, pmtn \,|\, \mathbb{E}\left[\sum w_j C_j\right]$ and most of its relaxations, optimal offline policies and the corresponding expected objective values are unknown. Therefore, we use lower bounds on the optimal value in order to compare the expected outcome of a policy with the expected outcome $\mathbb{E}\left[\,\textsc{Opt}\,\right]$ of an unknown optimal policy Opt. The trivial bound $\mathbb{E}\left[\,\textsc{Opt}\,\right] \geq \sum_j w_j(\, r_j + \mathbb{E}\left[\, P_j \,\right])$ is unlikely to suffice proving constant approximation guarantees. However, we are not aware of any other bounds known for the general preemptive problem. LP-based approaches are used in the non-preemptive setting [MSU99, SU05, CLQSL06, MUV06, Sch05] but it is unclear if or how they transfer.

In this section, we derive a new non-trivial lower bound for preemptive stochastic scheduling on parallel machines. We utilize the knowledge about Gipp's optimality for the single-machine problem without release dates; see Theorem 3.1. To that end, we show first that the *fast single-machine relaxation* introduced by Chekuri et al. [CMNS01] for the deterministic (online) scheduling environment applies in the stochastic setting as well.

Denote by $\mathcal{I}$ a scheduling instance of the parallel machine scheduling problem $\mathrm{P}\,|\,r_j, pmtn\,|\,\mathbb{E}\left[\sum w_j C_j\right]$, and let $\mathcal{I}'$ be the same instance to be scheduled on a single machine of speed m times the speed of the machines used for scheduling instance $\mathcal{I}$. An optimal single-machine policy $\textsc{Opt}_1$ yields an expected value $\mathbb{E}\left[\,\textsc{Opt}_1(\mathcal{I}')\,\right]$ on instance $\mathcal{I}'$ on the fast single machine.

Lemma 3.4. *The expected value of any parallel-machine policy* Π *applied to the parallel-machine scheduling instance* $\mathcal{I}$ *is bounded from below by the expected value of an optimal policy on instance* $\mathcal{I}'$ *on a fast single machine, that is,*

$$\mathbb{E}\left[\,\Pi(\mathcal{I})\,\right] \;\geq\; \mathbb{E}\left[\,\textsc{Opt}_1(\mathcal{I}')\,\right] .$$

Proof. Given a parallel-machine policy Π, we provide a policy Π' for the fast single machine that yields an expected objective value $\mathbb{E}\left[\,\Pi'(\mathcal{I}')\,\right] \leq \mathbb{E}\left[\,\Pi(\mathcal{I})\,\right]$ for any instance $\mathcal{I}$. Then the lemma follows since an optimal policy $\textsc{Opt}_1$ yields on the single machine an expected objective value $\mathbb{E}\left[\,\textsc{Opt}_1(\mathcal{I}')\,\right] \leq \mathbb{E}\left[\,\Pi'(\mathcal{I}')\,\right]$.

We construct policy Π' by letting its first decision point coincide with the first decision point of policy Π (the earliest release date). At any of its decision points, Π' can compute the jobs to be scheduled by policy Π and due to the fact that the processing times of all jobs are discrete random variables, it computes the earliest possible completion time of these jobs, in the parallel-machine schedule. The next decision point of Π', is the minimum of these possible completion times and the next decision point of Π. Between two consecutive decision points of Π', the policy schedules the same set of jobs that Π schedules, for the same amount of time. This is possible as the single machine, on which Π' operates, works m times as fast.

In this way, we ensure that all job completions in the parallel machine schedule obtained by Π, coincide with a decision point of policy Π'. Moreover, as Π' schedules the same set of jobs as Π between two decision points, any job that completes its processing at a certain time t in the schedule of Π, will also be completed by time t in the schedule of Π'. $\square$

With this relaxation, we derive a lower bound on the expected optimal value.

Theorem 3.5. *The expected value of an optimal policy* OPT *for the parallel machine problem $\mathcal{I}$ is bounded by*

$$\mathbb{E}\,[\,\text{OPT}(\mathcal{I})\,] \;\geq\; \frac{1}{m}\sum_j w_j \sum_{i=1}^{n_j} \sum_{(k,\ell)\in H'(j,i)} \Pr\left[P_j > y_{ji} \wedge P_k > y_{k\ell}\right]\cdot I_k(q_{k\ell}, y_{k\ell})\,.$$

Proof. We consider the fast single-machine instance $\mathcal{I}'$ as introduced above and relax it further to instance $\mathcal{I}'_0$ by setting all release dates equal. By Theorem 3.1, the resulting problem can be solved optimally by GIPP. Then, with Lemma 3.4 we have

$$\mathbb{E}\,[\,\text{OPT}(\mathcal{I})\,] \;\geq\; \mathbb{E}\,[\,\text{OPT}_1(\mathcal{I}')\,] \;\geq\; \mathbb{E}\,[\,\text{GIPP}(\mathcal{I}'_0)\,]\,. \tag{3.3}$$

From Lemma 3.3 we know that

$$\mathbb{E}\,[\,\text{GIPP}(\mathcal{I}'_0)\,] \;=\; \sum_j w_j \sum_{i=1}^{n_j} \sum_{(k,\ell)\in H'(j,i)} \Pr\left[P'_j > y'_{ji} \wedge P'_k > y'_{k\ell}\right]\cdot I'_k(q'_{k\ell}, y'_{k\ell})\,, \tag{3.4}$$

where the dashes indicate the modified variables in the fast single-machine instance $\mathcal{I}'_0$. By definition, $P'_j = P_j/m$ holds for any job j; this is also true for $\Pr\left[P_j > x\right] = \Pr\left[P'_j > x/m\right]$. Furthermore, the probability for the remaining processing time after y units of processing, $\Pr\left[P_j - y = x \mid P_j > y\right]$, remains the same on the fast machine. Therefore, the expected investment $I'_j(q', y')$ for any sub-job of length $q' = q/m$ of job j after it has received $y' = y/m$ units of processing coincides with

$$\begin{aligned} I'_j(q', y') &= \mathbb{E}\left[\,\min\{P'_j - y', q'\} \mid P'_j > y'\,\right] \\ &= \frac{1}{m}\mathbb{E}\left[\,\min\{P_j - y, q\} \mid P_j > y\,\right] \;=\; \frac{1}{m} I_j(q, y)\,. \end{aligned}$$

We conclude that the partition of jobs into quanta in instance $\mathcal{I}$ immediately gives the partition for the fast single-machine instance $\mathcal{I}'$. Each quantum (j,i) of job j maximizes the rank $R_j(q, y_{ji})$ and thus $q' = q/m$ maximizes the rank $R'_j(q/m, y/m) = R_j(q,y)/m$ on the single machine; hence, the quanta are simply shortened to an m-fraction of the original length, $q'_{ji} = q_{ji}/m$ and therefore, $y'_{ji} = \sum_{l=1}^{i-1} q'_{jl} = y_{ji}/m$.

Combining these observations with (3.3) and (3.4) yields

$$\mathbb{E}\,[\,\text{OPT}(\mathcal{I})\,] \;\geq\; \frac{1}{m}\sum_j w_j \sum_{i=1}^{n_j} \sum_{(k,\ell)\in H'(j,i)} \Pr\left[P_j > y_{ji} \wedge P_k > y_{k\ell}\right]\cdot I_k(q_{k\ell}, y_{k\ell})\,.$$

□

Theorem 3.5 above and Lemma 3.3 imply immediately

Corollary 3.6. *The lower bound on the optimal preemptive policy for parallel machine scheduling on an instance $\mathcal{I}$ equals an m-fraction of the expected value achieved by* Gipp *on the relaxed instance $\mathcal{I}_0$ without release dates but the same processing times to be scheduled on one machine, that is,*

$$\mathbb{E}\left[\,\textsc{Opt}(\mathcal{I})\,\right] \;\geq\; \frac{\mathbb{E}\left[\,\textsc{Gipp}(\mathcal{I}_0)\,\right]}{m}\,. \tag{3.5}$$

3.3.3 A Simple Policy on Parallel Machines

Simple examples show that Gipp is not an optimal policy for scheduling problems with release dates and/or multiple machines. The following policy is a generalization of Gipp to the problem $\mathrm{P}\,|\,r_j, pmtn\,|\,\mathbb{E}\left[\,\sum w_jC_j\,\right]$. The idea is to allow the preemption of processing quanta while the rank of each job gets only recomputed after its current quantum has completed.

Algorithm 10: Follow Gittins Index Priority Policy (F-Gipp)

```
At any time $t$, process an available job $j$ with highest rank $R_j(y_{j,k+1})$,
where $(j,k)$ is the last quantum of $j$ that has completed. Define $k = 0$
if no quantum of job $j$ has been completed.
```

Note that the decision time points in this policy are release dates and any time point when a quantum or a job complete. In contrast to the original Gittins index priority policy, F-Gipp considers only the rank $R_j(y_{ji} = \sum_{k=1}^{i-1} q_{jk})$ that a job had before processing quanta (j,i) even if (j,i) has been processing for some time less than q_{ji}. Informally speaking, the policy F-Gipp updates the ranks only after quantum completions and then follows Gipp.

Theorem 3.7. *The policy* F-Gipp *is a 2-approximation for the preemptive stochastic scheduling problem $\mathrm{P}\,|\,r_j, pmtn\,|\,\mathbb{E}\left[\,\sum w_jC_j\,\right]$.*

Proof. This proof incorporates ideas that we have used in deterministic (online) scheduling in the proof of Theorem 2.3, and which we apply now to the more complex stochastic setting. Fix a realization $p \in \Omega$ of processing times and consider a job j and its completion time $C_j^{\text{F-Gipp}}(p)$. Job j is processing in the time interval $[\,r_j, C_j^{\text{F-Gipp}}(p)\,]$. We split this interval into two disjunctive sets of sub-intervals, $T(j,p)$ and $\overline{T}(j,p)$, respectively. Let $T(j,p)$ denote the set of sub-intervals in which job j is processing and $\overline{T}(j,p)$ contains the remaining sub-intervals. Denoting the total length of all intervals in a set T by $|T|$, we have

$$C_j^{\text{F-GIPP}}(p) \;=\; r_j + |T(j,p)| + |\overline{T}(j,p)| \,.$$

The total length of intervals in $T(j,p)$ is, by definition, p_j. In intervals of the set $\overline{T}(j,p)$, no machine is idle and F-GIPP schedules only quanta with a higher priority than (j,i_p), the final quantum of job j. Thus $|\overline{T}(j,p)|$ is maximized if all these quanta are scheduled between r_j and $C_j^{\text{F-GIPP}}(p)$. This gives an upper bound on the overall length $|\overline{T}(j,p)|$ which is the total sum of all realized quantum lengths on m machines. That yields,

$$C_j^{\text{F-GIPP}}(p) \;\leq\; r_j \,+\, p_j \,+\, \frac{1}{m} \cdot \sum_{\substack{(k,\ell)\in H(j,i_p):\\ p_k > y_{k\ell}}} \min\{q_{k\ell}, p_k - y_{k\ell}\}\,.$$

Following the same arguments as in Lemma 3.3, weighted summation over all jobs and taking expectations on both sides give:

$$\sum_j w_j \mathbb{E}\left[\, C_j^{\text{F-GIPP}} \,\right] \;\leq\; \sum_j w_j \left(r_j + \mathbb{E}\left[\, P_j \,\right] \right) \\ \qquad + \frac{1}{m} \cdot \sum_j w_j \sum_{i=1}^{n_j} \sum_{(k,\ell)\in H'(j,i)} \Pr\left[P_j > y_{ji} \wedge p_k > y_{k\ell}\right] \cdot I_k(q_{k\ell}, y_{k\ell})\,.$$

Finally, we apply the trivial lower bound $\mathbb{E}\left[\,\text{OPT}\,\right] \geq \sum_j w_j(r_j + \mathbb{E}\left[\,P_j\,\right])$ and Theorem 3.5, and the approximation result follows. $\square$

F-GIPP applied to deterministic scheduling instances yields the same schedule as P-WSPT introduced in Section 2.2.3. Thus, it achieves the same performance (Theorem 2.3) as the currently best known online algorithm for the scheduling problem $\text{P}\,|\,r_j, pmtn\,|\sum w_j C_j$. This also implies that for general input instances the approximation factor of 2 is best possible for F-GIPP which follows directly from the deterministic worst-case instance for P-WSPT in Corollary 2.4.

Corollary 3.8. *The approximation ratio of* F-GIPP *is not better than* 2 *for the online problem* $\text{P}\,|\,r_j, pmtn\,|\,\mathbb{E}\left[\sum w_j C_j\,\right]$*, for any given number of machines.*

However, in the case of exponentially distributed processing times and the absence of release dates, F-GIPP coincides with the WSEPT rule which is known to solve various special cases of the problem optimally; see an overview in Section 3.2. In absence of release dates and for general processing times, our policy coincides with GIPP and is thus optimal (Theorem 3.1) even if jobs have individual weights.

3.3.4 An Alternative Approach

In this section we provide an alternative policy for solving the same preemptive stochastic scheduling problem. Again, our policy is based on classical Gipp. In contrast to the previous policy F-Gipp, we now deviate less from the original Gittins index priority rule and, thus, we use more information on the actual state of the set of known, unfinished jobs.

Let us mention in advance, that we do not yield improved performance guarantees; instead, we present an alternative 2-approximative policy. But while the analysis of F-Gipp is tight, we conjecture that our new policy has a much better performance than we prove here. In a way, we find ourselves in the stochastic counterpart of the situation with analyzing the deterministic online algorithms Preemptive Wspt and Wsrpt in Section 2.2.2.

3.3.4.1 Single Machine: Generalized Gittins Index Priority Policy

We consider the preemptive stochastic scheduling problem on a single processor, $1 \,|\, r_j, pmtn \,|\, \mathbb{E}\left[\sum w_j C_j\right]$. As mentioned earlier, Gipp is not an optimal policy even in this single-machine setting because jobs cannot start processing before their release dates which are not considered by Gipp and which may cause unnecessary idle time. A straightforward extension of the policy Gipp is to choose at any time the job with highest rank among the set of available jobs.

Algorithm 11: Generalized Gittins Index Priority Policy (Gen-Gipp)

```
At any time $t$, process an available job $j$ with currently highest
rank, $R_j(y_j(t))$, depending on the amount of processing $y_j(t)$ that the
job $j$ has completed by time $t$.
```

In principle, the jobs are still processed in non-increasing order of maximum ranks as in Gipp and F-Gipp. Applied to an instance with equal release dates, all three policies, Gipp, F-Gipp and Gen-Gipp, yield the same schedule. As in F-Gipp, the generalization in the policy Gen-Gipp concerns the fact that we incorporate release dates and cause preemptions of quanta whereas the Gipp policy preempts jobs only after completion of a quantum. Due to different arrival times, F-Gipp and Gen-Gipp preempt jobs within the processing of a quantum if a job with a higher rank is released. The crucial difference between both policies concerns the time for updating the rank of a preempted quantum: while F-Gipp recomputes the rank of a job only after a quantum has completed, Gen-Gipp considers at any time the actual maximum rank of a job. In that sense, it is an immediate generalization of the original Gittins index priority policy. The interesting question addresses the effect of these rank updates in case of job preemption on the ordering of quanta.

From Proposition 3.2 (a), we know that if a quantum (j, k) is preempted after $\gamma < q_{jk}$ units of processing, the rank of job j has not decreased, that is, $R_j(y_{jk} + \gamma) \geq R_j(y_{jk})$. Hence, all quanta with a lower priority than the original priority of (j, k) available at or after the time that (j, k) is preempted will not be processed before quantum (j, k) is completed.

Consider a realization of processing times $p \in \Omega$ and a job j in the schedule obtained by GEN-GIPP. Let i_p be the index of the quantum in which job j finishes, i. e., $y_{ji_p} < p_j \leq y_{ji_p} + q_{ji_p}$. Then the completion time $C_j^{\text{GEN-GIPP}}$ of job j can be bounded by its release date plus the total length of the quanta that have a higher rank than (j, i_p) at time r_j. This includes also quanta of jobs k with $r_k > r_j$ since they have rank $R_k(0)$ even though they are not available for scheduling.

However, this set of quanta contains not only quanta in $H(j, i_p)$ with a higher priority than (j, i_p). In presence of release dates the following situation is possible: A part of quantum $(k, \ell) \in L(j, i_p)$ is scheduled before quantum (j, i_p), which has higher rank than (k, ℓ), even though job j is available. This happens when job k has been processed for $\gamma \in (y_{k\ell}, y_{k,\ell+1})$ units of time before time r_j and its rank improved (increased) such that $R_k(\gamma) > R_j(y_{ji_p})$. We call this event $\mathcal{E}_{k,ji_p}(\gamma)$ and we say that *job k* or *one of its quanta disturbs* (j, i_p). Formally we define,

$$\begin{aligned}\mathcal{E}_{k,ji}(\gamma) \;=\; \{\, & \text{by time } r_j,\, k \text{ has been processed for } \gamma \text{ units of time} \\ & \text{and } R_k(\gamma) > R_j(y_{ji}) \,\}\,.\end{aligned}$$

The amount of time that the quantum (k, ℓ) disturbs (j, i) is given by

$$q_{k\ell}^{ji}(\gamma) = \max\{q \leq y_{k,\ell+1} - \gamma \;:\; R_k(\gamma + q) > R_j(y_{ji})\}.$$

Note that the event $\mathcal{E}_{k,ji}(\gamma)$ only depends on the (partial) realizations of jobs that have been processed before r_j and is therefore independent of P_j. Furthermore, the amount of processing γ is integral since w.l.o.g. we restricted all release dates and processing times to integer values and therefore GEN-GIPP preempts jobs only at integral points in time.

Now, let us come back to the completion time of a job j in a realization $p \in \Omega$. As stated above, it can be bounded by r_j plus the total sum of quanta that have a higher rank at time r_j. These are

(i) all quanta in $H(j, i_p)$ except of those for which event $\mathcal{E}_{j,k\ell}(\gamma)$ occurs with $p_j \in (\gamma, \gamma + q^{k\ell}_{ji_p}(\gamma)\,]$, i. e., quanta that are disturbed by (j, i_p) with j completing while it is disturbing, and

(ii) the quanta $(k, \ell) \in L(j, i_p)$ for which an event $\mathcal{E}_{k,ji_p}(\gamma)$ occurs for some $\gamma > y_{k\ell}$, i. e., quanta that disturb (j, i_p).

Formalized, that is,

Proposition 3.9. *Given a realization $p \in \Omega$ and a job j, let i_p be the index of the quantum in which this job finishes, i. e., $y_{ji_p} < p_j \leq y_{j,i_p+1}$. Then, the completion time of job j in the* Gen-Gipp *schedule can be bounded by*

$$\begin{aligned} C_j^{\text{Gen-Gipp}}(p) \;\leq\; & r_j \\ & + \sum_{(k,\ell)\in H(j,i_p)\,:\,p_k>y_{k\ell}} \min\{q_{k\ell}, p_k - y_{k\ell}\} && (3.6) \\ & - \sum_{\substack{(k,\ell)\in H(j,i_p):\\ p_k>y_{k\ell}}} \sum_{\substack{\gamma\,:\,\mathcal{E}_{j,k\ell}(\gamma),\\ p_j\in(\gamma,\gamma+q^{k\ell}_{ji_p}(\gamma)]}} \min\{q_{k\ell}, p_k - y_{k\ell}\} && (3.7) \\ & + \sum_{\substack{(k,\ell)\in L(j,i_p):\\ p_k>y_{k\ell}}} \sum_{\substack{\gamma\,:\,\mathcal{E}_{k,ji_p}(\gamma),\\ p_k>\gamma>y_{k\ell}}} \min\{q^{ji_p}_{k\ell}(\gamma), p_k - \gamma\}\,. && (3.8) \end{aligned}$$

Given the above bound for a particular realization, we compute the expected completion time of job j.

Lemma 3.10. *The expected completion time of job j under* Gen-Gipp *can be bounded by*

$$\begin{aligned} &\mathbb{E}\left[\, C_j^{\text{Gen-Gipp}} \right] \\ &\leq r_j + \sum_{i=1}^{n_j} \sum_{(k,\ell)\in H'(j,i)} \Pr\left[P_j > y_{ji} \,\wedge\, P_k > y_{k\ell}\right] \cdot I_k(q_{k\ell}, y_{k\ell}) \\ &- \sum_{i=1}^{n_j} \sum_{\substack{(k,\ell)\\ \in H(j,i)}} \sum_{\gamma=y_{ji}}^{\infty} \Pr\left[\mathcal{E}_{j,k\ell}(\gamma) \wedge \gamma < P_j \leq \gamma + q^{k\ell}_{ji}(\gamma)\right] \cdot \Pr\left[P_k > y_{k\ell}\right] \cdot I_k(q_{k\ell}, y_{k\ell}) \\ &+ \sum_{i=1}^{n_j} \sum_{(k,\ell)\in L(j,i)} \sum_{\gamma=y_{k\ell}}^{\infty} \Pr\left[y_{ji} < P_j \leq y_{j,i+1}\right] \cdot \Pr\left[\mathcal{E}_{k,ji}(\gamma) \wedge P_k > \gamma\right] \cdot I_k(q^{ji}_{k\ell}(\gamma), \gamma). \end{aligned}$$

Proof. The bound in Proposition 3.9 holds for each realization $p \in \Omega$. Taking the expectation over all realizations on both sides we get an upper bound on the expected completion time of a job j scheduled by GEN-GIPP. By linearity of expectation we can consider the sum of expected values of the summands (3.6), (3.7) and (3.8) separately.

Recall that $\chi(\mathcal{E})$ is an indicator random variable which equals 1 if and only if the event $\mathcal{E}$ occurs; furthermore, $\xi_{k\ell}$ denotes the special indicator random variable for the event $P_k > y_{k\ell}$. We show how to transform the expected values of (3.6) to (3.8) such that their sum plus $\mathbb{E}[\, r_j \,]$ equals the claimed expression. The term (3.6) equals exactly the right hand side of equation (3.1) in the proof of Lemma 3.3. In that proof we showed that

$$\mathbb{E}[\,(3.1)\,] = \sum_{i=1}^{n_j} \sum_{(k,\ell)\in H'(j,i)} \Pr[P_j > y_{ji} \wedge P_k > y_{k\ell}]\cdot I_k(q_{k\ell}, y_{k\ell}) = \mathbb{E}[\,(3.6)\,]\,.$$

Similarly, we transform the expected value of

$$\sum_{\substack{i\le n_j:\\ y_{ji}<P_j\le y_{j,i+1}}} \sum_{\substack{(k,\ell)\in H(j,i):\\ P_k>y_{k\ell}}} \sum_{\substack{\gamma:\mathcal{E}_{j,k\ell}(\gamma),\\ P_j\in(\gamma,\gamma+q_{ji}^{k\ell}(\gamma)]}} \min\{q_{k\ell}, P_k - y_{k\ell}\}$$

from (3.7) to

$$\mathbb{E}\left[\sum_{i=1}^{n_j}\chi(y_{ji}<P_j\le y_{j,i+1}) \sum_{\substack{(k,\ell)\in H(j,i):\\ P_k>y_{k\ell}}} \sum_{\substack{\gamma:\mathcal{E}_{j,k\ell}(\gamma),\\ P_j\in(\gamma,\gamma+q_{ji}^{k\ell}(\gamma)]}} \min\{q_{k\ell}, P_k - y_{k\ell}\}\right]$$

$$= \mathbb{E}\left[\sum_{i=1}^{n_j} \sum_{\substack{(k,\ell)\\ \in H(j,i)}} \sum_{\gamma=y_{ji}}^{\infty} \chi\left(\gamma < P_j \le \gamma + q_{ji}^{k\ell}(\gamma) \wedge P_k > y_{k\ell} \wedge \mathcal{E}_{j,k\ell}(\gamma)\right) \right.$$
$$\left. \cdot \min\{q_{k\ell}, P_k - y_{k\ell}\}\right]$$

$$= \sum_{i=1}^{n_j} \sum_{\substack{(k,\ell)\\ \in H(j,i)}} \sum_{\gamma=y_{ji}}^{\infty} \Pr\left[\mathcal{E}_{j,k\ell}(\gamma) \wedge \gamma < P_j \le \gamma + q_{ji}^{k\ell}(\gamma)\right]\cdot \Pr[P_k > y_{k\ell}]\cdot I_k(q_{k\ell}, y_{k\ell}).$$

Finally, the expected value of Term (3.8) can be reformulated in a similar way and therefore we omit the details for showing

$$\mathbb{E}[\,(3.8)\,]$$
$$= \sum_{i=1}^{n_j} \sum_{\substack{(k,\ell)\\ \in L(j,i)}} \sum_{\gamma=y_{k\ell}}^{\infty} \Pr[y_{ji} < P_j \le y_{j,i+1}] \cdot \Pr[\mathcal{E}_{k,ji}(\gamma) \wedge P_k > \gamma] \cdot I_k(q_{k\ell}^{ji}(\gamma), \gamma)\,.$$

The summation of expected values (3.6) to (3.8) concludes the proof. $\square$

Note that in the absence of release dates the events $\mathcal{E}_{j,k\ell}(\gamma)$ do not occur for any job $j \in \mathcal{J}$ and any $\gamma > 0$. Now, we can give the approximation guarantee.

