Department of Mathematics
TUM School of Computation, Information and Technology
Technical University of Munich

Stochastic Online Scheduling on Parallel Machines

Theoretical Analysis and Computational Study on the Performance Guarantees of Deterministic and Randomized Algorithms

Marta Piperno

Thesis for the attainment of the academic degree

Master of Science

at the TUM School of Computation, Information and Technology of the Technical University of Munich

Supervisor:
Prof. Dr. rer. nat. Andreas S. Schulz

Advisors:
Prof. Dr. rer. nat. Andreas S. Schulz

Submitted:
Rome, April 25th, 2024

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

[illegible]

Rome, April 25th, 2024 Marta Piperno

Abstract

In this work, we address the stochastic online scheduling problem of minimizing the expected total weighted completion time on identical parallel machines. Building on Schulz's previous work, we finalize the proofs of the performance guarantees of the algorithms developed in [Sch08]. We demonstrate that the randomized and deterministic algorithms have performance ratios of $2+\Delta$ and $2.309+1.309\Delta$, respectively, where Δ is an upper bound on the squared coefficient of variation of the processing times. Notably, these guarantees are the best known ones for the problem with generic processing time distributions. Moreover, we conduct a computational study on simulated data to evaluate the performance of the analyzed algorithms in practice.

Contents

1 Introduction

Scheduling theory is a branch of Operational Research that has been an active field of study since the 1950s. It investigates the decision-making process of planning the times at which tasks will utilize provided resources in order to optimize a given objective. The generality of this problem renders scheduling theory relevant across a diverse spectrum of applications, ranging from compiler optimization [Che+96] to patient allocation in hospitals [SWW19].

Initial research in this field primarily focused on instances where all problem data is available to the decision-maker. Nonetheless, numerous real-world applications are subject to various degrees and types of uncertainty. In the hospital example, the arrival of future patients and their characteristics are unknown until check-in. Additionally, the time required to attend to such patients can vary based on external variables. These uncertainties are addressed using techniques developed in stochastic and online scheduling. In stochastic scheduling problems, which have been analyzed since the 1980s [MRW84], tasks' processing times are treated as random variables whose realizations are revealed only upon the task's completion. On the other hand, in online scheduling problems, the existence of a task is unknown until its arrival, at which point, its characteristics become known. This framework has received particular attention since the 1990s [FST94; GW93; SWW95]. Megow, Uetz, and Vredeveld incorporated these two models and considered stochastic online scheduling problems for the first time [MUV06]. In this setting, tasks arrive over time without prior notice, as in online scheduling. Moreover, at release time, similarly to stochastic scheduling, only the distribution of the job's processing time is revealed, and its actual realization becomes known upon completion.

Many scheduling problems are known to be strongly NP-hard [LKB77]. Consequently, research efforts have been directed towards developing efficient approximation algorithms. These algorithms aim to find near-optimal solutions in polynomial time, providing practical methods for real-world scheduling problems. In the stochastic online setting, we say that an algorithm is a ρ-approximation if the expected objective value of the solution computed by the algorithm is within a factor ρ of the expected optimal value of an offline scheduling policy that has access to the tasks' attributes, release times, and processing time distribution at time 0. Such factor ρ is referred to as the performance guarantee of the algorithm. Among the many techniques explored within the domain of approximation algorithms, list scheduling policies are particularly relevant in our work. List scheduling policies schedule jobs according to the order of a priority list. Examples of widely studied list scheduling policies are the "Weighted Shortest Processing Time" ($WSPT$) rule and its preemptive and online versions [GL11; JS18; KK86; Smi+56]. These rules prioritize jobs with higher weight-to-processing time ratios, where the weights are the jobs' attributes that represent their relative importance and influence the objective function. An additional technique that has been proven to be particularly successful in scheduling theory is the exploitation of linear programming relaxations that can be solved efficiently [Hal+97; MSU99]. The solutions of these relaxations are used to design algorithms for the original problem. Additionally, they provide a lower bound on the performance of the optimal solution, facilitating the evaluation of the algorithm's performance when the optimal solution is unknown. A widely used tool for converting the solutions of the relaxations into feasible schedules for the problem is the notion of α-points. These points indicate the first point in time in which a job has been processed for a fraction α of its processing time in the solution of the relaxation [CW05; Goe+02].