Theorem 3.11. *The policy* Gen-Gipp *is a* 2*-approximation for the problem* $1\,|\,r_j, pmtn\,|\,\mathbb{E}\left[\sum w_j C_j\right]$.

Proof. Denote by $\mathcal{I}$ an instance of the problem $1\,|\,r_j, pmtn\,|\,\mathbb{E}\left[\sum w_j C_j\right]$, and let $\mathcal{I}_0$ be the relaxation of $\mathcal{I}$ in which we assume all release dates are zero. With Lemmata 3.10 and 3.3, we have prepared the ground for bounding the expected objective value, $\mathbb{E}\left[\,\text{Gen-Gipp}\,\right]$, of a schedule that has been obtained by Gen-Gipp.

$$\begin{aligned}\mathbb{E}\left[\,\text{Gen-Gipp}(\mathcal{I})\,\right] &= \sum_{j\in\mathcal{J}} w_j \mathbb{E}\left[\,C_j^{\text{Gen-Gipp}(\mathcal{I})}\,\right]\\ &\le \sum_{j\in\mathcal{J}} w_j r_j + \mathbb{E}\left[\,\text{Gipp}(\mathcal{I}_0)\,\right] + \sum_{j\in\mathcal{J}} w_j\ (O_j - N_j),\end{aligned}$$

where

$$O_j = \sum_{i=1}^{n_j} \sum_{\substack{(k,\ell)\\ \in L(j,i)}} \sum_{\gamma=y_{k\ell}}^{\infty} \Pr\left[y_{ji} < P_j \le y_{j,i+1}\right] \cdot \Pr\left[\mathcal{E}_{k,ji}(\gamma) \wedge P_k > \gamma\right] \cdot I_k(q_{k\ell}^{ji}(\gamma), \gamma)$$

$$N_j = \sum_{i=1}^{n_j} \sum_{\substack{(k,\ell)\\ \in H(j,i)}} \sum_{\gamma=y_{ji}}^{\infty} \Pr\left[\mathcal{E}_{j,k\ell}(\gamma) \wedge \gamma < P_j \le \gamma + q_{ji}^{k\ell}(\gamma)\right] \cdot \Pr\left[P_k > y_{k\ell}\right] \cdot I_k(q_{k\ell}, y_{k\ell}).$$

We claim that $\sum_{j\in\mathcal{J}} w_j\ (O_j - N_j) \le 0$ and give the proof in Lemma 3.12 below. This implies the theorem due to the trivial lower bound on the expected value of an optimal policy Opt, $\mathbb{E}\left[\,\text{Opt}(\mathcal{I})\,\right] \ge \sum_{j\in\mathcal{J}} w_j\, r_j$, and the fact that Gipp is an optimal policy for the relaxed problem instance without release dates $\mathcal{I}_0$ (Theorem 3.1), which gives $\mathbb{E}\left[\,\text{Opt}(\mathcal{I})\,\right] \ge \mathbb{E}\left[\,\text{Gipp}(\mathcal{I}_0)\,\right]$. ☐

Lemma 3.12. *Let*

$$O_j = \sum_{i=1}^{n_j} \sum_{\substack{(k,\ell)\\ \in L(j,i)}} \sum_{\gamma=y_{k\ell}}^{\infty} \Pr\left[y_{ji} < P_j \le y_{j,i+1}\right] \cdot \Pr\left[\mathcal{E}_{k,ji}(\gamma) \wedge P_k > \gamma\right] \cdot I_k(q_{k\ell}^{ji}(\gamma), \gamma)$$

and

$$N_j = \sum_{i=1}^{n_j} \sum_{\substack{(k,\ell)\\ \in H(j,i)}} \sum_{\gamma=y_{ji}}^{\infty} \Pr\left[\mathcal{E}_{j,k\ell}(\gamma) \wedge \gamma < P_j \le \gamma + q_{ji}^{k\ell}(\gamma)\right] \cdot \Pr\left[P_k > y_{k\ell}\right] \cdot I_k(q_{k\ell}, y_{k\ell}),$$

then

$$\sum_{j\in\mathcal{J}} w_j\ (O_j - N_j)\ \le\ 0\,.$$

Proof. In order to prove the claim, we first note that $(j,i) \in H(k,\ell)$, for jobs $k \neq j$, implies $(k,\ell) \in L(j,i)$ and vice versa. Moreover, the event $\mathcal{E}_{j,ji}(\gamma)$ is empty for all i and γ. Thus, we can transform $\sum_{k\in\mathcal{J}} w_k\ N_k$ by rearranging indices:

$$\begin{aligned}
&\sum_{k\in\mathcal{J}} w_k\ N_k \\
=&\sum_{k\in\mathcal{J}}\sum_{l=1}^{n_k}\sum_{\substack{(j,i)\\ \in H(k,\ell)}}\sum_{\gamma=y_{k\ell}}^{\infty} w_k \Pr\left[\mathcal{E}_{k,ji}(\gamma)\wedge \gamma < P_k \leq y_{k,\ell+1}\right]\cdot \Pr\left[P_j > y_{ji}\right]\cdot I_j(q_{ji},y_{ji}) \\
=&\sum_{j\in\mathcal{J}}\sum_{i=1}^{n_j}\sum_{\substack{(j,i)\\ \in L(k,\ell)}}\sum_{\gamma=y_{k\ell}}^{\infty} w_k \Pr\left[\mathcal{E}_{k,ji}(\gamma)\wedge \gamma < P_k \leq y_{k,\ell+1}\right]\cdot \Pr\left[P_j > y_{ji}\right]\cdot I_j(q_{ji},y_{ji}).
\end{aligned}$$

Note that if event $\mathcal{E}_{k,ji}(\gamma)$ occurs then $y_{k,\ell+1} = \gamma + q_{k\ell}^{ji}(\gamma)$. Moreover, by definition of the conditional probability it holds that

$$\Pr\left[y_{ji} < P_j \leq y_{j,i+1}\right] = \Pr\left[P_j > y_{ji}\right]\cdot \Pr\left[P_j \leq y_{j,i+1} \,|\, P_j > y_{ji}\right] ,$$

for any quantum (j,i). Moreover, due to the independence of the processing times, we know that

$$\Pr\left[\mathcal{E}_{k,ji}(\gamma)\wedge \gamma < P_k \leq y\right] = \Pr\left[\mathcal{E}_{k,ji}(\gamma)\wedge P_k > \gamma\right]\cdot \Pr\left[P_k \leq y \,|\, P_k > \gamma\right]$$

for any y. With these arguments we have

$$\begin{aligned}
&\sum_{j\in\mathcal{J}} w_j\,(O_j - N_j) \\
=&\sum_{j\in\mathcal{J}}\sum_{i=1}^{n_j}\sum_{\substack{(k,\ell)\\ \in L(j,i)}}\sum_{\gamma=y_{k\ell}}^{\infty}\Bigg(w_j \Pr\left[y_{ji} < P_j \leq y_{j,i+1}\right]\Pr\left[\mathcal{E}_{j,k\ell}(\gamma)\wedge P_k > \gamma\right]\cdot I_k(q_{k\ell}^{ji}(\gamma),\gamma) \\
&\qquad - w_k \Pr\left[\mathcal{E}_{k,ji}(\gamma)\wedge \gamma < P_k \leq \gamma + q_{k\ell}^{ji}(\gamma)\right]\cdot \Pr\left[P_j > y_{ji}\right]\cdot I_j(q_{ji},y_{ji})\Bigg)
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{j\in\mathcal{J}} \sum_{i=1}^{n_j} \sum_{\substack{(k,\ell)\\ \in L(j,i)}} \sum_{\gamma=y_{k\ell}}^{\infty} \Pr\left[\mathcal{E}_{j,k\ell}(\gamma) \wedge P_k > \gamma\right] \cdot \Pr\left[P_j > y_{ji}\right] \cdot \\
&\qquad\qquad \Big(w_j \Pr\left[P_j \le y_{j,i+1} \,|\, P_j > y_{ji}\right] \cdot I_k(q^{ji}_{k\ell}(\gamma), \gamma) \\
&\qquad\qquad\qquad - w_k \Pr\left[P_k \le \gamma + q^{ji}_{k\ell}(\gamma) \,|\, P_k > \gamma\right] \cdot I_j(q_{ji}, y_{ji}) \Big) \\
&\le 0\,.
\end{aligned}
$$

The final inequality holds because $R_k(q^{ji}_{k\ell}(\gamma), \gamma) \ge R_j(q_{ji}, y_{ji})$ if event $\mathcal{E}_{k,ji}(\gamma)$ occurs and thus,

$$
\begin{aligned}
\frac{w_k \Pr\left[P_k \le \gamma + q^{ji}_{k\ell}(\gamma) \,|\, P_k > \gamma\right]}{I_k(q^{ji}_{k\ell}(\gamma), \gamma)} &= R_k(q^{ji}_{k\ell}(\gamma), \gamma) \\
&\ge R_j(q_{ji}, y_{ji}) \\
&= \frac{w_j \Pr\left[P_j \le y_{j,i+1} \,|\, P_j > y_{ji}\right]}{I_j(q_{ji}, y_{ji})}\,. \qquad \square
\end{aligned}
$$

Note, that on deterministic problem instances, Gen-Gipp schedules at any time the job with highest ratio of weight over remaining processing time, thus it yields the same schedule as the algorithm Wsrpt introduced in Section 2.2.2. For this algorithm we gave a *bad instance* in Theorem 2.2 on which Wsrpt cannot achieve a performance ratio of 1.21 or less. Hence we have immediately the following corollary.

Corollary 3.13. *The policy* Gen-Gipp *does not yield an approximation guarantee of* 1.21 *or less for preemptive stochastic scheduling on a single machine.*

This rather large gap between lower and upper bound raises hope that the best approximation ratio of Gen-Gipp is less than 2. In that case it would improve on the performance of F-Gipp which is exactly 2 (Corollary 3.8). We underpin this conjecture further with the arguments that Gen-Gipp adapts more dynamically to the actual job rank, and moreover, it solves deterministic problem instances with equal weights optimally; in that case Gen-Gipp coincides with Schrage's [Sch68] Srpt rule which is known to find the optimal schedule. In contrast, F-Gipp fails to be optimal in that case as simple examples show.

3.3.4.2 A Randomized Policy on Parallel Machines

In this section we derive a randomized policy for multiple machines that utilizes the single-machine policy GEN-GIPP in a straightforward way. We prove again an approximation guarantee of 2.

Algorithm 12: Randomized Gittins Index Priority Policy (RAND-GIPP)

```
Assign a job at its release date to any of the $m$ machines by choosing one
with probability $1/m$. On each of the machines run the GEN-GIPP policy,
i. e., at any point in time schedule on each machine $i$ the quantum with
currently highest rank among the available, not yet completed jobs that
have been assigned to $i$.
```

Theorem 3.14. *The online policy* RAND-GIPP *is a* 2*-approximation for the preemptive problem on parallel machines,* $\mathrm{P}\,|\,r_j, pmtn\,|\,\mathbb{E}\,[\,\sum w_j C_j\,]$.

Proof. The policy applies the policy GEN-GIPP on each machine, and thus, parts of the performance analysis from the previous section can be recycled. Therefore, we avoid repeating rather complex terms and ask the reader to follow the references.

Consider a realization $p \in \Omega$ of processing times and a job j. Denote by $j \to i$ that job j is assigned to machine i in the considered realization. Since on each machine the single-machine policy GEN-GIPP is executed, the completion time of job j, given that it is processing on machine i, is given in Proposition 3.9 with a minor modification for our current setting, i. e., we sum up only over quanta of jobs that are assigned to the same machine i as job j. We denote the corresponding value by $(3.6)' + (3.7)' + (3.8)'$. Thus, the expected completion time of j over all random choices of the algorithm for assigning jobs $k \neq j$ to machines is

$$\begin{aligned}\mathbb{E}\left[\,C_j^{\mathrm{RG}}(p)\,|\,j \to i\,\right] \;&\leq\; r_j + \sum_{k \in \mathcal{J}} \Pr\left[k \to i\,|\,j \to i\right] \cdot \left[(3.6)' + (3.7)' + (3.8)'\right] \\ &\leq\; r_j + p_j + \frac{1}{m}\left[\,(3.6) + (3.7) + (3.8)\,\right].\end{aligned}$$

Unconditioning the expected completion time for realization p from the fixed machine assignment and using the fact that all jobs are assigned to any machine i with the same probability $1/m$, independently of each other, yield

$$\begin{aligned}\mathbb{E}\left[\, C_j^{\mathrm{RG}}(p) \,\right] &= \sum_{i=1}^{m} \Pr\left[j \to i\right] \cdot \mathbb{E}\left[\, C_j^{\mathrm{RG}}(p) \,|\, j \to i \,\right] \\ &\leq r_j + p_j + \frac{1}{m}\left[\,(3.6) + (3.7) + (3.8)\,\right].\end{aligned}$$

Taking the expectation over all realizations of processing times, $p \in \Omega$, gives, by linearity of expectations, the expected completion time of job j, that is,

$$\mathbb{E}\left[\, C_j^{\mathrm{RG}} \,\right] \leq r_j + \mathbb{E}\left[\, P_j \,\right] + \frac{1}{m}\mathbb{E}\left[\,(3.6) + (3.7) + (3.8)\,\right].$$

We consider now the weighted sum over all jobs. Notice, that we have computed the expected value $\mathbb{E}\left[\,(3.6) + (3.7) + (3.8)\,\right]$ earlier, in the proof of Lemma 3.10. With Lemma 3.12, this value is bounded from above by the expected value of the classical Gipp policy on the relaxed instance without release dates, $\mathbb{E}\left[\,\textsc{Gipp}(\mathcal{I}_0)\,\right]$. More formally, the total expected value of a schedule constructed by Rand-Gipp is

$$\begin{aligned}\mathbb{E}\left[\,\textsc{Rand-Gipp}\,\right] &= \sum_j w_j \mathbb{E}\left[\, C_j^{\mathrm{RG}} \,\right] \\ &\leq \sum_j w_j (r_j + \mathbb{E}\left[\, P_j \,\right]) + \frac{1}{m}\,\mathbb{E}\left[\,\textsc{Gipp}(\mathcal{I}_0)\,\right] \\ &\leq 2 \cdot \mathbb{E}\left[\,\textsc{Opt}\,\right].\end{aligned}$$

The second inequality follows from the trivial lower bound on the optimum value, $\mathbb{E}\left[\,\textsc{Opt}\,\right] \geq \sum_j w_j\,(r_j + \mathbb{E}\left[\, P_j \,\right])$, and from the bound given in Corollary 3.6. $\square$

CHAPTER 4

STOCHASTIC ONLINE SCHEDULING

In this chapter we consider a general model for scheduling with incomplete information. In this model, we naturally combine the main characteristics of online and stochastic scheduling as considered in the previous two chapters. Job processing times are assumed to be stochastic, but in contrast to traditional stochastic scheduling models, we assume that jobs arrive online, and there is no knowledge about the jobs that will arrive in the future. The model incorporates both, stochastic scheduling and online scheduling, as a special case.

The particular problem settings we consider are again preemptive and non-preemptive scheduling with the objective to minimize the total weighted completion times of jobs. We analyze online scheduling policies for that model, and derive performance guarantees that match (and in a special case improve) performance guarantees known for stochastic and online scheduling, respectively. These results do not only justify the general model for scheduling with incomplete information, they also show for certain scheduling problems that solving the problem with stochastic *and* online information, one can achieve the same performance guarantee as when solving the problem having only one type of limited knowledge.

4.1 MODEL AND PRELIMINARIES

In the preceding two chapters, we have considered two main frameworks for dealing with incomplete information in the theory of scheduling, one being *stochastic scheduling* in Chapter 3, and the other is *online scheduling* in Chapter 2. In stochastic scheduling the population of jobs is assumed to be known beforehand, but in contrast to deterministic models, the processing times of jobs are random variables. The actual processing times become known only upon completion of the jobs. The distribution functions of the respective random variables are assumed to be known beforehand. In online scheduling, the assumption is that the instance is presented to the scheduler only piecewise. The actual processing times are usually disclosed upon arrival of a job, and decisions must be made without any knowledge of the jobs to come.

In this chapter, we consider the *stochastic online scheduling* (SOS) model that generalizes both, stochastic scheduling and online scheduling. Like in online scheduling, we assume that the instance is presented to the scheduler piecewise, and nothing is known about jobs that might arrive in the future. Even the number of jobs is not known in advance. Once a job arrives, we assume, like in stochastic scheduling, that the probability distribution of its processing time is disclosed, but the actual processing time remains unknown until the job completes.

The goal is to find an SOS policy that minimizes the expected objective value. Our definition of a stochastic online scheduling policy integrates the traditional definition of stochastic scheduling policies into the setting where jobs arrive online. As described in Section 3.1, a scheduling policy specifies actions at decision times t. In order to decide, such a policy may utilize the complete information contained in the partial schedule up to time t, as well as a priori information about unscheduled jobs, i. e., it must be non-anticipatory. However, a stochastic online scheduling policy is required to be not only non-anticipatory but also online. Thus, at any time, it must not utilize any information about jobs that will be released in the future and it must not use the actual processing times of scheduled (or unscheduled) jobs that have not yet completed.

Generalizing the definitions of an approximative policy for traditional stochastic scheduling (Definition 1.4) and a competitive algorithm in online scheduling (Definition 1.1), we define the performance guarantee of a stochastic online scheduling policy as follows.

Definition 4.1 (Approximative SOS policy)**.** *A stochastic online scheduling policy* (SOS policy) Π *is a* ρ–approximation *if, for some* $\rho \geq 1$*, and all instances* $\mathcal{I}$ *of the given problem,*

$$\mathbb{E}\left[\,\Pi(\mathcal{I})\,\right] \leq \rho\, \mathbb{E}\left[\,\textsc{Opt}(\mathcal{I})\,\right].$$

Here, OPT$(\mathcal{I})$ *denotes an optimal offline stochastic scheduling policy. The value* ρ *is called the* performance guarantee *of policy* Π*, and we call* Π *also* ρ*–approximative policy.*

The policy Π is an online policy without any a priori knowledge of the set of jobs $\mathcal{J}$, and only learns about the existence of any job j upon its release date r_j. It has to compete with an adversary that knows the online sequence of jobs in advance. However, with respect to the processing times P_j of the jobs, the adversary is just as powerful as the policy Π itself, since it does not foresee their actual realizations p_j either.

As in traditional online optimization, the adversary in the proposed Sos model may choose an arbitrary sequence of jobs. These jobs, however, are stochastic, with corresponding processing time distributions. The actual processing times are realized according to exogenous probability distributions. Thus, the best the adversary can do is indeed to use an optimal stochastic scheduling policy in the traditional definition of stochastic offline scheduling policies. In this view, our model somewhat compares to the idea of a *diffuse adversary* as defined by Koutsoupias and Papadimitriou [KP00b]; see Section 1.2.1.3. Since deterministic processing times are contained as a special case, all lower bounds on the approximability known from deterministic online scheduling also hold for the Sos model.

Various (scheduling) problems can be modeled in this stochastic online setting. In the remainder we consider the particular settings of preemptive (Section 4.4) and non-preemptive (Section 4.3) scheduling with the objective to minimize the total weighted completion times of jobs. We derive approximative policies with performance guarantees that match – and in a special case even improve – the bounds known from the traditional stochastic setting.

4.2 Related Work

In the previous two chapters we gave overviews on results concerning problems to minimize the sum of (weighted) completion times in the *stochastic scheduling model* (Section 3.2) and in the *online scheduling model* (Section 2.2.1). In general, online algorithms for clairvoyant scheduling problems are not feasible in the Sos model where processing times are uncertain. In contrast, stochastic policies may run in an online environment. We mentioned in the previous chapter, that non-preemptive stochastic scheduling policies have been derived recently by Möhring et al. [MSU99] and by Skutella and Uetz [SU05]. However, these papers do not address the situation where jobs arrive online. In fact, for their algorithms it is essential that all jobs with their release dates, weights and probability distributions of processing times are given in advance.

A model that combines features of stochastic and online scheduling has also been considered by Chou, Liu, Queyranne, and Simchi-Levi [CLQSL06]. They proved asymptotic optimality of the Online Wsept rule for the single-machine problem $1 \,|\, r_j \,|\, \mathbb{E}\left[\sum w_j C_j\right]$, assuming that the weights and processing times can be bounded from above and below by constants. The definition of the adversary in their paper coincides with our definition. Hence, asymptotic optimality means that the ratio of the expected performance of the Wsept rule over the expected performance of an optimal stochastic scheduling policy tends to 1 as the number of jobs tends to infinity.

Later, Chen and Shen [CS06] extended this result and showed that any non-delaying algorithm[1] is asymptotically optimal for $1 \,|\, r_j \,|\, \mathbb{E}\,[\sum w_j\, C_j\,]$ if weights are bounded and processing times as well as inter-arrival times are identically distributed. Therefore, the asymptotic behavior seems an insensitive and debatable criteria for differentiating algorithms for such problems.

Subsequently to our work and inspired by a recent paper on online scheduling [CW05], Schulz [Sch05] gave a randomized 3-approximation policy for the stochastic online version of $\mathrm{P} \,|\, r_j \,|\, \mathbb{E}\,[\sum w_j\, C_j\,]$, under the assumption of a certain class of processing time distributions. As stated in his paper, a derandomized version of this policy matches our performance guarantee for this special class of processing time distributions.

4.3 Non-Preemptive Stochastic Online Scheduling

We propose simple, combinatorial online scheduling policies for stochastic online scheduling with and without release dates, on a single and on parallel machines. In a *non-preemptive* environment, we derive constant performance bounds for these policies.

For the problem of scheduling on identical parallel machines without release dates, $\mathrm{P} \,|\; |\, \mathbb{E}\,[\sum w_j\, C_j\,]$, we yield the performance guarantee

$$\rho \;=\; 1 + \frac{(m-1)(\Delta+1)}{2m}\,.$$

Here, Δ is an upper bound on the squared coefficients of variation of the processing time distributions P_j, that is, $\mathrm{Var}\,[P_j]/\mathbb{E}\,[\,P_j\,]^2 \leq \Delta$ for all jobs j. This performance guarantee matches the previously best known performance guarantee of Möhring et al. [MSU99]; they obtain the same bound for the performance of the Wsept rule in the traditional stochastic scheduling model. However, we derive this bound in a more restricted setting: We consider a stochastic online model where the jobs are presented to the scheduler sequentially (online-list model), and the scheduler must immediately and irrevocably assign jobs to machines, without knowledge of the jobs to come. Once the jobs are assigned to the machines, the jobs on each machine can

[1]A non-delaying algorithms keeps all machines busy as long as unfinished jobs are available for processing.

be sequenced in any order. We thus show that there exists an Sos policy in this restricted setting that achieves the same performance guarantee as the Wsept rule. Note that the Wsept rule is not a feasible policy in this setting, since it requires the knowledge of all jobs and their processing time distributions beforehand.

For the model with release dates we prove a slightly more complicated performance guarantee that is valid for a class of processing time distributions, which we call δ-NBUE, generalizing the well-known class of NBUE distributions; a definition is given in the following section. For identical parallel-machine scheduling with release dates $\mathrm{P}\,|\,r_j\,|\,\mathbb{E}\,[\sum w_j\,C_j\,]$, and δ-NBUE distributions for processing times, we obtain a performance guarantee of

$$\rho \;=\; 1+\max\left\{1+\frac{\delta}{\alpha}\;,\;\alpha+\delta+\frac{(m-1)(\Delta+1)}{2m}\right\}\,,$$

where α is an arbitrary positive parameter, and again, Δ is an upper bound on the squared coefficients of variation $\mathrm{Var}[P_j]/\mathbb{E}\,[\,P_j\,]^2$ of the processing time distributions P_j. Thus, we obtain for ordinary NBUE distributions, for example, where $\Delta=\delta=1$, a performance guarantee of $\rho<3.62-1/(2m)$. Thereby, we improve upon the previously best known performance guarantee of $4-1/m$ for traditional stochastic scheduling, which was derived for an LP based list scheduling policy [MSU99]. Again, this improved bound holds even though we consider an online model in which the jobs arrive over time, and the scheduler does not know anything about the jobs that are about to arrive in the future. Moreover, for deterministic processing times, where $\Delta=0$ and $\delta=1$, we obtain a performance guarantee $\rho<3.281$, matching the bound from deterministic online scheduling in Section 2.2.4.

For both online paradigms, the online-time model and the online-list model (see Section 1.2.1), our results are in fact achieved by *fixed-assignment policies*. That is, whenever a job is presented to the scheduler, it is immediately and irrevocably assigned to a machine. The sequencing of jobs on the individual machines is in both cases an online version of the traditional Wsept rule; see Section 3.2.

Let us emphasize at this point that all our policies are independent of special probability distribution functions. By model definition, all Sos policies are feasible policies in both, the traditional stochastic scheduling model and in the online scheduling model. The approximation guarantees also carry over directly. Thus, we show for a number of particular scheduling problems that the best known performance guarantees for stochastic and Sos policies match each other. That means that pure stochastic problems can be solved by a general Sos policies with the same (proven) performance as by running

	stochastic (offline) scheduling	(deterministic) online scheduling	stochastic online scheduling
$\mathrm{P}\,|\,|\,\mathbb{E}\,[\sum w_j C_j\,]$	$1+\frac{(m-1)(\Delta+1)}{2m}$ [MSU99]	1.21 [KK86]	$1+\frac{(m-1)(\Delta+1)}{2m}$
$\Delta = 0$	$\frac{3}{2} - \frac{1}{2m}$ [MSU99]	1.21 [KK86]	$\frac{3}{2} - \frac{1}{2m}$
$\mathrm{P}\,|\,r_j\,|\,\mathbb{E}\,[\sum w_j C_j\,]$			
$\Delta = 0$ (det. Alg.)	3.281	2.62 [CW05]	3.281
$\delta = 1$ (det. Alg.)	3.62	2.62 [CW05]	3.62
$\delta = 1$ (rand. Alg.)	3 [Sch05]	2 [SS02a]	3 [Sch05]

Table 4.1: Summary of special scheduling problems in which the best known performance guarantees for stochastic scheduling match our guarantees in the SOS model. The overview also contains the corresponding best known results for online scheduling which only leave a small gap to our more general SOS results. Previous results are marked, whereas those without a reference are proven in this thesis.

a stochastic offline policy. Moreover, the best known competitive ratio is in those cases not much smaller than the approximation guarantees in the stochastic models. Thus, we can say that in certain problem cases, one does not lose much performance by applying SOS policies to deterministic online and stochastic offline problems; Table 4.1 summarizes these problems.

4.3.1 A Lower Bound and a Discussion of Parameters

In order to estimate the performance of an SOS policy, we compare its expected outcome to the expected value of an optimal offline policy. As such a policy as well as its value are usually unknown, we are restricted to use lower bounds.

Möhring, Schulz, and Uetz [MSU99] derived a lower bound for an optimal policy in the traditional stochastic scheduling environment which is by definition also a lower bound for an optimal policy in the SOS model. It is a generalization of the lower bound by Eastman, Even, and Isaacs [EEI64] in Corollary 2.6 in Section 2.2.4 to stochastic processing times. In fact, the lower bound is derived from a stochastic linear programming relaxation, which is based on a class of valid stochastic inequalities derived from a corresponding class of valid inequalities proposed and extensively applied in the deterministic scheduling environment; see, e. g., Queyranne [Que93], Schulz [Sch96a], and Hall, Schulz, Shmoys, and Wein [HSSW97].

The lower bound as well as previous results in the stochastic offline model depend on an additional parameter Δ which is an upper bound on the squared coefficients of variation of the processing time distributions P_j, that is,

$$\mathrm{CV}[P_j]^2 \;=\; \frac{\mathrm{Var}\,[P_j]}{\mathbb{E}\,[\,P_j\,]^2} \;\leq\; \Delta\,, \qquad \text{for all jobs } j\,.$$

Lemma 4.2 (Möhring, Schulz, and Uetz [MSU99]). *For any instance $\mathcal{I}$ of the scheduling problem* $\mathrm{P}\,|\,r_j\,|\,\mathbb{E}\,[\,\sum w_j\,C_j\,]$ *and* $\Delta \geq 0$ *as defined, the expected objective function value for an optimal stochastic (offline) policy is*

$$\mathbb{E}\,[\,\textsc{Opt}(\mathcal{I})\,] \geq \sum_j w_j \sum_{k\in H(j)} \frac{\mathbb{E}\,[\,P_k\,]}{m} \;-\; \frac{(m-1)(\Delta-1)}{2m} \sum_j w_j \mathbb{E}\,[\,P_j\,]\,.$$

Here, we have used another piece of notation that comes handy later again. For a given job $j \in \mathcal{J}$, $H(j)$ denotes the set of jobs that have a higher priority in the order of ratios $w_j/\mathbb{E}\,[\,P_j\,]$, that is,

$$H(j) = \left\{ k \in \mathcal{J} \;\middle|\; \frac{w_k}{\mathbb{E}\,[\,P_k\,]} > \frac{w_j}{\mathbb{E}\,[\,P_j\,]} \right\} \cup \left\{ k \leq j \;\middle|\; \frac{w_k}{\mathbb{E}\,[\,P_k\,]} = \frac{w_j}{\mathbb{E}\,[\,P_j\,]} \right\}.$$

Accordingly, we define $L(j) = \mathcal{J}\backslash H(j)$ as those jobs that have lower priority than j in the order of ratios $w_j/\mathbb{E}\,[\,P_j\,]$. As a tie-breaking rule for jobs k with equal ratio $w_k/\mathbb{E}\,[\,P_k\,] = w_j/\mathbb{E}\,[\,P_j\,]$ we decide depending on the position in the online sequence relative to j; that is, if $k \leq j$ then k belongs to set $H(j)$, otherwise it is included in set $L(j)$. Note that by convention, we assume that $H(j)$ contains also job j.

The performance guarantees we give in this chapter are valid for a class of processing time distributions that we call δ-NBUE, generalizing the well-known class of NBUE distributions (new better than used in expectation).

Definition 4.3 (δ-NBUE). *A non-negative random variable X is δ-NBUE if, for $\delta \geq 1$,*

$$\mathbb{E}\,[\,X - t\,|\,X > t\;\,] \;\leq\; \delta\,\mathbb{E}\,[\,X\,]\,, \quad \text{for all } t \geq 0\,.$$

Ordinary NBUE distributions are therefore 1-NBUE by definition. Examples of ordinary NBUE (or 1-NBUE) distributions are exponential, uniform, or Erlang distributions. For a NBUE random variable X, Hall and Wellner [HW81] showed that the (squared) coefficient of variation is bounded by 1, that is, $\mathrm{Var}[X]/\mathbb{E}\,[\,X\,]^2 \leq 1$. With their techniques, we derive the following upper bound on the coefficient of variation for δ-NBUE distributed variables.

Lemma 4.4. *Let X be a δ-NBUE random variable and let the coefficient of variation of X be denoted by* $\mathrm{CV}[X] = \sqrt{\mathrm{Var}\,[X]}/\mathbb{E}\,[\,X\,]$. *Then*

$$\mathrm{CV}[X]^2 \;\leq\; 2\delta - 1\,.$$

Proof. We prove the lemma for continuous random variables X; the proof for discrete random variables goes along the same lines. Let X be a non-negative δ-NBUE random variable, with cumulative distribution function F and density f.
By definition of conditional expectation, we know that

$$\mathbb{E}\left[X - t\,\middle|\,X > t\right] \;=\; \frac{\int_t^{\infty}(x-t)f(x)\,\partial x}{1-F(t)}\,. \tag{4.1}$$

As $x - t = \int_0^{x-t}\partial y$, we can write the nominator of the right hand side as

$$\begin{aligned}\int_{x=t}^{\infty}(x-t)f(x)\,\partial x \;&=\; \int_{x=t}^{\infty}\int_{y=0}^{x-t} f(x)\,\partial y\;\partial x\\ &=\; \int_{y=0}^{\infty}\int_{x=y+t}^{\infty} f(x)\,\partial x\;\partial y\\ &=\; \int_{x=t}^{\infty} 1-F(x)\,\partial x\,, \end{aligned} \tag{4.2}$$

where the second equality is obtained by changing the order of integration.

As X is δ-NBUE, that is, $\mathbb{E}\,[\,X - t\,|\,X > t\,] \leq \delta\,\mathbb{E}\,[\,X\,]$, it follows from equations (4.1) and (4.2) that

$$\int_{x=t}^{\infty} 1-F(x)\,\partial x \;=\; \mathbb{E}\left[X - t\,\middle|\,X > t\right](1-F(t)) \;\leq\; \delta\,\mathbb{E}\,[\,X\,]\,(1-F(t)). \tag{4.3}$$

By integrating the right hand side of the above inequality over t, we obtain

$$\delta\,\mathbb{E}\,[\,X\,]\int_{t=0}^{\infty} 1-F(t)\,\partial t = \delta\,\mathbb{E}\,[\,X\,]^2. \tag{4.4}$$

Hall and Wellner [HW81, Equality (4.1)] showed that integrating the left hand side of (4.3) over t yields

$$\int_{t=0}^{\infty}\int_{x=t}^{\infty} 1-F(x)\;\partial x\;\partial t = \frac{1}{2}\,\mathbb{E}\left[\,X^2\,\right]. \tag{4.5}$$

Hence, using equations (4.4) and (4.5) in (4.3), we have

$$\mathbb{E}\left[X^2\right] \;\leq\; 2\delta\,\mathbb{E}\left[X\right]^2 .$$

Rearranging terms yields the desired bound on the squared coefficient of variation:

$$\mathrm{CV}[X]^2 \;=\; \frac{\mathbb{E}\left[X^2\right] - \mathbb{E}\left[X\right]^2}{\mathbb{E}\left[X\right]^2} \;\leq\; 2\delta - 1 . \qquad \square$$

With this result we can directly relate our approximation results to previous results on non-preemptive stochastic offline scheduling, which depend on the parameter Δ.

4.3.2 Scheduling on a Single Machine

In this section we consider non-preemptive stochastic online scheduling on a single machine. Inspired by the Algorithm α-Shift-Wspt that we have introduced in Section 2.2.4 for the deterministic online setting on parallel machines, we propose the following scheduling policy.

Algorithm 13: α-Shift-Wsept

Modify the release date r_j of each job j to $r'_j = \max\{r_j, \alpha\,\mathbb{E}\left[P_j\right]\}$, for some fixed $\alpha > 0$. At any time t, when the machine is idle, start the job with highest ratio $w_j/\mathbb{E}\left[P_j\right]$ among all available jobs, respecting the modified release dates. In case of ties, schedule the job with the smallest index first.

We first derive an upper bound on the expected completion time $\mathbb{E}\left[C_j^\alpha\right]$ of a job j when scheduling on a single machine according to the α-Shift-Wsept policy.

Lemma 4.5. *Let all processing times be δ-NBUE. Then the expected completion time of job j under α-Shift-Wsept on a single machine can be bounded by*

$$\mathbb{E}\left[C_j^\alpha\right] \;\leq\; \left(1 + \frac{\delta}{\alpha}\right) r'_j + \sum_{k \in H(j)} \mathbb{E}\left[P_k\right] .$$

Proof. We consider some job j. Let X denote a random variable measuring the remaining processing time of a job that is running at time r'_j, if such a job exists. Otherwise X has value 0. Moreover, for any job k, let ξ_k be an indicator random variable that equals 1 if and only if job k starts processing not earlier than time r'_j, i.e., $\xi_k = 1$ if and only if the start time S^α_k of job k is $S^\alpha_k \geq r'_j$. The start of job j will be postponed beyond r'_j by X, and until there are no more higher priority jobs available. Hence, the expected start time of job j can be bounded by

$$\begin{aligned}\mathbb{E}\left[\, S^\alpha_j \,\right] &\leq \mathbb{E}\Big[r'_j + X + \sum_{k \in H(j)\setminus\{j\}} P_k \xi_k\Big]\\ &= r'_j + \mathbb{E}\left[\, X \,\right] + \sum_{k \in H(j)\setminus\{j\}} \mathbb{E}\left[\, P_k \xi_k \,\right]\\ &\leq r'_j + \mathbb{E}\left[\, X \,\right] + \sum_{k \in H(j)\setminus\{j\}} \mathbb{E}\left[\, P_k \,\right],\end{aligned}$$

where the last inequality follows from the fact that $P_k \xi_k \leq P_k$ for any job k.

Next, we show that $\mathbb{E}\left[\, X \,\right] \leq (\delta/\alpha)\, r'_j$. If the machine just finishes a job at time r'_j or is idle at that time, X has value 0. Otherwise, some job ℓ is in process at time r'_j. Note that this job might have lower or higher priority than job j. Such job ℓ was available at time $r'_\ell < r'_j$, and by definition of the modified release dates, we therefore know that

$$\mathbb{E}\left[\, P_\ell \,\right] \;\leq\; \frac{1}{\alpha} r'_\ell \;<\; \frac{1}{\alpha} r'_j$$

for any such job ℓ. Moreover, letting $t = r'_j - S^\alpha_\ell$, the expected remaining processing time of such job ℓ is $\mathbb{E}\left[\, P_\ell - t \mid P_\ell > t \,\right]$, given that it is indeed in process at time r'_j. Due to the assumption of δ-NBUE processing times, we thus know that

$$\mathbb{E}\left[\, P_\ell - t \mid P_\ell > t \,\right] \;\leq\; \delta\, \mathbb{E}\left[\, P_\ell \,\right] \;\leq\; \frac{\delta}{\alpha} r'_j$$

for any job ℓ that could be in process at time r'_j. Hence, we obtain the bound $\mathbb{E}\left[\, X \,\right] \leq (\delta/\alpha)\, r'_j$. Finally, the fact that $\mathbb{E}\left[\, C^\alpha_j \,\right] = \mathbb{E}\left[\, S^\alpha_j \,\right] + \mathbb{E}\left[\, P_j \,\right]$ concludes the proof. □

In fact, it is quite straightforward to use Lemma 4.5 in order to show the following.

Theorem 4.6. *The α-Shift-Wsept algorithm is a $(\delta + 2)$-approximation for the stochastic online scheduling problem $1|r_j|\mathbb{E}\left[\sum w_j C_j\right]$ for δ-NBUE processing times. The best choice for the parameter α is $\alpha = 1$.*

Proof. We bound the expected value of a single-machine schedule obtained by α-Shift-Wsept using Lemma 4.5 and the definition of modified release dates $r'_j = \max\{r_j, \alpha\,\mathbb{E}[P_j]\}$.

$$\begin{aligned}
&\sum_j w_j\,\mathbb{E}\left[C_j^\alpha\right] \\
&\leq \left(1+\frac{\delta}{\alpha}\right)\sum_j w_j\ \max\{r_j, \alpha\,\mathbb{E}[P_j]\} + \sum_j w_j \sum_{k\in H(j)} \mathbb{E}[P_k] \\
&= \sum_j w_j\ \max\left\{(1+\frac{\delta}{\alpha})\,r_j,\ (\alpha+\delta)\,\mathbb{E}[P_j]\right\} + \sum_j w_j \sum_{k\in H(j)} \mathbb{E}[P_k] \\
&\leq \max\left\{1+\frac{\delta}{\alpha},\ \alpha+\delta\right\}\sum_j w_j\,(r_j+\mathbb{E}[P_j]) + \sum_j w_j \sum_{k\in H(j)} \mathbb{E}[P_k]\,.
\end{aligned}$$

Now, we can apply the trivial lower bound $\sum_j w_j(r_j+\mathbb{E}[P_j]) \leq \mathbb{E}[\textsc{Opt}(\mathcal{I})]$, and exploit the fact that $\sum_j w_j \sum_{k\in H(j)} \mathbb{E}[P_k] \leq \mathbb{E}[\textsc{Opt}(\mathcal{I})]$ by Lemma 4.2 (for $m=1$), and obtain

$$\sum_j w_j\,\mathbb{E}\left[C_j^\alpha\right] \leq \left(1+\max\left\{1+\frac{\delta}{\alpha},\ \alpha+\delta\right\}\right)\mathbb{E}[\textsc{Opt}(\mathcal{I})]\,.$$

Now, $\alpha = 1$ minimizes this expression, independently of δ, and the theorem follows. □

Note that for NBUE processing times, the result matches the currently best known performance bound of 3 derived by Möhring et al. in [MSU99] for the traditional stochastic scheduling model. Their LP-based policy, however, requires an a priori knowledge of the set of jobs $\mathcal{J}$, their weights w_j, and their expected processing times $\mathbb{E}[P_j]$. Moreover, in the deterministic online setting, the best possible algorithm is 2-competitive [AP04, HV96], hence the corresponding lower bound of 2 holds for the stochastic online setting, too.

4.3.3 Scheduling Jobs without Release Dates

Consider stochastic online scheduling on identical parallel machines. In the case that all jobs arrive at time 0, the problem effectively turns into a traditional stochastic scheduling problem $\mathrm{P}\,|\,|\,\mathbb{E}[\sum w_j C_j]$. For that problem, it is known from Möhring et al. [MSU99] that the Wsept rule yields an approximation guarantee of $1+(m-1)(\Delta+1)/(2m)$.

Nevertheless, we consider an online variant of the stochastic scheduling problem $P\,|\,|\,\mathbb{E}\left[\sum w_j C_j\right]$ that resembles the online-list model from online optimization; see Section 1.2.1 for an introduction. We assume that the jobs are presented to the scheduler sequentially, and each job must immediately and irrevocably be assigned to a machine, that is, we require a *fixed-assignment policy*. In particular, during this assignment phase, the scheduler does not know anything about the jobs that are still about to come. Once the jobs are assigned to the machines, the jobs on each machine may be sequenced in any order. We show that an intuitive and simple fixed-assignment policy exists that eventually yields the same performance bound as the one proved in [MSU99] for the WSEPT rule. In this context, recall that the WSEPT rule is not a feasible online policy in the considered online model.

We introduce the following notation: If a job j is assigned to machine i, this is denoted by $j \to i$. Now we can define the MININCREASE policy as follows.

Algorithm 14: MININCREASE (MI)

When a job j is presented to the scheduler, it is assigned to the machine i that minimizes the expression

$$\operatorname{incr}(j,i) = w_j \cdot \sum_{\substack{k\in H(j),\\ k<j,\, k\to i}} \mathbb{E}\left[P_k\right] \quad + \quad \mathbb{E}\left[P_j\right] \cdot \sum_{\substack{k\in L(j),\\ k<j,\, k\to i}} w_k \; + w_j\, \mathbb{E}\left[P_j\right] .$$

Once all jobs are assigned to machines, the jobs on each machine are sequenced in order of non-increasing ratios $w_j/\mathbb{E}\left[P_j\right]$.

Since WSEPT is known to be optimal on a single machine, MININCREASE in fact assigns each job j to the machine where it causes the least increase in the expected objective value, given the previously assigned jobs. This is expressed in the following lemma.

Lemma 4.7. *The expected objective value* $\mathbb{E}\left[\mathrm{MI}(\mathcal{I})\right]$ *of the* MININCREASE *policy equals*

$$\sum_j \min_{i=1,2,\ldots,m} \operatorname{incr}(j,i) .$$

Proof. The assignment of jobs to machines is independent of the realization of processing times. Hence, the expected completion time $\mathbb{E}\left[C_j\right]$ for some job j that has been assigned to machine i_j by MININCREASE is

$$\mathbb{E}\left[C_j\right] = \sum_{k\in H(j),\, k\to i_j} \mathbb{E}\left[P_k\right] ,$$

because all jobs that are assigned to the same machine i_j are sequenced in order of non-increasing ratios $w_j/\mathbb{E}[P_j]$. Now, weighted summation over all jobs gives by linearity of expectation

$$\begin{aligned}\mathbb{E}[\,\mathrm{MI}(\mathcal{I})\,] &= \mathbb{E}\left[\sum_j w_j C_j\right] = \sum_j w_j \sum_{k\in H(j),\,k\to i_j} \mathbb{E}[\,P_k\,] \\ &= \sum_j w_j \sum_{\substack{k\in H(j),\\ k\to i_j,k<j}} \mathbb{E}[\,P_k\,] + \sum_j w_j \sum_{\substack{k\in H(j),\\ k\to i_j,k>j}} \mathbb{E}[\,P_k\,] + \sum_j w_j \mathbb{E}[\,P_j\,]\,.\end{aligned}$$

This allows us to apply the following index rearrangement.

$$\sum_j w_j \sum_{k\in H(j),\,k>j} \mathbb{E}[\,P_k\,] \;=\; \sum_j \mathbb{E}[\,P_j\,] \sum_{k\in L(j),\,k<j} w_k\,. \tag{4.6}$$

Thus, we have

$$\begin{aligned}\mathbb{E}[\,\mathrm{MI}(\mathcal{I})\,] &= \sum_j w_j \sum_{\substack{k\in H(j),\\ k\to i_j,k<j}} \mathbb{E}[\,P_k\,] + \sum_j \mathbb{E}[\,P_j\,] \sum_{\substack{k\in L(j),\\ k\to i_j,k<j}} w_k + \sum_j w_j \mathbb{E}[\,P_j\,] \\ &= \sum_j \Bigg(w_j \sum_{\substack{k\in H(j),\\ k\to i_j,k<j}} \mathbb{E}[\,P_k\,] + \mathbb{E}[\,P_j\,] \sum_{\substack{k\in L(j),\\ k\to i_j,k<j}} w_k + w_j \mathbb{E}[\,P_j\,] \Bigg) \\ &= \sum_j \min_{i=1,2,\ldots,m} \operatorname{incr}(j,i)\,,\end{aligned}$$

where the second equality makes use of (4.6) applied to each individual machine i_j, and the last equality holds since i_j is the machine minimizing $\operatorname{incr}(j,i)$ over all machines $i = 1, 2, \ldots, m$. □

Now, we can derive the following performance guarantee for the MinIncrease policy.

Theorem 4.8. *Consider the stochastic online scheduling problem on parallel machines* $\mathrm{P}|\,|\mathbb{E}[\sum w_j C_j]$ *as described above. Given that for all jobs* j *and some constant* $\Delta \geq 0$ *holds* $\mathrm{Var}[P_j]/\mathbb{E}[\,P_j\,]^2 \leq \Delta$, *the* MinIncrease *policy is a* ρ*-approximation, where*

$$\rho = 1 + \frac{(m-1)(\Delta+1)}{2m}\,.$$

Proof. From Lemma 4.7 we know that $\mathbb{E}[\,\mathrm{MI}(\mathcal{I})\,] = \sum_j \min_i \mathrm{incr}\,(j,i)$ and, thus,

$$\begin{aligned}
&\mathbb{E}[\,\mathrm{MI}(\mathcal{I})\,]\\
&\quad= \sum_j \min_i \Big\{ w_j \sum_{\substack{k\in H(j),\\ k<j,\, k\to i}} \mathbb{E}[\,P_k\,] \;+\; \mathbb{E}[\,P_j\,] \sum_{\substack{k\in L(j),\\ k<j,\, k\to i}} w_k \;+ w_j\mathbb{E}[\,P_j\,]\Big\}\\
&\quad\le \sum_j \frac{1}{m}\Big(w_j \sum_{k\in H(j),\, k<j} \mathbb{E}[\,P_k\,] + \mathbb{E}[\,P_j\,] \sum_{k\in L(j),\, k<j} w_k \Big) + \sum_j w_j \mathbb{E}[\,P_j\,]\,,
\end{aligned}$$

where the inequality holds because the least expected increase is not more than the average expected increase over all machines.

Now, we first apply the index rearrangement (4.6) as in Lemma 4.7, and then plug in the inequality of Lemma 4.2. Using the trivial fact that the sum $\sum_j w_j\mathbb{E}[\,P_j\,]$ is a lower bound for the expected performance $\mathbb{E}[\,\textsc{Opt}(\mathcal{I})\,]$ of an optimal policy, we thus obtain

$$\begin{aligned}
&\mathbb{E}[\,\mathrm{MI}(\mathcal{I})\,]\\
&\quad\le \frac{1}{m}\sum_j \Big(w_j \sum_{k\in H(j),\, k<j} \mathbb{E}[\,P_k\,] + w_j \sum_{k\in H(j),\, k>j} \mathbb{E}[\,P_k\,]\Big) + \sum_j w_j\mathbb{E}[\,P_j\,]\\
&\quad= \frac{1}{m}\sum_j w_j \sum_{k\in H(j)} \mathbb{E}[\,P_k\,] + \frac{m-1}{m}\sum_j w_j\mathbb{E}[\,P_j\,]\\
&\quad\le \mathbb{E}[\,\textsc{Opt}(\mathcal{I})\,] + \frac{(m-1)(\Delta-1)}{2m}\sum_j w_j\mathbb{E}[\,P_j\,] + \frac{m-1}{m}\sum_j w_j\mathbb{E}[\,P_j\,]\\
&\quad\le \Big(1+\frac{(m-1)(\Delta+1)}{2m}\Big)\cdot \mathbb{E}[\,\textsc{Opt}(\mathcal{I})\,]\,. \qquad\square
\end{aligned}$$

As mentioned above, this performance guarantee matches the currently best known performance guarantee for the traditional stochastic setting, which was derived for the performance of the WSEPT policy by Möhring et al. in [MSU99]. The WSEPT policy, however, requires the knowledge of all jobs with their weights w_j and expected processing times $\mathbb{E}[\,P_j\,]$ at the outset. In contrast, the MININCREASE policy decides on machine assignments online, without any knowledge of the jobs to come. Finally, note that these two policies are indeed different; this follows from simple examples.

4.3.3.1 A Lower Bound for Fixed-Assignment Policies

The requirement of a fixed assignment of jobs to machines beforehand may be interpreted as ignoring the additional information that evolves over time in the form of the actual realization of processing times. In the following, we therefore give a lower bound on the expected performance $\mathbb{E}\,[\,\textsc{Fix}(\mathcal{I})\,]$ of an optimal stochastic scheduling policy Fix that assigns jobs to machines beforehand. Obviously, this lower bound holds for the best possible Sos policy, too.

Theorem 4.9. *For stochastic parallel-machine scheduling,* $\mathrm{P}\,|\,|\,\mathbb{E}\,[\sum C_j\,]$, *with unit weights and independent and identically distributed exponential processing times,* $P_j \sim \exp(1)$, *there exist instances* $\mathcal{I}$ *such that*

$$\mathbb{E}\,[\,\textsc{Fix}(\mathcal{I})\,] \;\geq\; 3\,(\sqrt{2}-1)\cdot\mathbb{E}\,[\,\textsc{Opt}(\mathcal{I})\,] - \varepsilon\,,$$

for any $\varepsilon > 0$ *with* $3(\sqrt{2}-1) \approx 1.24$. *Hence, no policy that uses fixed assignments of jobs to machines can perform better in general.*

Note that the theorem is formulated for the special case of exponentially distributed processing times. Stronger bounds can be obtained for arbitrary distributions. However, since our performance guarantees, as in [MSU99], depend on the coefficient of variation of the processing times, we are particularly interested in lower bounds for classes of distributions where this coefficient of variation is small. The coefficient of variation of exponentially distributed random variables equals 1. For example, for the case of $m = 2$ machines, we get a lower bound of $8/7 \approx 1.14$ on the performance of any fixed-assignment policy, and for that case the performance bound of MinIncrease equals $2 - 1/m = 1.5$.

Proof of Theorem 4.9. Consider an instance with m machines and $n = m + k$ exponentially distributed jobs, $P_j \sim \exp(1)$, where $k \geq 1$ is an integer. The optimal stochastic scheduling policy is Sept [BDF81], and the expected performance is (see, e. g., [Uet02, Cor. 3.5.17])

$$\begin{aligned}\mathbb{E}\,[\,\textsc{Opt}(\mathcal{I})\,] \;&=\; \mathbb{E}\,[\,\textsc{Sept}(\mathcal{I})\,] \;=\; \sum_j \mathbb{E}\left[\,C_j^{\textsc{Sept}}\,\right]\\ &=\; m + \sum_{j=m+1}^{n} \frac{j}{m} \;=\; m + k + \frac{k(k+1)}{2m}\,.\end{aligned}$$

When, in a fixed assignment, one machine has to process at least two jobs more than another machine, the assignment can be improved by moving one job from the most loaded machine to the least loaded machine. Therefore, the best fixed-assignment policy tries to distribute the jobs evenly over the machines. That is, it assigns $1+\lfloor\frac{k}{m}\rfloor$ jobs to $m-k+m\lfloor\frac{k}{m}\rfloor$ machines and $1+\lceil\frac{k}{m}\rceil$ jobs to $k-m\lfloor\frac{k}{m}\rfloor$ machines. Hence, there are m jobs with $\mathbb{E}\left[\, C_j \,\right]=\ell$ for each ℓ in the range $1,\dots,1+\lfloor\frac{k}{m}\rfloor$, and $k-m\lfloor\frac{k}{m}\rfloor$ jobs with $\mathbb{E}\left[\, C_j \,\right]=2+\lfloor\frac{k}{m}\rfloor$. The expected performance for the best fixed-assignment policy Fix is

$$\mathbb{E}\left[\, \textsc{Fix}(\mathcal{I}) \,\right] = \sum_j \mathbb{E}\left[\, C_j^{\textsc{Fix}} \,\right] = m + 2k + \left(k - \frac{m}{2} - \frac{m}{2}\left\lfloor \frac{k}{m} \right\rfloor \right) \cdot \left\lfloor \frac{k}{m} \right\rfloor .$$

For $m < k \leq 2m$, the expected value $\mathbb{E}\left[\, \textsc{Fix}(\mathcal{I}) \,\right]$ is equal to $3k$. Hence, for $m < k \leq 2m$, the ratio $\mathbb{E}\left[\, \textsc{Fix}(\mathcal{I}) \,\right]/\mathbb{E}\left[\, \textsc{Opt}(\mathcal{I}) \,\right]$ is

$$\frac{\mathbb{E}\left[\, \textsc{Fix}(\mathcal{I}) \,\right]}{\mathbb{E}\left[\, \textsc{Opt}(\mathcal{I}) \,\right]} = \frac{3k}{m + k + k(k+1)/(2m)} ,$$

which is maximized for $k(m) \in \{\lfloor\sqrt{2}m\rfloor, \lceil\sqrt{2}m\rceil\}$. With this choice of k, the ratio $\mathbb{E}\left[\, \textsc{Fix}(\mathcal{I}) \,\right]/\mathbb{E}\left[\, \textsc{Opt}(\mathcal{I}) \,\right]$ tends to $3\sqrt{2}/(2+\sqrt{2}) = 3(\sqrt{2}-1) \approx 1.24$ as m tends to infinity. $\square$

Note that the lower bound of 1.24 holds whenever m, the number of machines, tends to infinity. For smaller numbers of machines, e.g. $m = 2, 3,$ or 4, we use smaller numbers $k = k(m)$, namely $k(2) = 1, k(3) = 2$, and $k(4) = 2$, and obtain the lower bounds $8/7 \approx 1.14$, $7/6 \approx 1.16$, and $32/27 \approx 1.18$.

4.3.3.2 A Lower Bound for MinIncrease

The lower bound on the performance ratio for *any* fixed-assignment policy given in Theorem 4.9 holds for the MinIncrease policy, too. Hence, MinIncrease cannot be better than 1.24-approximative. For general (i.e., non-exponential) probability distributions we obtain an improved lower bound on the expected performance of MinIncrease relative to the expected performance of an optimal scheduling policy.

Theorem 4.10. *The* MinIncrease *policy does not yield an approximation ratio less than* 3/2 *on general instances of the stochastic online scheduling problem* $\mathrm{P} \,|\, r_j \,|\, \mathbb{E}\left[\, \sum w_j C_j \,\right]$.

Proof. In order to prove the theorem, we provide an instance that consists of $n-1$ deterministic unit length jobs and one job with a stochastic two-point distributed processing time. There are $m = 2$ machines, and we assume that n, the number of jobs, is even. The $n-1$ deterministic jobs have unit weight $w_j = 1$; they appear first in the online sequence. The final job in the online sequence is the stochastic job. It has processing time $p_j = n^2/4$ with probability $2/n$, and $p_j = 1$ with probability $1-2/n$. The weight w_j of the stochastic job equals the value of its expected processing time, that is, $1 - 2/n + n/2$.

The MinIncrease policy assigns $n/2-1$ deterministic jobs to one machine, and $n/2$ deterministic jobs to the other. The stochastic job is assigned to the machine with $n/2-1$ deterministic jobs. Hence, the expected objective value of the schedule under MinIncrease is $\mathbb{E}\left[\sum w_j C_j\right] = 3n^2/4 + o(n^2)$. An optimal stochastic policy would start the uncertain job and one deterministic job at time 0. At time $t = 1$ it is known if the stochastic job has completed, or if it blocks the machine for another $n^2/4 - 1$ time units. If the stochastic job has completed then the remaining unit jobs are distributed equally on both machines, otherwise all deterministic jobs are scheduled on the same machine. Thus, the expected objective value of an optimal schedule is $\mathbb{E}\left[\,\text{Opt}(\mathcal{I})\,\right] = n^2/2 + o(n^2)$. The ratio of both values tends to $3/2$ if the number of jobs tends to infinity. $\square$

Note, however, that this result is less meaningful in comparison to the performance bound of Theorem 4.8, which depends on an upper bound Δ on the squared coefficient of variation.

4.3.4 Scheduling Jobs with Individual Release Dates

In this section, we consider the Sos version of $\mathrm{P} \mid r_j \mid \mathbb{E}\left[\sum w_j C_j\right]$ in the online setting where jobs arrive over time at their release dates. The main idea is to adopt the MinIncrease policy to this setting. However, the difference is that we are no longer equipped with an optimal policy (as it was Wsept in the previous section) to schedule the jobs that are assigned to a single machine. In addition, even if we knew such a policy for a single machine, it would not be straightforward how to use it in the setting with parallel machines to define a feasible online scheduling policy. However, we propose to use the α-Shift-Wsept rule as introduced in Section 4.3.2 to sequence the jobs that we have assigned to the same machine. The assignment of jobs to machines, on the other hand, remains the same as before in the case without release dates. In a sense, when assigning jobs to machines, we thus ignore the possible gain of information that occurs over time in the online-time model.

Algorithm 15: Modified MININCREASE

When a job j is presented to the scheduler at its release date r_j, it is assigned to the machine i that minimizes the expression

$$\operatorname{incr}(j,i) \;=\; w_j \sum_{\substack{k \in H(j),\\ k<j,\, k\to i}} \mathbb{E}[\,P_k\,] \quad + \quad \mathbb{E}[\,P_j\,] \sum_{\substack{k \in L(j),\\ k<j,\, k\to i}} w_k \;+\; w_j \mathbb{E}[\,P_j\,]\,.$$

On each machine, the jobs assigned to this machine are sequenced according to the α-SHIFT-WSEPT rule.

The crucial observation is that the α-SHIFT-WSEPT policy on machine i_j learns about job j's existence immediately at time r_j. Hence, for each single machine, it is indeed feasible to use the α-SHIFT-WSEPT rule, and the so-defined policy is a feasible SOS policy.

Theorem 4.11. *Consider the stochastic online scheduling problem on parallel machines with release dates,* $\mathrm{P}|r_j|\mathbb{E}[\sum w_j C_j]$. *Given that all processing times are δ-NBUE, the modified* MININCREASE *policy running* α-SHIFT-WSEPT *on each single machine is a ρ–approximation, where*

$$\rho = 1 + \max\left\{ 1 + \frac{\delta}{\alpha} \;,\; \alpha + \delta + \frac{(m-1)(\Delta+1)}{2m} \right\}.$$

Allowing δ as the only parameter characterizing the probability distribution of processing times we have a slightly weaker approximation guarantee of

$$\rho = 1 + \max\left\{ 1 + \frac{\delta}{\alpha},\; \alpha + \delta\,\frac{(2m-1)}{m} \right\}.$$

Proof. Consider some job j, and let i_j be the machine to which job j is assigned. Then, by Lemma 4.5 we know that

$$\mathbb{E}[\,C_j\,] \;\le\; \left(1 + \frac{\delta}{\alpha}\right) r'_j + \sum_{k \in H(j),\, k \to i_j} \mathbb{E}[\,P_k\,], \tag{4.7}$$

and the expected value of MININCREASE can be bounded by

$$\mathbb{E}[\,\mathrm{MI}(\mathcal{I})\,] \;\le\; \left(1 + \frac{\delta}{\alpha}\right) \sum_j w_j r'_j + \sum_j w_j \sum_{k \in H(j),\, k \to i_j} \mathbb{E}[\,P_k\,]\,. \tag{4.8}$$

For the second part of the right hand side of (4.8), we can use the same index rearrangement as in the proof of Lemma 4.7; see (4.6). We thus obtain

$$\begin{aligned}
&\sum_j w_j \sum_{k \in H(j),\, k \to i_j} \mathbb{E}[\,P_k\,] \\
&\quad = \sum_j \Bigg(w_j \sum_{\substack{k \in H(j),\\ k \to i_j,\, k<j}} \mathbb{E}[\,P_k\,] \;+\; \mathbb{E}[\,P_j\,] \sum_{\substack{k \in L(j),\\ k \to i_j,\, k<j}} w_k \;+\; w_j\, \mathbb{E}[\,P_j\,] \Bigg).
\end{aligned}$$

By definition of the modified MinIncrease algorithm, we know that any job j is assigned to the machine, which minimizes the term in parentheses. Hence, by the same averaging argument as before, we know that

$$\begin{aligned}
&\sum_j w_j \sum_{k \in H(j),\, k \to i_j} \mathbb{E}[\,P_k\,] \\
&\quad \le \sum_j \Bigg(w_j \sum_{k \in H(j),\, k<j} \frac{\mathbb{E}[\,P_k\,]}{m} + \mathbb{E}[\,P_j\,] \sum_{k \in L(j),\, k<j} \frac{w_k}{m} + w_j\, \mathbb{E}[\,P_j\,] \Bigg) \\
&\quad = \sum_j w_j \sum_{k \in H(j)} \frac{\mathbb{E}[\,P_k\,]}{m} \;+\; \frac{m-1}{m} \sum_j w_j\, \mathbb{E}[\,P_j\,]\,,
\end{aligned}$$

where the last equality again follows from index rearrangement. Plugging this into (4.8), leads to the following bound on the expected performance of MinIncrease:

$$\mathbb{E}[\,\mathrm{MI}(\mathcal{I})\,] \le \Big(1+\frac{\delta}{\alpha}\Big) \sum_j w_j r'_j + \sum_j w_j \sum_{k \in H(j)} \frac{\mathbb{E}[\,P_k\,]}{m} + \frac{m-1}{m} \sum_j w_j \mathbb{E}[\,P_j\,]\,.$$

Applying the bound of Lemma 4.2 into the above inequality, we obtain

$$\begin{aligned}
&\mathbb{E}[\,\mathrm{MI}(\mathcal{I})\,] \\
&\quad \le \Big(1+\frac{\delta}{\alpha}\Big) \sum_j w_j r'_j + \mathbb{E}[\,\textsc{Opt}(\mathcal{I})\,] + \frac{(m-1)(\Delta+1)}{2m} \sum_j w_j \mathbb{E}[\,P_j\,] \\
&\quad = \mathbb{E}[\,\textsc{Opt}(\mathcal{I})\,] + \sum_j w_j \bigg(\Big(1+\frac{\delta}{\alpha}\Big) r'_j + \frac{(m-1)(\Delta+1)}{2m} \mathbb{E}[\,P_j\,] \bigg), \qquad (4.9)
\end{aligned}$$

where again, Δ is an upper bound on the squared coefficient of variation of the processing time distributions P_j. By bounding r'_j by $r_j + \alpha\mathbb{E}[\,P_j\,]$, we obtain the following bound on the term in parentheses of the right hand side of (4.9).

$$\begin{aligned}\Big(1+\frac{\delta}{\alpha}\Big)\,r'_j \;+\; &\frac{(m-1)(\Delta+1)}{2m}\,\mathbb{E}\,[\,P_j\,] \\ &\le\;\Big(1+\frac{\delta}{\alpha}\Big)r_j+\Big(\alpha+\delta+\frac{(m-1)(\Delta+1)}{2m}\Big)\mathbb{E}\,[\,P_j\,] \\ &\le\;\Big(r_j+\mathbb{E}\,[\,P_j\,]\Big)\max\Big\{1+\frac{\delta}{\alpha}\,,\;\alpha+\delta+\frac{(m-1)(\Delta+1)}{2m}\Big\}\,.\end{aligned}$$

By using this inequality in equation (4.9), and applying the trivial lower bound of $\sum_j w_j(r_j+\mathbb{E}\,[\,P_j\,])\le\mathbb{E}\,[\,\textsc{Opt}(\mathcal{I})\,]$ on the expected optimum performance, we get the claimed performance bound of

$$\rho \;=\; 1+\max\Big\{1+\frac{\delta}{\alpha}\,,\;\alpha+\delta+\frac{(m-1)(\Delta+1)}{2m}\Big\}\,.$$

Since all processing times are δ-NBUE, we know by Lemma 4.4 that $\Delta\le 2\delta-1$, and thus, the second claim of the theorem follows. □

For NBUE processing times, where $\Delta=\delta=1$, Theorem 4.11 yields a performance bound of $\rho=2+\max\{\frac{1}{\alpha},\;\alpha+\frac{m-1}{m}\}$. This term is minimal for the parameter $\alpha=(\sqrt{5m^2-2m+1}-m+1)/(2m)$, which gives the following Corollary.

Corollary 4.12. *Consider the* SOS *version of the problem,* $\mathrm{P}|r_j|\mathbb{E}\,[\sum w_j\,C_j\,]$. *Given that all processing times are NBUE, the modified* MININCREASE *policy yields for the parameter* $\alpha=(\sqrt{5m^2-2m+1}-m+1)/(2m)$ *an approximation ratio of*

$$\rho \;=\; 2+\frac{\sqrt{5m^2-2m+1}+m-1}{2m} \;<\; \frac{5+\sqrt{5}}{2}-\frac{1}{2m} \;\approx\; 3.62-\frac{1}{2m}\,.$$

This result improves upon the previously best known bound of $4-1/m$ by Möhring et al. [MSU99] for the traditional stochastic (offline) problem. However, subsequently to our work, Schulz [Sch05] gave for this problem a randomized 3-approximative policy. A derandomized version of his policy yields a performance guarantee that matches the approximation ratio of MININCREASE in Corollary 4.12 above.

More generally, for δ-NBUE processing times, Theorem 4.11 yields the performance guarantee $\rho=\max\{1+\delta/\alpha,\;\alpha+\delta(2m-1)/m\}$, which can be minimized by choosing α properly.

Corollary 4.13. *Consider the stochastic online scheduling problem on parallel machines,* $\mathrm{P}|r_j|\mathbb{E}\left[\sum w_j C_j\right]$. *Given that all processing times are* δ*-NBUE, the modified* MinIncrease *policy yields, for the best choice of* α*, an approximation ratio of*

$$\rho \;<\; \frac{3}{2} + \frac{\delta\,(2m-1)}{2m} + \frac{\sqrt{4\delta^2+1}}{2}\,.$$

Moreover, for deterministic processing times, where $\Delta = 0$ and $\delta = 1$, Theorem 4.11 yields $\rho = 2 + \max\{1/\alpha,\ \alpha + m - 1/(2m)\}$. Optimizing for α leads to the final corollary.

Corollary 4.14. *Consider the online version of the problem* $\mathrm{P}\,|\,r_j\,|\sum w_j C_j$. *The stochastic policy* MinIncrease *yields on deterministic problem instances an approximation ratio of*

$$\rho \;=\; 2 + \frac{\sqrt{17m^2-2m+1}+m-1}{4m} \;<\; 3.281\,,$$

for any number of machines m*, if* $\alpha = (\sqrt{17m^2-2m+1} - m + 1)/(4m)$.

This result leaves only a small gap to the currently best known bound of 2.62 for deterministic online scheduling by Correa and Wagner [CW05]. In fact, it matches the competitive ratio of the deterministic parallel-machine version of α-Shift-Wsept, namely α-Shift-Wspt, as introduced in Section 2.2.4.

4.3.5 Randomized Job Assignment to Parallel Machines

As a matter of fact, the MinIncrease policy can be interpreted as the derandomized version of a policy that assigns jobs uniformly at random to the machines. Random job assignment ignores available information; nevertheless, this method is known to be quite powerful in obtaining good approximation algorithms for scheduling [SS02b] and other optimization problems [MR95]. Schulz and Skutella [SS02b], for example, apply a random assignment strategy, based on the solution of an LP-relaxation, for scheduling jobs with deterministic processing times on unrelated machines. For the special case of identical machines, their approach corresponds to assigning jobs uniformly at random to the machines. The random assignment strategy for the stochastic online scheduling problem at hand is as follows.

Algorithm 16: RANDASSIGN

When a job is presented to the scheduler, it is assigned to machine i with probability $1/m$ for all $i = 1, \ldots, m$. The jobs assigned to machine i are scheduled according to the α-SHIFT-WSEPT policy.

Theorem 4.15. *Consider the stochastic online scheduling problem on parallel machines with release dates,* $\mathrm{P}|r_j|\mathbb{E}\left[\sum w_j C_j\right]$. *Given that all processing times are δ-NBUE, the* RANDASSIGN *policy is a ρ–approximation, where*

$$\rho = 1 + \max\left\{1 + \frac{\delta}{\alpha}\ ,\ \alpha + \delta + \frac{(m-1)(\Delta+1)}{2m}\right\} .$$

Proof. Consider a job j and let i denote the machine to which it has been assigned. Let $\Pr\left[j \to i\right]$ be the probability for job j being assigned to machine i. Then, by Lemma 4.5 we know that

$$\begin{aligned}
\mathbb{E}\left[C_j \,|\, j \to i\right] &\le \left(1 + \frac{\delta}{\alpha}\right) r'_j + \sum_{k \in H(j)} \Pr\left[k \to i \,|\, j \to i\right] \cdot \mathbb{E}\left[P_k \,|\, j \to i\right] \\
&= \left(1 + \frac{\delta}{\alpha}\right) r'_j + \sum_{k \in H(j)} \Pr\left[k \to i \,|\, j \to i\right] \cdot \mathbb{E}\left[P_k\right] .
\end{aligned}$$

The probability that a job is assigned to a certain machine is equal for all machines, i.e., $\Pr\left[j \to i\right] = 1/m$ for all $i = 1, \ldots, m$, and for any job j. Unconditioning the expected value of job j's completion time yields

$$\begin{aligned}
\mathbb{E}\left[C_j\right] &= \sum_{i=1}^{m} \Pr\left[j \to i\right] \cdot \mathbb{E}\left[C_j \,|\, j \to i\right] \\
&\le \left(1 + \frac{\delta}{\alpha}\right) r'_j + \sum_{i=1}^{m} \Pr\left[j \to i\right] \cdot \sum_{k \in H(j)} \Pr\left[k \to i \,|\, j \to i\right] \cdot \mathbb{E}\left[P_k\right] \\
&= \left(1 + \frac{\delta}{\alpha}\right) r'_j + \sum_{k \in H(j)} \frac{\mathbb{E}\left[P_k\right]}{m} + \frac{m-1}{m}\,\mathbb{E}\left[P_j\right] ,
\end{aligned}$$

where the last equality is due to the independence of the job assignments to the machines. Then, the expected value of RANDASSIGN, $\mathbb{E}\left[\mathrm{RA}(\mathcal{I})\right]$, can be bounded by

$$\begin{aligned}
\mathbb{E}\left[\mathrm{RA}(\mathcal{I})\right] &= \sum_j w_j \mathbb{E}\left[C_j\right] \\
&\le \left(1 + \frac{\delta}{\alpha}\right) \sum_j w_j r'_j + \sum_j w_j \sum_{k \in H(j)} \frac{\mathbb{E}\left[P_k\right]}{m} + \frac{m-1}{m} \sum_j w_j \mathbb{E}\left[P_j\right].
\end{aligned}$$

This bound equals the upper bound that we achieved on the expected performance of the (modified) MinIncrease policy in the proof of Theorem 4.11. Hence, we conclude the proof in the same way and with the same result as for MinIncrease. □

4.4 Preemptive Stochastic Online Scheduling

We consider now *preemptive* stochastic online scheduling to minimize the total weighted completion time. We argue that the results obtained in Chapter 3 on traditional stochastic (offline) scheduling transfer to the more general Sos model.

An optimal policy in the Sos model coincides by construction with an optimal offline policy in the traditional stochastic scheduling model. And thus, all lower bounds on an expected optimal value utilized to give performance guarantees in Chapter 3 still hold in our generalized model.

Moreover, the given stochastic policies are feasible policies in the Sos model where jobs arrive online. They employ the Gipp policy in different ways; roughly speaking, their decisions are based on the Gittins index or rank of each job, which is a dynamic value that depends on the probability distribution of the job's processing time and the information about the current status of the job in the schedule. Thus, Gipp itself and each of the proposed extensions F-Gipp, Gen-Gipp, and Rand-Gipp are not only non-anticipative, which is enforced by the stochastic scheduling model, but also online. At no point in time any of the policies uses information of jobs that will be released in future.

Thus, Theorem 3.11 in Section 3.3 implies directly the following result.

Corollary 4.16. *The single-machine policy* Gen-Gipp *is a* 2*-approximation for the* Sos *version of the problem* $1 \,|\, r_j, pmtn \,|\, \mathbb{E}\,[\sum w_j C_j\,]$.

For the deterministic online version of this problem, Sitters [Sit04] derived an algorithm with a competitive ratio of 1.56 which creates only a small gap to the approximation guarantee of 2 for the general Sos policy.

Furthermore, from Theorems 3.7 and 3.14 follows immediately the following corollary.

Corollary 4.17. *Consider the* Sos *version of the stochastic scheduling problem* $\mathrm{P} \,|\, r_j, pmtn \,|\, \mathbb{E}\,[\sum w_j C_j\,]$. *Both policies,* F-Gipp *and* Rand-Gipp*, yield an approximation ratio of* 2.

This performance guarantee of 2 matches the currently best known result in deterministic online scheduling on parallel machines, $\mathrm{P} \,|\, r_j, pmtn \,|\, \sum w_j C_j$, see Theorem 2.3 in Section 2.2.3, even though we consider a more general model. Thus, one does not give up (provable) performance by using the SOS policy for solving a purely online or stochastic problem instance.

For the sake of completeness we add Table 4.2 with a short overview of the results in all models. Notice that in the preemptive scheduling environment, the approximation guarantees in the stochastic and SOS model match independently of the processing time distributions.

	stoch. scheduling	online scheduling	SOS
$1 \mid r_j, pmtn \mid \mathbb{E}[\sum w_j C_j]$	2	1.56 [Sit04]	2
$\mathrm{P} \mid r_j, pmtn \mid \mathbb{E}[\sum w_j C_j]$	2	2	2

Table 4.2: Performance results of the currently best known algorithms for preemptive scheduling on a single and on parallel machines in all three models for dealing with incomplete information. Online scheduling results are only feasible for deterministic problem instances. In contrast, all other results hold for general stochastic problems with discrete processing time distributions.

Chapter 5

An Online Deadline–Tsp or How to Whack Moles

We consider a problem in which requests with deadlines arrive online over time at points of a metric space. One or more servers move in the metric space at constant speed and serve requests by reaching the corresponding position before the deadline. The goal is to guide the servers through the metric space such that the number of served requests is maximized.

We study the online problem on the real line and on the uniform metric space. While on the line no deterministic algorithm can achieve a constant competitive ratio, we provide competitive algorithms for the uniform metric space. Our online investigations are complemented by complexity results for the offline problem.

5.1 Model and Preliminaries

A server can move in a metric space at constant speed starting at a designated origin. Requests arrive online over time at any point in the space. Each request has a deadline and asks for service before this deadline. A request is satisfied (or served) if the server has reached the point of the request in the space by its deadline. The goal of an algorithm is to guide the server through the metric space such that the number of served requests is maximized.

This problem describes exactly the popular *whack-a-mole-game* which is common on carnivals or as a computer game. In this game moles pop up at certain holes from under the ground and, after some time, disappear again. The player is equipped with a hammer and her goal is to hit as many moles as possible while they are above the ground. Clearly, from the viewpoint of the player this game is an online optimization problem since the times and positions where moles will peek out are not known in advance. She has to decide without knowledge of the future which of the currently visible moles to whack and how to move the hammer into a “promising” position. The main questions are, what is a good strategy for whacking moles – if there exists any? And how much better could one perform if one had magical powers and knew in advance where moles will show up?

This online problem has also been investigated under the name *dynamic traveling repair problem* by Irani, Lu and Regan [ILR04]. We prefer not to adopt this name because of its misleading similarity to the problem name (*dynamic*) *traveling repairman problem*. The latter is typically reserved for (stochastic) variants of the TSP with the objective of minimizing the total waiting time (or flow time) of all requests; compare with the brief introduction in Section 1.1.2. In contrast, the objective in our problem setting is to maximize the number of requests that are served before their deadline. We emphasize that our problem significantly distinguishes from other variants of the (online or dynamic) TSP or TRP by the presence of deadlines and the absence of the requirement that all requests must be satisfied.

Furthermore, Bansal, Blum, Chawla, and Meyerson [BBCM04] considered the whack-a-mole problem in an offline environment under the name *vehicle routing with time-windows*. Typically vehicle routing describes a much more general problem class where vehicles with bounded capacity deliver objects from a source to a destination; see, e. g., the book by Toth and Vigo [TV02]. In contrast, our problem corresponds to the special case where source and destination coincide. In addition, the simpler problem where all requests are known from time zero on, is called *deadline-TSP* in the offline investigations in [BBCM04]. We feel that the name *online deadline-TSP* could match our whack-a-mole problem quite well. Therefore, we entitle this chapter with both names but will stick in the remaining part of the chapter to our originally chosen problem name.[1]

Examining the problem, it turned out that two parameters play a crucial role for the analysis: the time $T_j \geq 0$ that a mole j stays above ground and the maximum number N of moles that can be in one hole simultaneously. We restrict our investigations to a uniform popup duration for all moles and denote it by $T = T_j$ for all moles j.

Then the whack-a-mole problem with popup duration T and mole-per-hole limit N (briefly $\text{WHAM}_{T,N}$) can be formulated in mathematical terms as follows: We are given a metric space $M = (X, d)$ with a distinguished origin $0 \in X$ and a sequence $\sigma = (r_1, \ldots, r_m)$ of requests (moles). A server (hammer) moves on the metric space M at unit speed. It starts at the origin at time 0. Each request $r_j = (t_j, p_j)$ specifies a *release time* t_j and a point (hole) $p_j \in X$ where the mole pops up. A request r_j is served

[1]Bansal et al. [BBCM04] published their work subsequently but without knowing of our [KMV04, GKMV06] and Irani et al.'s work [ILR04].

if the server reaches the point p_j in the time interval $[t_j, t_j + T]$. We will refer to $t_j + T$ as the *deadline* of the request. A server can serve more than one but no more than N requests at once. The goal is to whack as many moles as possible. If no confusion can occur we write only Wham instead of $\text{Wham}_{T,N}$.

Our contributions are twofold. First, we provide complexity results for the offline problem offline-Wham. In Section 5.3 we derive a dynamic program for offline-Wham on unweighted graphs with integral release times and deadlines, which runs in polynomial time if the maximum degree of the graph, Δ, and the popup duraton, T, are bounded by a polynomial in the input size. We complement this result by showing that the problem is $\mathcal{NP}$-hard for arbitrary values of T.

Our main contribution lies in the analysis of the *online* problem Wham. We show in Section 5.4 that no deterministic algorithm for Wham on the line can achieve a constant competitive ratio. This unfortunate situation remains true even if one allows randomization. However, if the line is truncated to certain finite intervals, the situation changes and bounded competitive ratios can be shown; Table 5.1 displays the results.

	upper bound	lower bound	
$L \leq T/4$	1	1	
$L = T$	$3T + 1$	$\max\{T+1, \lfloor 3T/2 \rfloor\}$	
$L > T$	—	"∞"	(same as on $\mathbb{R}_+$ and $\mathbb{R}$)

Table 5.1: Lower and upper bounds on the competitive ratio for $\text{Wham}_{T,N=1}$ on the truncated line $[-L, L]$. These results are accomplished in Section 5.4.

From the viewpoint of the *whack-a-mole* player, the situation on the uniform metric space is better than on the line; see Table 5.2 for a summary of the results for this case that are accomplished in Section 5.5.

	upper bound		lower bound	
$T \geq 2$	$\dfrac{\lceil \lfloor T/2 \rfloor + T \rceil}{\lfloor T/2 \rfloor} \in [3,5]$	(see Figure 5.3 for a plot)	2	
$T = 1$	2	(for all t_j integral)	2	(for all t_j integral)
	$2N$	(for general t_j)	$2N$	(for general t_j)

Table 5.2: Lower and upper bounds on the competitive ratio for $\text{Wham}_{T,N}$ on the uniform metric space; see Section 5.5 for the corresponding investigations.

We conclude our study of online algorithms by showing how our results extend to the case of multiple servers in Section 5.6.

5.2 Related Work and Discussion of Results

In earlier work on WHAM under the name *dynamic traveling repair problem*, Irani, Lu, and Regan [ILR04] give two deterministic algorithms for $\text{WHAM}_{T,N}$ in general metric spaces with competitive ratios that are formulated in terms of the diameter of the metric space. Their ratios translated into the notation used in this thesis and restricted to the uniform metric space are $\frac{3T}{T-2}$ and $4\left\lceil\frac{2T}{T-1}\right\rceil\left(\left\lceil\frac{2T}{T-1}\right\rceil+1\right)$, respectively.

We improve these results in the following ways: For the restricted line segment $[-T,T]$ and the uniform metric space with $T=1$ our algorithms are the first competitive ones, since the bounds of [ILR04] cannot be applied. Moreover, for the uniform metric space we decrease known competitive ratios substantially. For instance, for popup duration $T=2$, our algorithm IWTM achieves a competitive ratio of 3, while the results in [ILR04] yield a ratio of 80. Surprisingly all of our competitiveness results are obtained by simple algorithms.

In terms of lower bounds, Irani et al. [ILR04] show that there is a metric space in which no deterministic algorithm can achieve a constant competitive ratio. We show that this results is already true on the real line and against more restricted adversary models.

Investigations on the offline version of WHAM have been carried out by Bansal, Blum, Chawla, and Meyerson [BBCM04] on general metric spaces and for general deadlines. They yield an $\mathcal{O}(log^2 n)$-approximation where n denotes the number of requests. The competitiveness results for both of our online algorithms on the uniform metric space translate into approximation results for the corresponding offline setting.[2] Thus, by considering this particular metric space and restricted time windows (deadlines), we improve the general result in [BBCM04] in special cases.

The problem WHAM is a special case of the *online traveling salesman problem* (OLTSP) with an unusual objective function. Typically, the OLTSP is studied for objectives as minimizing the makespan [AKR00, AFL+01, FS01], the weighted sum of completion times [FS01, KdPPS03], and the maximum/average flow time [HKR00, KLL+02]. The latter problems are also known as *online traveling repairman problem*; see Section 1.1.2 for an introduction of problems in this class. In contrast, WHAM's objective is to maximize the number of requests served before their deadlines (whereby not

[2]In general, in online optimization, there is no requirement on the computational complexity of algorithms. (See a discussion of this issue in Section 1.2.1.) In fact, the algorithm Replan that we apply for WHAM on the line, in Section 5.4.2, does not run efficiently. However, this is the only online algorithm in this thesis that makes use of this relaxed assumption – any other online algorithm runs in polynomial time, and thus, their competitive ratios translate directly into approximation ratios.

all requests need to be served).

Traveling salesman problems can be viewed as generalizations of certain scheduling problems. For example, the relation between the original TSP and scheduling problems with setup costs is well-known; we refer to Section 1.1.2 for more details. In [BHS01], Baruah, Haritsa, and Sharma show that no deterministic algorithm can achieve a constant competitive ratio for the scheduling problem of maximizing the number of jobs completed before their deadlines. Kalyanasundaram and Pruhs [KP03] give two deterministic algorithms and show that for every instance at least one of them has constant competitive ratio. Thus, they provide a randomized algorithm which is constant competitive for the online scheduling problem of $1 \mid r_j, pmtn \mid \sum U_j$. However, it is not clear whether and how their results carry over to the more general class of WHAM.

Another related area which has gained a lot of interest in recent years concerns *buffer management problems* that occur when storing packets in Quality-of-Service (QoS) networks. Informally speaking, the problem for a single queue is as follows: packets of different values arrive online over time and may enter a buffer of bounded capacity or may be dropped. At the end of a time slot, one packet from the buffer can be transmitted. An algorithm has to decide which packets to deliver and which packets to drop. The goal is to transmit the set of packets with highest total value.

In the particular problem called *s-uniform delay buffer problem*, there is a fixed bound s on the delay of a packet, that is, the difference between deadline and release date of a packet has exactly the value s. This problem is equivalent to online scheduling of unit-size jobs with individual deadlines, $1 \mid r_j, p_j = 1 \mid \sum w_j U_j$.

Observe that the special case with equal weights is very similar to the whack-a-mole problem where at most one *mole* can peek out of a hole at the same time, $\text{WHAM}_{T,N=1}$, on the uniform metric space and with integral release dates. Bansal et al. [BFK$^+$04] present a 1.75-competitive algorithm for the weighted scheduling or buffer management problem.[3] However, results do not directly transfer – indeed, the mentioned result even seems to contradict our lower bound – since by our definition of WHAM a server could satisfy 2 requests "at once", one in the moment of its release and the other one at its deadline. Clearly, this problem setting is discussible; nevertheless, it is the model we investigate in the following sections.

[3]Actually, Bansal et al. [BFK$^+$04] investigated a different buffer model in which packets are delivered in strict FIFO-order. However, it is known that a ρ-competitive algorithm for the FIFO model can be translated into a competitive online algorithm for the s-uniform delay buffer problem with the same performance [KLM$^+$04, KMvS05].

5.3 The Complexity of Offline Whack-a-Mole

In this section we investigate the complexity of the offline problem OFFLINE-WHAM where all moles and their respective release dates are known in advance. We first give a polynomial-time algorithm for a special case of the problem. Then, we show that OFFLINE-WHAM is $\mathcal{NP}$-hard on the line.

In this section we slightly diverge from the notation used for the online problem in allowing more general deadlines $d_j \geq t_j$ for the requests than just $d_j = t_j + T$, where T is the popup duration. In this more general context T will denote the maximum popup duration of any mole. We also allow a weight $n_j \geq 0$ to be associated with request r_j. The goal of the problem becomes to maximize the weight of the whacked moles.

5.3.1 When Whacking Is Easy

We consider the following scenario: The metric space $M = (X, d)$ has n points and is induced by an undirected unweighted graph $G = (V, E)$ with $V = X$, i.e., for each pair (x, y) of points in the metric space M, $d(x, y)$ equals the shortest path length in G between vertices x and y. We also assume that for each mole the release date $t_j \geq 1$ and the deadline d_j are integers.

Theorem 5.1. *Suppose that the metric space is induced by an unweighted graph of maximum degree Δ. Then, the* OFFLINE-WHAM *with integral release dates t_j and deadlines d_j can be solved in time $\mathcal{O}(nm(T + m)(\Delta + 1)^{2T})$, where $T := \max_{1 \leq j \leq m}(d_j - t_j)$ is the longest time a mole stays above the ground; n denotes the number of vertices in the graph and m is the total number of requests.*

Proof. The time bound claimed is achieved by a simple dynamic programming algorithm. Let $0 < t_1 < t_2 < \cdots < t_k$ with $k \leq m$ be the (distinct) times where moles show up. We set $t_0 := 0$.

The idea for a dynamic programming algorithm is the following: For each relevant point t in time and each vertex $v \in V$ we compute the maximum number of moles caught subject to the constraint that at time t we end up at v. Essentially the only issue in the design of the algorithm is how one keeps track of moles that have been whacked "on the way". The key observation is that for any time t that we consider the only moles that need to be carefully accounted for are those that have popped up in the time interval $[t - T, t]$. Any mole that popped up before time $t - T$ will have disappeared at time t anyway.

Given a vertex v, a *history track* is a sequence $s = (v_1, v_2, \ldots, v_k = v)$ of vertices in G such that for $i = 1, \ldots, k$ we have $d(v_i, v_{i+1}) = 1$ whenever $v_i \neq v_{i+1}$. We define the time-span of the history track s to be $\bar{d}(s) = k$. The history track s encodes a route starting at vertex v_1 at some time t, walking along edges of the graph and ending up at v at time $t + \bar{d}(s)$ with the interpretation if $v_i = v_{i+1}$ we remain at vertex v_i for a unit of time. Notice that in an unweighted graph with maximum degree at most Δ, there are at most $(\Delta + 1)^L$ history tracks of length $L \in \mathbb{N}$ ending at a specific vertex v.

Given the concept of a history track, the dynamic programming algorithm is straightforward. For $t \in \{t_0, \ldots, t_k\}$, $v \in V$ and all history tracks s, with $\bar{d}(s) = \min(t, T)$, ending in v at time t, we define $M[t, v, s]$ to be the maximum number of moles hit in any solution that starts in the origin at time 0, ends at v at time t, and follows the history track s for the last $\bar{d}(s)$ units of time.

The values $M[0, v, s]$ are all zero, since no mole raises its head before time 1. Given all the values $M[t, v, s]$ for all $t = t_0, \ldots, t_{j-1}$, we can compute each value $M[t_j, v, s]$ easily.

Assume that $t_j \leq t_{j-1} + T$. Then, from the history track s we can determine a vertex v' such that the server must have been at vertex v' at time t_{j-1}. This task can be achieved in time $\mathcal{O}(T)$ by backtracking s. The value $M[t_j, v, s]$ can be computed from the $\mathcal{O}((\Delta + 1)^T)$ values $M[t_{j-1}, v', s']$ by adding the number of moles whacked and subtracting the number of moles accounted for twice. The latter task is easy to achieve in time $\mathcal{O}(T + m)$ given the history tracks s and s'. Hence, the time needed to compute $M[t_j, v, s]$ is $\mathcal{O}((T + m)(\Delta + 1)^T)$.

It remains to treat the case that $t_j > t_{j-1} + T$. Let $t := t_{j-1} + T$. Notice that no mole can be reached after time t and before time t_j, since all moles released no later than t_{j-1} will have disappeared by time t. Any solution that ends up at vertex v at time t_j must have been at some vertex v' at time t. We first compute the "auxiliary values" $M[t, v', s']$ for all $v' \in V$ and all history tracks s by the method outlined in the previous paragraph. Then, the value $M[t_j, v, s]$ can be derived as the maximum over all values $M[t, v', s']$, where the maximum ranges over all vertices v' such that v can be reached by time t_j given that we are at v' at time t and given the histories s and s' (which must coincide in the relevant part).

Since the dynamic programming table has $\mathcal{O}(nm(\Delta + 1)^T)$ entries, the total time complexity of the algorithms is $\mathcal{O}(nm(T + m)(\Delta + 1)^{2T})$. □

The above dynamic program can easily be adjusted for metric spaces induced by weighted graphs with integral edge weights. Each edge e is then replaced by a path of $w(e)$ vertices, where $w(e)$ denotes the length of edge e. The time bound for the above procedure becomes then $\mathcal{O}(\bar{n}m(T+m)(\Delta+1)^{2T})$, where $\bar{n} = n + \sum_{e\in E}(w(e)-1)$. Hence, whenever $(\Delta+1)^{2T}$ is pseudo-polynomially bounded, offline-Wham can be solved in pseudo-polynomial time on these weighted graphs.

5.3.2 When Whacking Is Hard

It follows from Theorem 5.1 that offline-Wham can be solved in polynomial time if $(\Delta+1)^{2T}$ is bounded by a polynomial in the input size. On the other hand, the problem on a graph with unit edge weights, all release times zero and all deadlines equal to n, the number of holes, contains the Hamiltonian Path Problem as a special case. Thus, it is $\mathcal{NP}$-hard to solve.

Another special case of the offline-Wham is obtained when at most one mole is in a hole at a time, the metric space is the line and release dates as well as deadlines are general. Tsitsiklis [Tsi92] showed that on this metric space the traveling salesman or repairman problem with general time windows constraints is $\mathcal{NP}$-complete. This implies that the offline-Wham on the line with general release dates and deadlines is $\mathcal{NP}$-hard.

In his proof, Tsitsiklis uses the fact that the length of the time windows may vary per request. This raises the question whether offline-Wham on the line with uniform popup durations is $\mathcal{NP}$-hard. In the following theorem we show that this is the case if one allows arbitrary weights to be associated with the moles.

Theorem 5.2. offline-Wham *on the line is $\mathcal{NP}$-hard even if the time moles stay above ground is equal for all moles, i.e., $d_i - t_i = d_j - t_j = T$ for all requests r_i, r_j.*

Proof. We show the theorem by a reduction from Partition, which is well known to be $\mathcal{NP}$-complete [Kar72, GJ79]. An instance of Partition consists of n items $a_i \in \mathbb{Z}^+$, $i = 1,\dots,n$, with $\sum_i a_i = 2B$. The question is whether there exists a subset $S \subset \{1,\dots,n\}$, such that $\sum_{i\in S} a_i = B$.

Given an instance $\mathcal{I}$ of Partition, we construct an instance $\mathcal{I}_{\text{Wham}}$ for offline-Wham, with $m = 3n$ requests. Let $B = \frac{1}{2}\sum_i a_i$ and $K = B + 1$. The time each mole stays above ground is $T = 2B$. There are $2n$ requests r_i^+ and r_i^-, $i = 1,\dots,n$ which are released at time $(2i-1)K$ and have deadline $(2i-1)K + T$. The position of r_i^+ is $K + a_i$ with weight $K + a_i$, and the position of r_i^- equals $-K$ with weight K. Finally, there are n requests r_i^0 at the origin, where r_i^0 is released at time $2iK$, has deadline $2iK + T$, and weight K.

We claim that at least $2nK + B$ moles can be whacked if and only if $\mathcal{I}$ is a YES-instance for PARTITION.

Let S be a partition of $\mathcal{I}$, i.e., $\sum_{i \in S} a_i = B$. Then whacking the moles in the order $(r_1^{\alpha_1}, r_1^0, \ldots, r_n^{\alpha_n}, r_n^0)$, where $\alpha_i = +$ if $i \in S$ and $\alpha_i = -$ if $i \notin S$, is feasible and yields the desired bound, as tedious computation can show.

Suppose conversely that there exists a route for the whacker such that it reaches at least $2nK + B$ moles. Notice that as the locations of the holes of requests r_i^+ and r_i^- are at least $2K > 2B$ apart, the whacker can whack at most one of these requests. The moles of requests r_i^+ and r_i^- pop up after time $t_{i-1}^+ + T$, and therefore the whacker cannot catch the moles of request r_{i-1}^+ and r_i^+ at the same time. The same is true for requests r_{i-1}^0 and r_i^0. Suppose the whacker moves to the hole of r_i^+ or r_i^- after first whacking the moles of r_i^0. The earliest possible arrival time in the hole of r_i^+ or r_i^- is at least $2iK + K = (2i+1)K$ and by this time the moles of r_i^+ and r_i^- have gone down again. Hence, when whacking r_i^0 and either r_i^+ or r_i^-, the request r_i^+ or r_i^- need to be whacked before r_i^0. If the whacker doesnot whack the moles of request r_i^0, for some i, she catches at most $(2n-1)K + 2B < 2nK + B$ moles. The same holds for the case that the whacker doesnot whack any of the moles of requests r_i^+ and r_i^-. Therefore, the whacker needs to reach all moles popping up at the origin and for each i it also needs to whack all moles of either r_i^+ or r_i^-. Hence, by the above considerations we know that when at least $2nK + B$ moles are whacked, the whacker needs to hit first the moles of r_i^+ or r_i^- and then those of r_i^0 before going to the hole of request r_{i+1}^+ or r_{i+1}^-.

Let $S = \{i : \text{moles of } r_i^+ \text{ are whacked}\}$ be the set of requests served in the positive part of the line. We claim that $\sum_{i \in S} a_i = B$. Obviously $\sum_{i \in S} a_i \geq B$ since the total weight of moles whacked is at least $2nK + B$. Suppose that $\sum_{i \in S} a_i > B$ and let $S' \subseteq S$ be the smallest subset of S such that if $i, j \in S$ with $i < j$ and $j \in S'$ then $i \in S'$ and $\sum_{i \in S'} a_i > B$ and let $k = \max\{i : i \in S'\}$. Then $\sum_{i \in S' \setminus \{k\}} a_i \leq B$. The whacker leaves the origin for request r_k^+ at time $2(k-1)K + 2\sum_{i \in S' \setminus \{k\}} a_i \leq 2(k-1)K + 2B < t_k^0$. The next time the whacker reaches the origin is $2kK + \sum_{i \in S'} a_i > 2kK + 2B$ and by then the moles of request r_k^0 have gone under ground. Hence, it cannot reach the moles of request r_k^0 and is not able to whack $2nK + B$ moles. $\square$

5.4 Whack-a-Mole on the Line

In this and the following section we investigate the existence of competitive algorithms for WHAM. Our lower bound results are not only established for the standard adversary, the optimal offline algorithm, but also for more restricted adversaries. We stress that our competitiveness results hold for the stronger standard adversary.

5.4.1 Lower Bounds – How Well We Can't Whack

Already simple examples show that online algorithms perform badly, when compared to an all-powerful adversary.

Theorem 5.3. *Let $T \geq 0$ be arbitrary. No deterministic online algorithm can achieve a constant competitive ratio for* $\text{WHAM}_{T,N}$ *on the line.*

Proof. Consider an online algorithm ALG and assume w.l.o.g. that its position at time T is $p_{\text{ALG}}(T) \leq 0$. The sequence consists of one mole, which pops up at time T at position $T+1$. As the adversary can be at position T by time T, it has reached the mole by time $T+1$. ALG, on the other hand, can not be in $T+1$ before time $2T+1$ and by then the mole has gone under ground again. $\square$

In the proof above the adversary abuses its power in the sense that it moves to a point where a mole will pop up without revealing this information to the online whacker.

We mentioned in Section 1.2.1, that several adversaries exist in the literature that are restricted in their power. Krumke et al. [KLL+02] introduced the so-called *non-abusive adversary* that is defined on the line; it may only move in a certain direction if there is still a pending request in this direction. For WHAM we extend this definition.

Definition 5.4 (*Non-abusive adversary on the line*)**.** *A* non-abusive adversary *on the line may only move in a certain direction if there is still a pending request in this direction, and if it can reach it before the deadline. The adversary may also* go home*, that means, it may move back to the origin.*

Theorem 5.3 can be extended to the non-abusive adversary. In the following theorem, we give a stronger lower bound on the half line $\mathbb{R}_+$ which clearly implies the same bound for the complete line $\mathbb{R}$.

Theorem 5.5. *Let $T \geq 0$ be arbitrary. No deterministic online algorithm can achieve a constant competitive ratio for* $\text{WHAM}_{T,N}$ *on the half line $\mathbb{R}_+$ even against a non-abusive adversary. This result continues to hold even if $N = 1$.*

Proof. We prove the theorem for popup duration $T = 1$. This proof can be extended to general T by multiplying each position and release time by T. Moreover, our construction only uses a mole-per-hole-limit of $N = 1$.

For any integral constant $c > 2$, we show that there exists a request sequence on which the adversary whacks at least c times as many moles as any deterministic online algorithm. This implies the theorem. The adversarial sequence consists of at most three parts.

σ_1: At each integral time point t, a mole appears in hole $t+1$.
σ_2: At each integral time point, a mole pops up in hole 0.
σ_3: At each integral time point t, a mole is released in hole $t_0+t+1-\lceil t_1 \rceil$.

The sequence starts at time $t = 0$ with σ_1. Let $t_0 \leq c$ be the first integral point in time at which the position of the algorithm's whacker at time t, $p_{\text{ALG}}(t) \neq t$, or $t_0 = c$ if no such $t \leq c$ exists.

If $p_{\text{ALG}}(t_0) \neq t_0$, then the adversary continues with subsequence σ_1 up to time at least $c\,t_0$. As $p_{\text{ALG}}(t_0) < t_0$, the online algorithm cannot reach any of the moles released at or after time t_0, and catches at most t_0 moles. The adversary, on the other hand, can whack all moles, by always moving to the right, and serves at least $c\,t_0$ requests.

If $p_{\text{ALG}}(t_0) = t_0$, that means, $t_0 = c$, then the adversary stops subsequence σ_1, after time $t_0 - 1$, and continues the sequence with subsequence σ_2 beginning at time t_0. Let $t_1 \leq c^2 + c + 1$ be the first time $c < t < c^2 + c + 1$ at which $p_{\text{ALG}}(t) = 1$, or $t_1 = c^2 + c + 1$ if no such time $c < t < c^2 + c + 1$ exists.

If $t_1 = c^2 + c + 1$, then the sequence stops at this time. The algorithm has not whacked any of the moles released in subsequence σ_2, and thus, it has served at most $t_0 = c$ moles. The adversary, by staying at the origin, has reached all moles of subsequence σ_2, and thus, it has whacked at least c^2 moles.

If $t_1 < c^2 + c + 1$, then the adversary stops the subsequence σ_2 at time $\lceil t_1 - 1 \rceil$ and continues with σ_3 starting at time t_1. As $p_{\text{ALG}}(t_1) = 1 < t_0$, the algorithm cannot reach any of the requests released in the third subsequence, nor has it served any of the requests in σ_2. Hence, it has whacked at most c moles. The adversary, can reach all moles of the third subsequence, as well as all moles of the first one, if it is moving to the right during the first subsequence, and is remaining in hole t_0 during the second one. By continuing σ_3 for at least $c^2 - c$ time units, the adversary whacks at least c^2 moles. □

The negative results above raise the question whether randomized algorithms can perform better.

Theorem 5.6. *For any $T \geq 0$, no randomized algorithm achieves a constant competitive ratio for* $\textsc{Wham}_{T,N}$ *on the line $\mathbb{R}$ against an oblivious adversary. The result remains true for $N = 1$.*

Proof. For the sake of a contradiction, suppose that there exists a c-competitive randomized online algorithm, for some constant c. Let $K = \lceil c \rceil + 1$, and consider the holes $x_i = i(2T + 1)$, for $i = 0, 1, \ldots, K - 1$. The adversarial sequence consists of only one mole, released at time $t = (K - 1)(2T + 1)$. Let p_i denote the probability that the randomized whacker is within distance T of hole x_i at time t. As the distance between the holes is more than $2T$, these probabilities sum up to $\sum_{0 \leq i \leq K-1} p_i \leq 1$. Therefore, there is at least one hole x_i where the algorithm's whacker is in reachable distance with probability $p_i \leq 1/K$. At time t the adversary releases one mole in this hole x_i. While the adversary certainly catches that mole, the expected value for the algorithm is at most $1/K < 1/c$ which is a contradiction to the assumption of a c-competitive algorithm. □

The lower bound results above suggest to restrict the metric space further. In the sequel, we consider the truncated line, $[-L, L]$. Before we embark on lower and upper bound proofs, let us rule out the easy cases. If $L > T$, then no constant competitive ratio can be achieved as a very simple one-mole-sequence shows: suppose the whacker of an online algorithm is at the origin or on the left of it at time T then release one mole in a position larger than T. On the other hand, if $L \leq T/4$, then a trivial algorithm which continuously moves between the end points of the line segment is able to reach each request in time and is therefore optimal. Hence, the only interesting case is $T/4 < L \leq T$.

We consider the problem on the restricted line $[-T, T]$ with unit distances between holes, that is, on $[-T, T] \cap \mathbb{Z}$. Observe that the dynamic program proposed in Section 5.3.1 solves the related offline-problem efficiently for constant popup duration T and integral release dates. In the following theorem we assume for ease of notation that T is integral. The result can be easily transferred to non-integral values, increasing the competitive ratio by at most 1. We simply replace T by $\lfloor T \rfloor$ and adjust the release dates.

Theorem 5.7. *Let $T \in \mathbb{N}$. No deterministic online algorithm for* $\textsc{Wham}_{T,N}$ *on the line segment $[-T, T] \cap \mathbb{Z}$ can achieve a competitive ratio less than*

$$\max\{\, NT + 1,\ N \left\lfloor \frac{3T}{2} \right\rfloor \},$$

even against a non-abusive adversary.

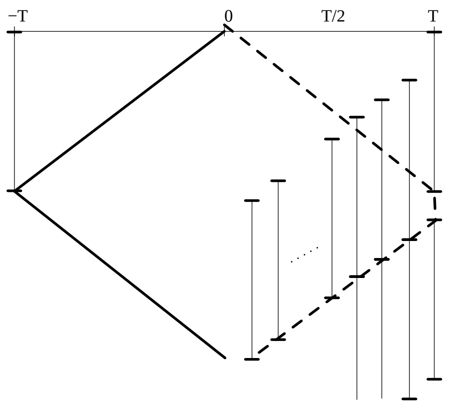


Figure 5.1: Lower bound instance for any deterministic online algorithm Alg on the truncated line $[-T, T]$; dashed line: adversary's tour; solid line: Alg's tour. Each request is represented by a vertical line between the release date and deadline.

Proof. In each boundary position, T and $-T$, one mole is released at time 0. If an online algorithm does not catch a mole by time T then the instance stops. Otherwise, assume w.l.o.g. that it started heading towards $-T$ (the movement must be started at time 0). Then, at time $t = 1$ another $N - 1$ moles are released in hole T. Moreover, at each of the times $t = 2, \ldots, T$, N moles get released in holes $T + 1 - t$ and at time $t = T + 1, \ldots, \lfloor 3T/2 \rfloor$ N moles appear in holes $2T + 1 - t$. If at any time, the algorithm's whacker changes its direction, the sequence stops and the algorithm cannot catch any mole. The construction is illustrated in Figure 5.1 on the following page.

A simple calculation shows that no online algorithm is able to catch any of the moles in $[1, T] \cap \mathbb{Z}$ (hence it whacks only a single mole in $-T$) while the adversary whacks all moles on the positive part of the line. Thus, $N(\lfloor 3T/2 \rfloor)$ is a lower bound on the competitive ratio.

The fact that $NT + 1$ is another lower bound on the competitive ratio is even easier to establish. After two initial moles, with weight 1, are released at time 0 in $-T$ and T, a second wave of T moles, each with weight N, is released at time T in $1, 2, \ldots, T$ (this assumes that the algorithm moves to the left initially). $\square$

5.4.2 Upper Bounds – How Well We Can Whack

As mentioned in the previous section, Wham on the truncated line $[-L, L]$, with $L \leq T/4$ or $L > T$, are trivially easy or have been shown to be hopeless. In this section, we only consider the line segment $[-T, T]$, for which we have shown a lower bound of $\max\{NT+1, N\lfloor 3T/2 \rfloor\}$ on the competitive ratio. For this case, we analyze the following algorithm which is probably folklore; it can be found also under different names like Reopt or Optimal [AKR00, GKR01].

```
Algorithm 17: Replan (Rp)
At any time when a new request becomes known, compute an
optimal route on all pending requests, with start in the current
position and ending at the origin. Change the current route if
and only if another route allows to whack more moles.
```

Theorem 5.8. Rp *achieves a competitive ratio of* $3T+1$ *for* $\text{Wham}_{T,N=1}$ *on the line segment* $[-T,T]\cap\mathbb{Z}$ *when at most one mole can be peeking out of a hole.*

Proof. Assume that Rp is not c-competitive; we prove the theorem by yielding a contradiction for $c = 3T+1$. Let $\text{Alg}(\sigma)$ denote the number of served requests for an algorithm Alg on a sequence σ. Denote σ as a smallest sequence for which $\text{Rp}(\sigma) < \frac{1}{c}\text{Opt}(\sigma)$. Partition σ into $\sigma = \sigma' \cup \sigma_c$, where σ_c consists of the last c requests. Since $\text{Rp}(\sigma') \leq \text{Rp}(\sigma)$, we thus yield,

$$\text{Rp}(\sigma') \leq \text{Rp}(\sigma) < \frac{1}{c}\,\text{Opt}(\sigma) \leq \frac{1}{c}\,(\text{Opt}(\sigma') + c) \leq \text{Rp}(\sigma') + 1.$$

Due to the integrality of $\text{Rp}(\sigma)$ and $\text{Rp}(\sigma')$, we know that $\text{Rp}(\sigma) = \text{Rp}(\sigma')$. By definition of Rp, that means that the algorithm does not change its route for any request in σ_c and it does not whack any of those moles. On the other hand, the optimal offline algorithm must serve all requests in σ_c. Otherwise we could remove unwhacked requests from σ_c without changing the routes or solution values of Rp and Opt and thus, the sequence σ was not a smallest sequence satisfying $\text{Rp}(\sigma) < \frac{1}{c}\ \text{Opt}(\sigma)$

In the remainder we show that Rp serves at least one request of σ_c, for $c \geq 3T+1$, which is a contradiction to the previous observations, and thus, we disprove the assumption that Rp were not c-competitive for $c \geq 3T+1$.

Let t_{max} be the latest release date of moles in the sequence σ' whacked by Rp. Consider the last mole r_ℓ whacked by Rp and denote the time of whacking by C_ℓ and the position of the mole by p_ℓ, respectively. Assume w.l.o.g. that $p_\ell \geq 0$; the other case is symmetric. Note, that $C_\ell \leq t_{max} + T$, and the time when the whacker returns at the origin is $C_\ell + p_\ell \leq t_{max} + 2T$.

By definition of σ_c, all requests $r_j \in \sigma_c$ have release dates $t_j \geq t_{max}$. If there is any request released at or after Rp 's return to the origin, then the whacker is able to catch it, which leads to the contradiction. Therefore, we assume that $t_{max} \leq t_j < C_\ell + p_\ell$ for all $r_j \in \sigma_c$.

After time C_ℓ, there is no reachable pending mole from σ' for Rp. Hence, if there is a request $r_j \in \sigma_c$ at a distance of at most T from p_ℓ with $t_j \geq C_\ell$, then Rp catches at least one of these moles. This holds at least for all points on the non-negative half line. Hence, in holes of the interval $[0, T]$ excluding the hole p_ℓ cannot be more than one mole per hole in sequence σ_c, which sum up to at most T moles. On the negative part of the line, at most two moles per hole can be released in the time interval $[t_{max}, C_\ell + p_\ell)$ of length less than $2T$.

Hence, the number of moles in σ_c is not more than $3T$, and we have a contradiction for any $c \geq 3T + 1$. This completes the proof of a competitive ratio of $c = 3T + 1$ for Rp. $\square$

We note that the above result directly generalizes to an upper bound of $3NT + 1$ on the competitive ratio of Rp for a general mole-per-hole limit of N.

5.5 Whack-a-Mole on the Uniform Metric Space

Recall that the uniform metric space is induced by a complete graph with unit edge weights. The number of vertices in this graph is n. Observe that for popup duration $T < 1$ trivially no deterministic algorithm can be competitive against the standard adversary. In case of the non-abusive adversary, the situation is trivial again, since the adversary can never catch any mole except for those at the origin.

5.5.1 Lower Bounds – How Well We Can't Whack

A lower bound of 2 for Wham on the uniform metric space has been derived earlier by Irani, Lu, and Regan [ILR04]. We give a different proof for the same bound. However, our construction uses fewer nodes in the metric space and, more important, there is no positive amount of time where more than one request is available at a single hole. Also, note that the lower bounds are shown against the most restricted adversary, the non-abusive one. Here, we use a natural extension of Definition 5.4 to the uniform metric space.

Definition 5.9 (Non-abusive adversary on the uniform metric space)**.** *A* non-abusive adversary *on the uniform metric space may only move on a direct shortest path to a pending request whose deadline can be met. The adversary may also* go home, *that means, it may move back to the origin.*

Theorem 5.10. *Let $n \geq 3T + 2$, that is, $T \leq (n-2)/3$. No deterministic online algorithm for* $\text{Wham}_{T,N=1}$ *can achieve a competitive ratio smaller than* 2 *against a non-abusive adversary.*

Proof. The idea for our instance is the following: we make sure that for any integral time $t \geq T$ two moles have deadline t and a third mole is released in one of these two holes, such that an optimal offline algorithm can whack two moles per time unit but an online algorithm catches at most one.

At each time $t = 0, \dots, T-1$ the adversary releases two moles: one in position $p_1(t) = 2t+1$ and the other in $p_2(t) = 2t+2$. At time $t = T, T+1, \dots$ three moles are released: two moles are released in empty holes $p_1(t)$ and $p_2(t)$ and the third mole, either, in position $p_3(t) = p_2(t-T)$ if the online algorithm is in $p_1(t-T)$ at time t, or, in $p_3(t) = p_1(t-T)$, otherwise. Note that at time t, at most $3T$ moles have deadline at least t, and as $n \geq 3T+2$, there are at least two holes left with no moles at time t. □

In case of $N \geq 1$ the above lower bound can be improved.

Theorem 5.11. *No deterministic online algorithm for* $\text{Wham}_{T=1,N}$ *has a competitive ratio less than* $2N$*, even against a non-abusive adversary.*

Proof. After an initial step, a non-abusive adversary Adv constructs a sequence consisting of phases such that in each phase it whacks at least $2N$ times as many moles as an online algorithm Alg does. Each phase starts at a time t when the adversary arrives in a hole. We denote by t' the latest deadline of the moles that are in this hole at time t. Note that $t \leq t' < t+1$, since the popup duration is 1. There are two possible positions for Alg to be at time t:

Case (a): Alg is in a vertex point different from the position of Adv;
Case (b): Alg is on an edge.

Moreover, if there are at the beginning of the phase some pending requests released before time t then Alg cannot reach any of them.

In Case (a), two moles are released at time t in holes where neither Alg nor Adv are. If Alg does not immediately go to one of these moles, it cannot whack any of them, whereas the adversary catches one of these moles. Otherwise, at time $\bar{t} = \max\{t', t+1/2\}$ the adversary releases N moles in his current position and N moles in a hole v that is not incident to the edge on which Alg is. Thus, Alg cannot whack any of them. Hence, it whacks at most one mole, whereas Adv reaches $2N$ moles by remaining in his position until time $\bar{t}$ and then moving to v.

In Case (b), Alg is in the interior of an edge and thus, it cannot reach any vertex point which is not incident to this edge by time $t + 1$. The adversary releases one mole in a free hole, i.e., a vertex point where no mole is and which is not incident to the edge on which Alg is. Hence, Alg does not whack any mole, and Adv hits one mole.

An initial sequence consisting of two requests in two different holes each releasing a single mole ensures that we end up either in Case (a) or Case (b). This completes the proof. $\square$

Note that in the proof of the above lower bound we use the fact that release dates may be non-integral. As we will see in the next section, this restriction is essential, because for integral release dates we are able to provide a 2-competitive algorithm.

For the sake of completeness we conclude this section of lower bounds with a brief consideration of randomized algorithms. We can easily extend the deterministic lower bound in Theorem 5.10 to a lower bound for randomized algorithms by blowing up the number of possible holes. Instead of releasing at each time t a mole in each of two free holes, we release moles in $k \geq 2$ free holes. We argue that for at least one of these holes the probability that the online server is in this hole at time $t + T$ is at most $1/k$. The $(k + 1)$st mole at time $t + T$ is released in this hole. Then, the expected number of moles caught by an online algorithm is from time $t = T$ onwards $\frac{1}{k} \cdot 2 + \frac{k-1}{k} \cdot 1 = \frac{k+1}{k}$, whereas the optimal offline algorithm can catch 2 moles.

Corollary 5.12. *Each randomized algorithm has a competitive ratio of at least* 2 *on the uniform metric space.*

5.5.2 Upper Bounds – How Well We Can Whack

In this section we analyze simple algorithms for Wham and give performance guarantees for the online problem on a uniform metric space.

Algorithm 18: First Come First Killed (Fcfk)

```
At any time $t$, move to a hole which contains a request with earliest
release date, breaking ties in favor of the point where the most moles
are above ground. If none of the moles that are above ground can be
reached by the algorithm, then the whacker does not move.
```

Theorem 5.13. *Algorithm* Fcfk *is* $2N$*-competitive for* $\text{Wham}_{T=1,N}$ *on the uniform metric space when all moles have unit popup duration.*

Proof. Partition the input sequence into maximal subsequences, such that each subsequence consists of requests that are released while Fcfk is serving continously, i.e., it is constantly moving between holes. We show that an optimal offline algorithm Opt whacks at most $2N$ times as many moles as Fcfk does for each subsequence.

Consider such a subsequence σ'. We denote by C_j^{Alg} the time where algorithm Alg whacks request r_j. If r_j is not caught, then we set $C_j^{\text{Alg}} = \infty$. Define the time at which Opt whacks its last mole of σ' as

$$t_{\max} = \max\{\, C_j^{\text{Opt}} : r_j \in \sigma' \text{ and } C_j^{\text{Opt}} < \infty \,\},$$

where we consider an optimum Opt that whacks its last mole as early as possible. Moreover, we define $t_{\min}$ such that the value $(t_{\max} - t_{\min})$ is integral and $\min_{j\in\sigma'} t_j \leq t_{\min} < \min_{j\in\sigma'} t_j + 1$. In each interval $(t, t+1]$ for $t = t_{\min}, \ldots, t_{\max} - 1$, Fcfk hits at least one mole and Opt cannot whack more than $2N$ moles. It remains to show that the moles which are reached by Opt before $t_{\min}$ can be compensated for by Fcfk.

If Fcfk whacks its last mole of σ' no later than time $t_{\max}$, then Opt catches at most N moles in the interval $(t_{\max} - 1, t_{\max}]$ since no new request can be released at $t_{\max}$ due to the maximality of the subsequence σ'. Moreover, Opt can whack at most N moles in the interval $(\min_{j\in\sigma'} t_j, t_{\min}]$. Therefore, the number of moles reached by Opt during the period before $t_{\min}$ can be accounted for by the moles caught in the last interval by Opt and thus, sum up to at most $2N$.

On the other hand, if Fcfk still whacks a mole from σ' after time $t_{\max}$, the number of moles caught by Opt during the first period is at most N times the number of moles hit by Fcfk after $t_{\max}$. □

The following lemma shows that the competitive ratio of $2N$ is tight for Fcfk and integral popup duration, even if one considers the more restricted adversary.

Lemma 5.14. *Let* $T \geq 1$ *be an integer.* Fcfk *has a competitive ratio at least* $2N$ *for* $\text{Wham}_{T,N}$ *on the uniform metric space against a non-abusive adversary.*

Proof. At time $t = 0$, the adversary releases T requests in holes $1, \ldots, T$, each of them with weight 1. At time $t = 1/2$, in hole $2T + 1$, a request is released with N moles. At time t, for $t = 1, \ldots, T - 1$, one request in

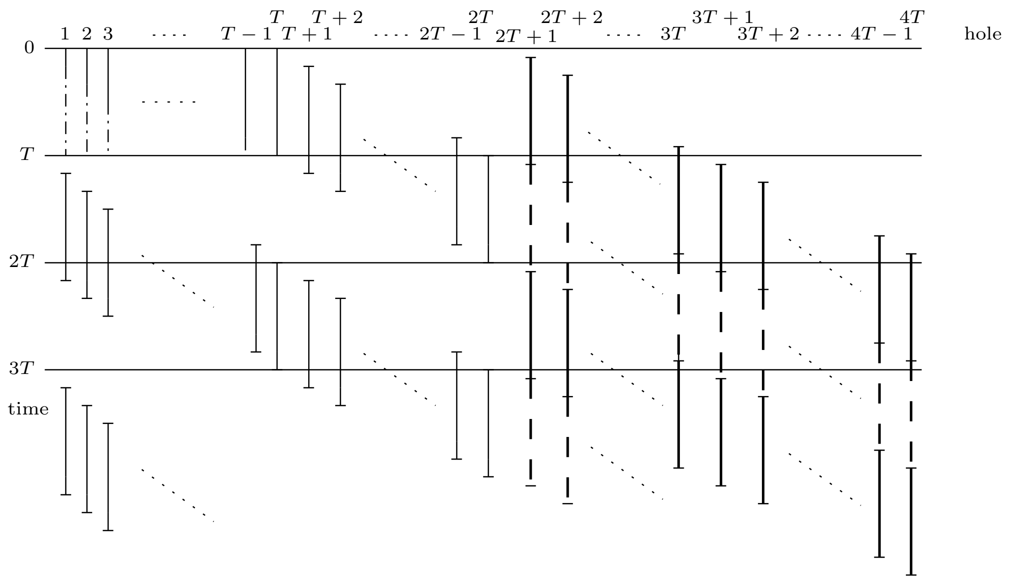


Figure 5.2: Lower bound sequence for Fcfk. Each request is represented by a vertical line between the release date and deadline. Thick lines illustrate requests with N moles, the thin lines depict requests with single moles. The line segment is dashed after the request has been served by an adversary Adv, and dash-dotted after being served by Fcfk and Adv. Notice that from time T onwards, Fcfk serves all its requests by their deadlines.

hole $T+t$ is given with one mole and at time $t+1/2$ a request with N moles is given in $2T+1+t$. At time t, for $t = T, T+1, \ldots$, one mole is popping up in hole $1+(T+t-1) \mod 2T$. And at time $t+1/2$ two requests are given, each with N moles: one in $2T+1+(t \mod T)$ and one in $3T+1+(t \mod T)$. This sequence is visualized in Figure 5.2.

Up to time T, Fcfk whacks the moles released at time 0. After time T it moves to the hole with the earliest released request that it can reach. As the requests with N moles are released $1/2$ time unit later than the requests with a single mole, Fcfk is not able to whack any of the higher weighted requests. Hence, it catches one mole in each unit length time interval. In each unit length interval from time $T+1/2$ onwards, there is one hole where a request with N moles has its deadline and a new request with N moles is released. Hence, the adversary Adv can whack $2N$ moles in every unit length time interval after time T. $\square$

Recall that no deterministic online algorithm can be better than 2-competitive (Theorem 5.10). Hence, by Theorem 5.13 we know that Fcfk achieves the best-possible competitive ratio in the case of a mole-per-hole limit of N=1. For general N but $T = 1$, Fcfk is also best-possible by Theorem 5.11. For the special case of integral release dates and $T = 1$, Fcfk obtains a competitive ratio of 2 even for general N.

Theorem 5.15. *If all release times are integral, then* FCFK *is* 2*-competitive for* $\text{WHAM}_{T=1,N}$ *on the uniform metric space.*

Proof. Due to the integral release dates, both, the optimal offline algorithm OPT and FCFK, are in holes at integral points in time. Moreover, OPT serves at most two requests released at the same time because of the unit popup duration. FCFK on the other hand, whacks at least the moles of one request released at a certain time and by definition it chooses the request with the highest number of moles. Therefore, it reaches at least half of the moles whacked by OPT. □

Obviously, FCFK's flaw lies in ignoring all requests with a later deadline even though they could contribute with a higher weight to the objective value. In order to overcome this drawback we consider another algorithm which we call *Ignore and Whack the Most* (IWTM). In this algorithm, we divide the time horizon into intervals of length $\ell = \lfloor \frac{T}{2} \rfloor$, and we denote these intervals by $I_i = [(i-1)\ell, i\ell)$, for $i = 0, 1, 2, \ldots, L$, where I_L is the last interval in which moles are whacked. We say that at time t, the *current interval* is the interval I_i for which $t \in I_i$. Note that these intervals only have a positive length for $T \geq 2$.

When formulating the algorithm IWTM we allow the algorithm to whack only a subset of the moles available at a certain hole. Although our problem definition would force all moles at v to be whacked, this condition can be enforced within the algorithm by keeping a "virtual scenario".

Algorithm 19: Ignore and Whack the Most (IWTM)

At any time when the whacker is in a hole, it moves to the hole with the highest number of pending moles released in the previous interval. Only those moles will be whacked.

Theorem 5.16. *The algorithm* IWTM *is* ρ*-competitive for the online problem* $\text{WHAM}_{T,N}$ *on the uniform metric space where*

$$\rho = \frac{\lceil \lfloor \frac{T}{2} \rfloor + T \rceil}{\lfloor \frac{T}{2} \rfloor}, \qquad \text{for popup duration } T \geq 2\,.$$

Proof. Let k_i, for $i = 1, 2, \ldots, L$, denote the number of moles released in interval I_i, whacked by OPT, and let h_i denote the number of moles whacked by IWTM during interval I_i. Then

$$\text{OPT}(\sigma) = \sum_i k_i, \qquad \text{and} \qquad \text{IWTM}(\sigma) = \sum_i h_i\,. \tag{5.1}$$

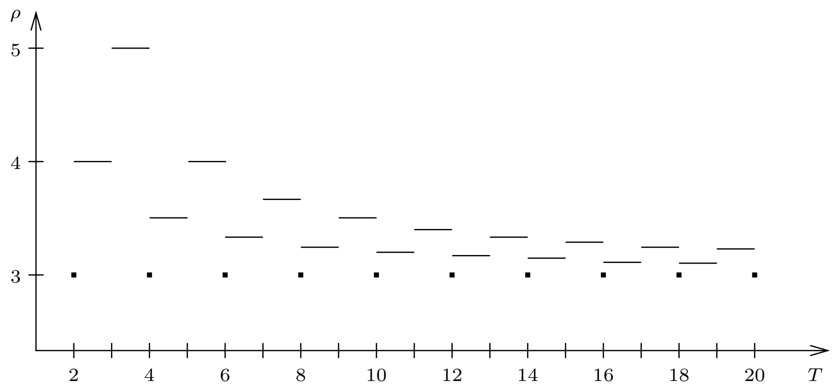


Figure 5.3: Competitive ratio for IWTM for popup duration $T \in [2, 20]$.

Moreover, since no moles are released in the last interval I_L, it follows that $k_L = 0$.

First note that at integral time points IWTM is always in a hole. Therefore, during interval I_{i+1} it can visit ℓ holes. If it visits fewer than ℓ holes, then the number of requests released in interval I_i is less than ℓ. Hence, OPT cannot reach more than h_{i+1} moles of those released in I_i.

Conversely, suppose that IWTM visits exactly ℓ holes during interval I_{i+1}. The optimum can visit at most $\lceil \ell + T \rceil$ holes of requests released in interval I_i. By definition IWTM serves the ℓ holes with the highest weight of pending requests released in I_i. Therefore, $h_{i+1} \geq (\ell / \lceil \ell + T \rceil) k_i$. Hence, by Equations (5.1), we know that

$$\text{IWTM}(\sigma) \geq \frac{\ell}{\lceil \ell + T \rceil} \text{OPT}(\sigma)\,.$$

Recall that $\ell = \lfloor T/2 \rfloor$. □

For popup duration, T, ranging from 2 to 20, the values of the competitive ratio of IWTM are depicted in Figure 5.3.

5.6 How to Whack with Multiple Servers

In the previous sections we investigated WHAM with one server (whacker). In this section, we discuss how the results extend to the case of multiple servers; we denote the generalized problem by *k-server* WHAM.

For the case of the line it is easy to adopt the lower bound result in Theorem 5.3 and prove that no deterministic algorithm can achieve a constant competitive ratio for k-server WHAM on the line and on the half line, $\mathbb{R}^+$, against an all-knowing adversary.

Consider now Wham on the uniform metric space. We examine an immediate extension of Iwtm, as presented in the previous Section, to k servers which we call Iwtm$_k$.

Algorithm 20: Iwtm$_k$

At any time when a whacker is in a hole, it moves to the hole with the highest number of pending moles released in the previous interval and which no other server is going to whack. Only those moles will be whacked.

In the following we show that the performance guarantee for Iwtm in Theorem 5.16 holds also for its k-server version. To that end, consider a sequence σ for k-server Wham$_{T,N}$. Let $\sigma_1, \ldots, \sigma_k$ be disjoint subsequences of σ, such that σ_i contains all requests served by the ith server in the *optimal* solution. We claim that Iwtm$_k$ reaches on sequence σ at least as many moles as (the single server) Iwtm whacks in total on all sequences $\sigma_1, \ldots, \sigma_k$.

Lemma 5.17. *Consider a request sequence σ of the problem k-server* Wham. *Let* Iwtm$_k(\sigma)$ *denote the number of requests that* Iwtm$_k$ *serves on σ, whereas* Iwtm(σ_i) *denotes the number of requests reached by* Iwtm *with a single server on subsequence σ_i, for $i = 1, \ldots, k$ as defined above. Then,*

$$\textsc{Iwtm}_k(\sigma) \;\geq\; \sum_{i=1}^{k} \textsc{Iwtm}(\sigma_i)\,.$$

Proof. Consider all requests released during a time interval I_h in all subsequences $\sigma_1, \ldots, \sigma_k$. All these requests are considered by Iwtm$_k$ during interval I_{h+1}, and can be served by Iwtm$_k$. As Iwtm$_k$ chooses to move to those requests with highest weight, it will whack at least as many moles released in I_h as the sum of the Iwtm(σ_i). □

Theorem 5.18. *The algorithm* Iwtm$_k$ *is ρ-competitive for the k-server version of the problem* Wham$_{T,N}$ *on the uniform metric space with*

$$\rho \;=\; \frac{\left\lceil \lfloor \frac{T}{2} \rfloor \;+\; T \right\rceil}{\lfloor \frac{T}{2} \rfloor}\,.$$

Proof. By definition of subsequences $\sigma_1, \ldots, \sigma_k$, an optimal algorithm, Opt, reaches on the request sequence σ exactly the sum of all requests reached by the the optimal single servers on the subsequences, Opt$_1(\sigma_i)$, that is,

$$\textsc{Opt}(\sigma) \;=\; \sum_{i=1}^{k} \textsc{Opt}_1(\sigma_i)\,.$$

By Theorem 5.16, we know that the single-server algorithm Iwtm is ρ-competitive. Combined with Lemma 5.17 this concludes the proof:

$$\begin{aligned}\sum_{i=1}^{k} \textsc{Opt}_1(\sigma_i) &\leq \sum_{i=1}^{k} \rho \cdot \textsc{Iwtm}(\sigma_i) \\ &\leq \rho \cdot \textsc{Iwtm}_k(\sigma)\,.\end{aligned}$$

□

Bibliography

[ABC$^+$99] 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*, Proceedings of the 40th IEEE Symposium on the Foundations of Computer Science (New York, NY, USA), 1999, pp. 32–43.

[AE02] Yossi Azar and Leah Epstein, *On-line scheduling with precedence constraints*, Discrete Applied Mathematics **119** (2002), 169–180.

[AFL$^+$01] Giorgio Ausiello, Esteban Feuerstein, Stefano Leonardi, Leen Stougie, and Maurizio Talamo, *Algorithms for the on-line traveling salesman*, Algorithmica **29** (2001), no. 4, 560–581.

[AGA99] Ali Allahverdi, Jatinder N. D. Gupta, and Tariq Aldowaisan, *A review of scheduling research involving setup considerations*, Omega **27** (1999), no. 2, 219–239.

[AKR00] Norbert Ascheuer, Sven O. Krumke, and Jörg Rambau, *Online dial-a-ride problems: Minimizing the completion time*, Proceedings of the 17th International Symposium on Theoretical Aspects of Computer Science (Lille, France), Lecture Notes in Computer Science, vol. 1770, Springer, 2000, pp. 639–650.

[Alb98] Susanne Albers, *A competitive analysis of the list update problem with lookahead*, Theoretical Computer Science **197** (1998), 95–109.

[AM06] Christoph Ambühl and Monaldo Mastrolilli, *Single machine precedence constrained scheduling is a vertex cover problem*, Proceedings of 14th European Symposium on Algorithms (Zurich, Switzerland) (Yossi Azar and Thomas Erlebach, eds.), Lecture Notes in Computer Science, no. 4168, Springer, 2006, pp. 28–39.

[AP04] Edward J. Anderson and Chris N. Potts, *On-line scheduling of a single machine to minimize total weighted completion time*, Mathematics of Operations Research **29** (2004), 686–697.

[BBCM04] Nikhil Bansal, Avrim Blum, Shuchi Chawla, and Adam Meyerson, *Approximation algorithms for deadline-TSP and vehicle routing with time-windows*, Proceedings of the 36th ACM Symposium on the Theory of Computing (Chicago, IL, USA), 2004, pp. 166–174.

[BDBK$^+$94] Shai Ben-David, Allan Borodin, Richard M. Karp, Gábor Tardos, and Avi Wigderson, *On the power of randomization in on-line algorithms*, Algorithmica **11** (1994), 2–14.

[BDF81] John L. Bruno, Peter J. Downey, and Greg N. Frederickson, *Sequencing tasks with exponential service times to minimize the expected flowtime or makespan*, Journal of the ACM **28** (1981), 100–113.

[Bea55] E.M.L. Beale, *On minimizing a convex function subject to linear inequalities*, Journal of the Royal Statistical Society. Series B. Methodological **17** (1955), 173–184; discussion, 194–203.

[BEY98] Allan Borodin and Ran El-Yaniv, *Online computation and competitive analysis*, Cambridge University Press, 1998.

[BFK$^+$04] Nikhil Bansal, Lisa K. Fleischer, Tracy Kimbrel, Mohammad Mahdian, Baruch Schieber, and Maxim Sviridenko, *Further improvements in competitive guarantees for qos buffering*, Proceedings of the 31st International Colloquium on Automata, Languages and Programming (Turku, Finland), Lecture Notes in Computer Science, Springer, 2004, pp. 196–207.

[BHS01] Sanjoy K. Baruah, Jayant R. Haritsa, and Nitin Sharma, *On-line scheduling to maximize task completions*, The Journal of Combinatorial Mathematics and Combinatorial Computing **39** (2001), 65–78.

[BL97] John R. Birge and François Louveaux, *Introduction to stochastic programming*, Springer Series in Operations Research, Springer, 1997.

[BLMS$^+$06] Luca Becchetti, Stefano Leonardi, Alberto Marchetti-Spaccamela, Guido Schäfer, and Tjark Vredeveld, *Average-case and smoothed competitive analysis of the multilevel feedback algorithm*, Mathematics of Operations Research **31** (2006), no. 1, 85–108.

[BLS92] Allan Borodin, Nathan Linial, and Michael E. Saks, *An optimal on-line algorithm for metrical task system*, Journal of the ACM **39** (1992), no. 4, 745–763.

[Bru04] Peter Brucker, *Scheduling algorithms*, 4th ed., Springer, 2004.

[CG85] Edward G. Coffman Jr. and Edgar N. Gilbert, *On the expected relative performance of list scheduling*, Operations Research **33** (1985), 548–561.

[CH99] Fabián A. Chudak and Dorit S. Hochbaum, *A half-integral linear programming relaxation for scheduling precedence-constrained jobs on a single machine.*, Operations Research Letters **25** (1999), no. 5, 199–204.

[CHW89] Edward G. Coffman Jr., Micha Hofri, and Gideon Weiss, *Scheduling stochastic jobs with a two point distribution on two parallel machines*, Probability in the Engineering and Informational Sciences **3** (1989), 89–116.

[CKW68] D. Chazan, Allan G. Konheim, and B. Weiss, *A note on time sharing*, Journal of Combinatorial Theory **5** (1968), 344–369.

[CLQSL06] Cheng-Feng Mabel Chou, Hui Liu, Maurice Queyranne, and David Simchi-Levi, *On the asymptotic optimality of a simple on-line algorithm for the stochastic single machine weighted completion time problem and its extensions*, Operations Research **54** (2006), no. 3, 464–474.

[CM99] Chandra Chekuri and Rajeev Motwani, *Precedence constrained scheduling to minimize sum of weighted completion times on a single machine.*, Discrete Applied Mathematics **98** (1999), no. 1–2, 29–38.

[CMNS01] Chandra Chekuri, Rajeev Motwani, B. Natarajan, and Clifford Stein, *Approximation techniques for average completion time scheduling*, SIAM Journal on Computing **31** (2001), 146–166.

[CQSL06] Cheng-Feng Mabel Chou, Maurice Queyranne, and David Simchi-Levi, *The asymptotic performance ratio of an on-line algorithm for uniform parallel machine scheduling with release dates*, Mathematical Programming **106** (2006), no. 1, 137–157.

[CS05] José R. Correa and Andreas S. Schulz, *Single-machine scheduling with precedence constraints*, Mathematics of Operations Research **30** (2005), no. 4, 1005–1021.

[CS06] Gang Chen and Zuo-Jun Max Shen, *Probabilistic asymptotic analysis on stochastic online scheduling problems*, To appear in IIE Transactions, 2006.

[CW05] José Correa and Michael Wagner, *LP-based online scheduling: From single to parallel machines*, Mathematical Programming Society Conference on Integer Programming and Combinatorial Optimization (Berlin, Germany) (Michael Jünger and Volker Kaibel, eds.), Lecture Notes in Computer Science, vol. 3509, Springer, 2005, pp. 196–209.

[Dan55] George B. Dantzig, *Linear programming under uncertainty*, Management Science **1** (1955), 197–206.

[Dea05] Brian C. Dean, *Approximation algorithms for stochastic scheduling problems*, Ph.D. thesis, Massachusetts Institute of Technology, 2005.

[DST03] Shane Dye, Leen Stougie, and Asgeir Tomasgard, *The stochastic single resource service-provision problem.*, Naval Research Logistics **50** (2003), no. 8, 869–887.

[EEI64] W. L. Eastman, Shimon Even, and I. M. Isaacs, *Bounds for the optimal scheduling of n jobs on m processors*, Management Science **11** (1964), 268–279.

[EKM04] Thomas Erlebach, Vanessa Kääb, and Rolf H. Möhring, *Scheduling AND/OR-networks on identical parallel machines*, Proceedings of the First International Workshop on Approximation and Online Algorithms, WAOA 2003 (Budapest, Hungary) (Klaus Jansen and Roberto Solis-Oba, eds.), Lecture Notes in Computer Science, vol. 2909, Springer, 2004, pp. 123–136.

[EvS03] Leah Epstein and Rob van Stee, *Lower bounds for on-line single-machine scheduling*, Theoretical Computer Science **299** (2003), 439–450.

[FKST98] Anja Feldmann, Ming-Yang Kao, Jiří Sgall, and Shang-Hua Teng, *Optimal online scheduling of parallel jobs with dependencies*, Journal of Combinatorial Optimization **1** (1998), no. 4, 393–411.

[FS01] Esteban Feuerstein and Leen Stougie, *On-line single server dial-a-ride problems*, Theoretical Computer Science **268** (2001), no. 1, 91–105.

[FW98a] Amos Fiat and Gerhard J. Woeginger, *Competitive odds and ends*, Online Algorithms: The State of the Art (Amos Fiat and Gerhard J. Woeginger, eds.), Lecture Notes in Computer Science, vol. 1442, Springer, Berlin, 1998, pp. 385–394.

[FW98b] ______, *Online algorithms: The state of the art*, Lecture Notes in Computer Science, vol. 1442, Springer, Berlin, 1998.

[FW99] ______, *On-line scheduling on a single machine: Minimizing the total completion time*, Acta Informatica **36** (1999), 287–293.

[Gar85] Robert S. Garfinkel, *Motivation and modeling*, The traveling salesman problem: a guided tour of combinatorial optimization (Eugene L. Lawler, Jan Karel Lenstra, Alexander H. G. Rinnooy Kan, and David B. Shmoys, eds.), John Wiley & Sons Ltd., 1985, pp. 17–36.

[GI99] Ashish Goel and Piotr Indyk, *Stochastic load balancing and related problems*, Proceedings of the 40th IEEE Symposium on the Foundations of Computer Science (New York, NY, USA), 1999, pp. 579–586.

[Git79] John C. Gittins, *Bandit processes and dynamic allocation indices*, Journal of the Royal Statistical Society, Series B **41** (1979), 148–177.

[Git89] ______, *Multi-armed bandit allocation indices*, Wiley, New York, 1989.

[GJ79] Michael R. Garey and David S. Johnson, *Computers and intractability: A guide to the theory of $\mathcal{NP}$-completeness*, W.H. Freeman and Company, New York, 1979.

[GKMV06] Sandra Gutiérrez, Sven O. Krumke, Nicole Megow, and Tjark Vredeveld, *How to whack moles*, Theoretical Computer Science **361** (2006), 329–341.

[GKR01] Martin Grötschel, Sven O. Krumke, and Jörg Rambau, *Online optimization of large scale systems*, Springer, Berlin Heidelberg New York, 2001.

[GLLR79] Ronald L. Graham, Eugene L. Lawler, Jan Karel Lenstra, and Alexander H. G. Rinnooy Kan, *Optimization and approximation in deterministic sequencing and scheduling: A survey*, Annals of Discrete Mathematics **5** (1979), 287–326.

[Goe96] Michel X. Goemans, *A supermodular relaxation for scheduling with release dates*, Proceedings of the 5th Mathematical Programming Society Conference on Integer Programming and Combinatorial Optimization (Vancouver, BC, Canada) (William H. Cunningham, S. Thomas McCormick, and Maurice Queyranne, eds.), Lecture Notes in Computer Science, vol. 1084, Springer, 1996, pp. 288–300.

[GPRS04] Anupam Gupta, Martin Pál, R. Ravi, and Amitabh Sinha, *Boosted sampling: approximation algorithms for stochastic optimization.*, Proceedings of the 36th ACM Symposium on the Theory of Computing (Chicago, IL, USA), 2004, pp. 417–426.

[GQS$^+$02] Michel X. Goemans, Maurice Queyranne, Andreas S. Schulz, Martin Skutella, and Yaoguang Wang, *Single machine scheduling with release dates*, SIAM Journal on Discrete Mathematics **15** (2002), 165–192.

[Gra69] Ronald L. Graham, *Bounds on multiprocessor timing anomalies*, SIAM Journal on Applied Mathematics **17** (1969), no. 2, 416–429.

[GW00] Michel X. Goemans and David P. Williamson, *Two-dimensional Gantt charts and a scheduling algorithm of Lawler*, SIAM Journal on Discrete Mathematics **13** (2000), no. 3, 281–294.

[HKR00] Dietrich Hauptmeier, Sven O. Krumke, and Jörg Rambau, *The online dial-a-ride problem under reasonable load*, Proceedings of the 4th Italian Conference on Algorithms and Complexity (Rome, Italy) (Gian Carlo Bongiovanni, Giorgio Gambosi, and Rossella Petreschi, eds.), Lecture Notes in Computer Science, vol. 1767, Springer, 2000, pp. 125–136.

[HSSW97] 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*, Mathematics of Operations Research **22** (1997), 513–544.

[HV96] Han Hoogeveen and Arjen P. A. Vestjens, *Optimal on-line algorithms for single-machine scheduling*, Proceedings of the 5th Mathematical Programming Society Conference on Integer Programming and Combinatorial Optimization (Vancouver, BC, Canada) (William H. Cunningham, S. Thomas McCormick, and Maurice Queyranne, eds.), Lecture Notes in Computer Science, vol. 1084, Springer, 1996, pp. 404–414.

[HW81] W. J. Hall and J. A. Wellner, *Mean residual life*, Proceedings of the International Symposium on Statistics and Related Topics (Ottawa, ON, Canada) (M. Csörgö, D. A. Dawson, J. N. K. Rao, and A. K. Md. E. Saleh, eds.), 1981, pp. 169–184.

[IKMM04] Nicole Immorlica, David R. Karger, Maria Minkoff, and Vahab S. Mirrokni, *On the costs and benefits of procrastination: approximation algorithms for stochastic combinatorial optimization problems*, Proceedings of the 15th ACM-SIAM Symposium on Discrete Algorithms (New Orleans, LA, USA), 2004, pp. 691–700.

[ILR04] Sandy Irani, Xiangwen Lu, and Amelia Regan, *On-line algorithms for the dynamic traveling repair problem*, Journal of Scheduling **7** (2004), no. 3, 243–258.

[Käm89] Thomas Kämpke, *Optimal scheduling of jobs with exponential service times on identical parallel processors.*, Operations Research **37** (1989), no. 1, 126–133.

[Kar72] Richard M. Karp, *Reducibility among combinatorial problems*, Complexity of Computer Computations (R. E. Miller and J. W. Thatcher, eds.), Plenum Press, New York, 1972, pp. 85–103.

[KdPPS03] Sven O. Krumke, Willem de Paepe, Diana Poensgen, and Leen Stougie, *News from the online traveling repairman*, Theoretical Computer Science **295** (2003), 279–294.

[KK86] Tsuyoshi Kawaguchi and Seiki Kyan, *Worst case bound of an LRF schedule for the mean weighted flow-time problem*, SIAM Journal on Computing **15** (1986), 1119–1129.

[KLL$^+$02] Sven O. Krumke, Luigi Laura, Maarten Lipmann, Alberto Marchetti-Spaccamela, Willem de Paepe, Diana Poensgen, and Leen Stougie, *Non-abusiveness helps: An $O(1)$-competitive algorithm for minimizing the maximum flow time in the online traveling salesman problem*, Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization (Rome, Italy) (Klaus Jansen, Stefano Leonardi, and Vijay V. Vazirani, eds.), Lecture Notes in Computer Science, vol. 2462, Springer, 2002, pp. 200–214.

[KLM$^+$04] Alexander Kesselman, Zvi Lotker, Yishay Mansour, Boaz Patt-Shamir, Baruch Schieber, and Maxim Sviridenko, *Buffer overflow management in QoS Switches*, SIAM Journal on Computing **33** (2004), no. 3, 563–583.

[KMRS88] Anna R. Karlin, Mark S. Manasse, Larry Rudolph, and Daniel Dominic Sleator, *Competitive snoopy paging*, Algorithmica **3** (1988), 70–119.

[KMV04] Sven O. Krumke, Nicole Megow, and Tjark Vredeveld, *How to whack moles*, Processdings of the First International Workshop on Approximation and Online Algorithms, WAOA 2003 (Budapest, Hungary) (Klaus Jansen and Roberto Solis-Oba, eds.), Lecture Notes in Computer Science, no. 2909, Springer, 2004.

[KMvS05] Alexander Kesselman, Yishay Mansour, and Rob van Stee, *Improved competitive guarantees for qos buffering*, Algorithmica **43** (2005), no. (1–2), 63–80.

[Kon68] Allan G. Konheim, *A note on time sharing with preferred customers*, Probability Theory and Related Fields **9** (1968), 112–130.

[KP00a] Bala Kalyanasundaram and Kirk R. Pruhs, *Speed is as powerful as clairvoyance*, Journal of the ACM **47** (2000), 617–643.

[KP00b] Elias Koutsoupias and Christos H. Papadimitriou, *Beyond competitive analysis*, SIAM Journal on Computing **30** (2000), 300–317.

[KP03] Bala Kalyanasundaram and Kirk R. Pruhs, *Maximizing job completions online*, Journal of Algorithms **49** (2003), no. 1, 63–85.

[Law78] Eugene L. Lawler, *Sequencing jobs to minimize total weighted completion time subject to precedence constraints*, Annals of Discrete Mathematics **2** (1978), 75–90.

[Leu04] Joseph Y-T. Leung, *Handbook of scheduling: Algorithms, models, and performance analysis*, Chapman & Hall/CRC, 2004.

[LLKS85] Eugene L. Lawler, Jan Karel Lenstra, Alexander H.G. Rinnooy Kan, and David B. Shmoys (eds.), *The traveling salesman problem: A guided tour of combinatorial optimization*, Wiley Series in Discrete Mathematics & Optimization, Wiley, 1985.

[LLLR84] Jacques Labetoulle, Eugene L. Lawler, Jan Karel Lenstra, and Alexander H. G. Rinnooy Kan, *Preemptive scheduling of uniform machines subject to release dates*, Progress in Combinatorial Optimization (William R. Pulleyblank, ed.), Academic Press Canada, Waterloo, ON, Canada, 1984, pp. 245–261.

[LR78] Jan Karel Lenstra and Alexander H. G. Rinnooy Kan, *Complexity of scheduling under precedence constraints*, Operations Research **26** (1978), no. 1, 22–35.

[LRB77] Jan Karel Lenstra, Alexander H. G. Rinnooy Kan, and Peter Brucker, *Complexity of machine scheduling problems*, Annals of Discrete Mathematics **1** (1977), 243–362.

[LSS03] Xiwen Lu, René A. Sitters, and Leen Stougie, *A class of on-line scheduling algorithms to minimize total completion time*, Operations Research Letters **31** (2003), 232–236.

[MK94] Weizhen Mao and Rex K. Kincaid, *A look-ahead heuristic for scheduling jobs with release dates on a single machine*, Computers & Operations Research **21** (1994), no. 10, 1041–1050.

[Möh89] Rolf H. Möhring, *Computationally tractable classes of ordered sets*, Algorithms and Order (I. Rival, ed.), vol. 255, Kluwer Academic, 1989, pp. 105–193.

[MQW03] François Margot, Maurice Queyranne, and Yaoguang Wang, *Decompositions, network flows, and a precedence constrained single-machine scheduling problem*, Operations Research **51** (2003), no. 6, 981–992.

[MR95] Rajeev Motwani and Prabhakar Raghavan, *Randomized algorithms*, Cambridge University Press, 1995.

[MRW84] Rolf H. Möhring, Franz Josef Radermacher, and Gideon Weiss, *Stochastic scheduling problems I: General strategies*, Zeitschrift für Operations Research **28** (1984), 193–260.

[MRW85] Rolf H. Möhring, Franz Josef Radermacher, and Gideon Weiss, *Stochastic scheduling problems II: Set strategies*, Zeitschrift für Operations Research **29** (1985), no. 3, A65–A104.

[MSS03] Nicole Megow, René A. Sitters, and Leen Stougie, *Preemptive online scheduling to minimize the sum of weighted completion times*, Working paper, 2003.

[MST98] Rajeev Motwani, Vijay Saraswat, and Eric Torng, *Online scheduling with lookahead: Multipass assembly lines*, INFORMS Journal on Computing **10** (1998), no. 3, 331–340.

[MSU99] Rolf H. Möhring, Andreas S. Schulz, and Marc Uetz, *Approximation in stochastic scheduling: the power of LP-based priority policies*, Journal of the ACM **46** (1999), 924–942.

[MUV06] Nicole Megow, Marc Uetz, and Tjark Vredeveld, *Models and algorithms for stochastic online scheduling*, Mathematics of Operations Research **31** (2006), no. 3, 513–525.

[Pin83] Michael Pinedo, *Stochastic scheduling with release dates and due dates*, Operations Research **31** (1983), 559–572.

[Pin02] Michael Pinedo, *Scheduling: Theory, algorithms, and systems*, 2nd ed., Prentice Hall, Upper Saddle River, NJ, 2002.

[Pis03] Nicolai N. Pisaruk, *A fully combinatorial 2-approximation algorithm for precedence-constrained scheduling a single machine to minimize average weighted completion time.*, Discrete Applied Mathematics **131** (2003), no. 3, 655–663.

[PST04] Kirk R. Pruhs, Jiří Sgall, and Eric Torng, *Online scheduling*, Handbook of Scheduling: Algorithms, Models, and Performance Analysis (Joseph Y-T. Leung, ed.), Chapman & Hall/CRC, 2004.

[PSW98] Cynthia A. Phillips, Clifford Stein, and Joel Wein, *Minimizing average completion time in the presence of release dates*, Mathematical Programming **82** (1998), 199–223.

[Que93] Maurice Queyranne, *Structure of a simple scheduling polyhedron*, Mathematical Programming (Series A) **58** (1993), no. 2, 263–285.

[Ros03] Sheldon M. Ross, *Introduction to probability models*, 8th ed., Probability and Mathematical Statistics, Harcourt Academic Press, 2003.

[Rot66] Michael H. Rothkopf, *Scheduling with random service times*, Management Science **12** (1966), 703–713.

[RS04] R. Ravi and Amitabh Sinha, *Hedging uncertainty: Approximation algorithms for stochastic optimization problems*, Proceedings of the 10th Mathematical Programming Society Conference on Integer Programming and Combinatorial Optimization (New York, NY, USA), Lecture Notes in Computer Science, vol. 3064, 2004, pp. 101–115.

[Sch68] Linus Schrage, *A proof of the optimality of the shortest remaining processing time discipline*, Operations Research **16** (1968), 687–690.

[Sch96a] Andreas S. Schulz, *Polytopes and scheduling*, Dissertation, Technische Universität Berlin, 1996.

[Sch96b] ______, *Scheduling to minimize total weighted completion time: Performance guarantees of LP-based heuristics and lower bounds*, Proceedings of the 5th Mathematical Programming Society Conference on Integer Programming and Combinatorial Optimization (Vancouver, BC, Canada) (William H. Cunningham, S. Thomas McCormick, and Maurice Queyranne, eds.), Lecture Notes in Computer Science, vol. 1084, Springer, 1996, pp. 301–315.

[Sch05] ______, *New old algorithms for stochastic scheduling*, Algorithms for Optimization with Incomplete Information (Susanne Albers, Rolf H. Möhring, Georg Ch. Pflug, and Rüdiger Schultz, eds.), Dagstuhl Seminar Proceedings, no. 05031, 2005.

[Sei00] Steven S. Seiden, *A guessing game and randomized online algorithms*, Proceedings of the 32nd ACM Symposium on the Theory of Computing (Portland, OR, USA), 2000, pp. 592–601.

[Sev74] Kenneth C. Sevcik, *Scheduling for minimum total loss using service time distributions*, Journal of the ACM **21** (1974), 65–75.

[Sga98] Jiří Sgall, *On-line scheduling – a survey*, Online Algorithms: The State of the Art (Amos Fiat and Gerhard J. Woeginger, eds.), Lecture Notes in Computer Science, vol. 1442, Springer, Berlin, 1998, pp. 196–231.

[Sid75] Jeffrey B. Sidney, *Decomposition algorithms for single-machine sequencing with precedence relations and deferral costs*, Operations Research **23** (1975), 283–298.

[Sit04] René A. Sitters, *Complexity and approximation in routing and scheduling*, Ph.D. thesis, Technische Universiteit Eindhoven, 2004.

[Smi56] Wayne E. Smith, *Various optimizers for single-stage production*, Naval Research Logistics Quarterly **3** (1956), 59–66.

[SS02a] Andreas S. Schulz and Martin Skutella, *The power of α-points in preemptive single machine scheduling*, Journal of Scheduling **5** (2002), 121–133.

[SS02b] ______, *Scheduling unrelated machines by randomized rounding*, SIAM Journal on Discrete Mathematics **15** (2002), 450–469.

[SS04] David B. Shmoys and Chaitanya Swamy, *Stochastic optimization is (almost) as easy as deterministic optimization.*, Proceedings of the 45th IEEE Symposium on the Foundations of Computer Science (Rome, Italy), 2004, pp. 228–237.

[SS06a] Alexander Souza and Angelika Steger, *The expected competitive ratio for weighted completion time scheduling*, Theory of Computing Systems **39** (2006), 121–136.

[SS06b] Chaitanya Swamy and David B. Shmoys, *Approximation algorithms for 2-stage stochastic optimization problems*, SIGACT News **37** (2006), no. 1, 33–46.

[SSS06] Mark Scharbrodt, Thomas Schickinger, and Angelika Steger, *A new average case analysis for completion time scheduling*, Accepted for publication in Journal of the ACM, 2006.

[ST85] Daniel Dominic Sleator and Robert Endre Tarjan, *Amortized efficiency of list update and paging rules*, Communications of the ACM **28** (1985), 202–208.

[ST04] Daniel A. Spielman and Shang-Hua Teng, *Smoothed analysis: Why the simplex algorithm usually takes polynomial time*, Journal of the ACM **51** (2004), no. 3, 385–463.

[SU05] Martin Skutella and Marc Uetz, *Stochastic machine scheduling with precedence constraints*, SIAM Journal on Computing **34** (2005), no. 4, 788–802.

[SV97] Leen Stougie and Arjen P. A. Vestjens, *Randomized on-line scheduling: How low can't you go?*, Manuscript, 1997.

[Tsi92] John N. Tsitsiklis, *Special cases of traveling salesman and repairman problems with time windows*, Networks **22** (1992), 263–282.

[TV02] Paolo Toth and Daniele Vigo, *The vehicle routing problem*, Discrete Mathematics and Applications 9, SIAM, 2002.

[Uet02] Marc Uetz, *Algorithms for deterministic and stochastic scheduling*, Cuvillier Verlag, Göttingen, Germany, 2002.

[Uet03] Marc Uetz, *When greediness fails: examples from stochastic scheduling*, Operations Research Letters **31** (2003), no. 6, 413–419.

[Ves97] Arjen P. A. Vestjens, *On-line machine scheduling*, Ph.D. thesis, Eindhoven University of Technology, Netherlands, 1997.

[Web82] Richard R. Weber, *Scheduling jobs with stochastic processing requirements on parallel machines to minimize makespan or flow time*, Journal of Applied Probability **19** (1982), 167–182.

[Wei90] Gideon Weiss, *Approximation results in parallel machines stochastic scheduling*, Annals of Operations Research **26** (1990), 195–242.

[Wei92] ______, *Turnpike optimality of Smith's rule in parallel machines stochastic scheduling*, Mathematics of Operations Research **17** (1992), 255–270.

[Wei95] Gideon Weiss, *On almost optimal priority rules for preemptive scheduling of stochastic jobs on parallel machines*, Advances in Applied Probability **27** (1995), 827–845.

[Yao77] Andrew Chi-Chih Yao, *Probabilistic computations: toward a unified measure of complexity (extended abstract)*, Proceedings of the 18th IEEE Symposium on the Foundations of Computer Science, 1977, pp. 222–227.

[YL99] Wen-Hwa Yang and Ching-Jong Liao, *Survey of scheduling research involving setup times*, International Journal of System Science **30** (1999), no. 2, 143–155.

Zusammenfassung

Die Unvollständigkeit von Informationen ist ein allgegenwärtiges Problem beim Lösen von realistischen Optimierungsproblemen. Die Einschränkung betrifft typischerweise die Unsicherheit (Stochastik) in den Problemdaten oder das Fehlen von Informationen über Teile der Probleminstanz. Die vorliegende Arbeit beschäftigt sich mit dem Umgang mit unvollständiger Information beim Lösen von Schedulingproblemen. Als Scheduling bezeichnet man im Allgemeinen die zeitliche Zuordnung von Vorgängen auf Ressourcen mit beschränkter Kapazität.

Es gibt zwei Standardansätze für die Modellierung unvollständiger Information in Schedulingproblemen: das stochastische Schedulingmodell und das online Schedulingmodell. Beim stochastischen Scheduling wird davon ausgegangen, dass die Dauer eines jeden Vorgangs durch eine Zufallsvariable gegeben ist. Zu Beginn der Planung sind alle Vorgänge mit ihren problemspezifischen, deterministischen Daten und den Verteilungsfunktionen für die Vorgangsdauern bekannt. Erst während der Bearbeitung eines Vorgangs, gewinnt der Planer Informationen über die tatsächliche Realisierung der Dauer. Im online Scheduling dagegen, geht man davon aus, dass eine Probleminstanz Stück für Stück während des Planungsprozesses bekannt wird; das heißt, zu jedem Zeitpunkt müssen Entscheidungen getroffen werden, die lediglich auf den bisher bekannten Daten beruhen. Sobald ein Vorgang bekannt ist (release date), kennt ein Planer alle Vorgangsdaten, und diese sind deterministisch.

Jedes der Modelle findet seine Legitimation in den entsprechenden Anwendungsproblemen. Allerdings sind auch praktische Situationen vorstellbar, in denen beide Arten unvollständiger Information zutreffen. Darin begründet sich das Bedürfnis nach einem integrierten Modell.

Die vorliegende Arbeit untersucht Maschinen-Schedulingprobleme mit der Zielfunktion die Summe der (gewichteten) Fertigstellungszeiten der Vorgänge zu minimieren. Für die zwei traditionellen Modelle werden Algorithmen analysiert und Gütegarantien erzielt, die die derzeitig besten bekannten Resulte verbessern. Insbesondere werden erste Algorithmen für stochastisches Scheduling mit Vorgangsunterbrechungen vorgestellt, für die konstante Gütegarantien bewiesen werden. Das verallgemeinerte Modell, welches Unsicherheit in den Daten und das Fehlen von Informationen kombiniert, erfüllt den Anspruch, die beiden klassischen Modelle als Spezialfälle mitzu-

modellieren. Das heißt, Algorithmen in dem allgemeinen Modell lösen sowohl deterministische online Probleme als auch stochastische (offline) Probleme. Weiterhin wird die Güte eines Algorithmus in diesem Modell so definiert, dass sie sich direkt in einen Kompetitivitätsfaktor (online Scheduling) und in einen Approximationsfaktor (stochastisches Scheduling) übersetzen lässt.

Solch ein integriertes Modell erscheint relativ natürlich und ist von praktischem und theoretischem Interesse. Es ist jedoch völlig unklar, ob ein Algorithmus unter solchen informationstechnisch eingeschränkten Bedingungen Lösungen von beweisbar konstanter Güte finden kann. Die vorliegende Arbeit beantwortet diese Frage mit Algorithmen, die in dem verallgemeinerten Modell konstante Gütegarantien erreichen. Die Resultate sind nicht nur konkurrenzfähig zu den bisher besten bekannten Algorithmen für die Standardmodelle sondern sie gleichen ihnen bzw. verbessern sie sogar in einem Fall. Damit zeigt die vorliegende Arbeit für bestimmte Schedulingprobleme, dass Algorithmen in einem sehr allgemeinen Modell die gleiche bzw. keine viel schlechtere (derzeitig beweisbare) Güte erzielen als Algorithmen, die nur auf spezielle Probleme zugeschnitten sind.