This thesis focuses on the stochastic online problem denoted as $P|r_j, p_j \sim stoch, online|E[\sum_{j\in N} w_j C_j]$ in Graham's notation [Gra+79]. In this problem, each job j is characterized by a non-negative weight w_j, release time r_j, and processing time distribution P_j. The jobs must be processed on identical parallel machines without preemption, with the objective of minimizing the expected total weighted completion time $E[\sum_{j\in N} w_j C_j]$. In this setting, Schulz developed deterministic and randomized approximation algorithms that run in polynomial time, which to this day have the best-known performance guarantees for the problem with generic processing time distributions [Sch08]. These algorithms employ a linear programming relaxation for the stochastic online problem, which can be interpreted as the relaxation of a single machine problem having deterministic processing times equivalent to the ratio between the expected processing times and the number of machines of the original problem. Remarkably, this relaxation can be solved by the online preemptive $WSPT$ [Goe+02]. This solution is then used to define policies for the original problem by assigning jobs in order of their α-points. Notably, these algorithms can be interpreted as a generalization to the stochastic setting of the ones described in [SS02].

The primary objective of this thesis is to finalize the proofs on the performance guarantee of the algorithms developed in the paper "Stochastic Online Scheduling Revisited" [Sch08]. We show that the randomized algorithm outlined in the paper has a performance guarantee of $2 + \Delta$, where Δ is an upper bound on the squared coefficient of variation of the processing times. Furthermore, using the method of conditional probability [Spe94], we demonstrate that this algorithm can be derandomized to an offline one with the same performance guarantee. Additionally, from the proof, we derive the explicit algorithm's structure. Schulz also devised a deterministic algorithm that works online. Improving the guarantee stated in the paper for $\Delta \leq 0.2361$, we show that it has a performance guarantee of $1.309\Delta + 2.309$. Moreover, when the algorithm is applied to the single machine problem, we establish a performance guarantee of $1.309\Delta + 1.618$, improving the state-of-the-art result for $\Delta \leq 0.764$. We complete our work with a computational study of the analyzed policies on simulated data. Given the limited literature on experimental studies concerning scheduling policies based on linear programming relaxations, this analysis provides valuable insights into their performance in practice.

This work is organized as follows. In Chapter 2, we formally introduce our problem along with the notation and essential tools required for the subsequent chapters. Additionally, we present a literature review on the state-of-the-art results for the problem under consideration and its offline and deterministic versions, as well as an overview of the techniques typically employed for addressing these problems. In Chapter 3, we first prove the validity of the relaxation proposed by Schulz in [Sch08], and we discuss its online solvability. Subsequently, we proceed to describe the algorithms and demonstrate their performance guarantees. In Chapter 4, we discuss the implementation procedure and present the results of our computational study. We conclude by indicating potential directions for future research inspired by the insights gained from our work.

2 Preliminaries

In scheduling problems, we are interested in the optimal way to process a set of n jobs $N = \{1, \dots, n\}$ on a set of machines $M = \{1, \dots, m\}$ to minimize a given objective function. More specifically, a schedule designates which job is processed on which machine at any time. Jobs are characterized by their weight $w_j \geq 0$, processing times $p_{j,i} \geq 0$ on each machine i, and release time $r_j \geq 0$, which is the earliest time in which the processing of job j can start. In the scheduling literature, the 3-file notation $\alpha|\beta|\gamma$ introduced by Graham et al. in [Gra+79] is used to categorize problems based on the characteristics of the jobs, machines, and objective function.

- The field α specifies the machine environment. For instance, $\alpha = 1$ refers to the problems with a single machine, i.e., when $m = 1$. Meanwhile, $\alpha = P$ is used for scenarios with identical parallel machines. In these two cases, jobs' processing times are not machine-dependent. Therefore, we have $p_j = p_{j,i}$ for any $i \in M$. $\alpha = Q$ denotes the problems with uniform machines. In this scenario, each machine i has a specific speed s_i, and the processing time of a job j on machine i can be expressed as $p_{j,i} = \frac{p_j}{s_i}$. Finally, $\alpha = R$ refers to the unrelated machine environment where machines have jobs-specific speeds $s_{j,i}$ and $p_{j,i} = \frac{p_j}{s_{j,i}}$. When a specific number of machines m is considered, we usually write Pm, Qm, and Rm.
- The field β specifies the characteristics of the jobs. r_j indicates that jobs can have non-trivial release times, i.e., $r_j > 0$. $pmtn$ is used when jobs can be preempted; that is, when the processing of any job can be interrupted and resumed later on. $prec$ denotes that there are precedence constraints between the jobs, i.e., before a job can start, its predecessor must be completed. In addition, we write $online$ when we consider an online problem and $p_j \sim stoch$ if jobs' processing times are stochastic. When the field is empty, the problem under consideration is the default one with non-preemptive deterministic jobs with trivial release times a no precedence constraints.
- The field γ denotes the objective function we are considering. It is usually a function of the jobs' completion times $C = (C_1, \dots, C_n)$. Commonly studied objective functions are the total weighted completion time $\sum_{j \in N} w_j C_j$ and the makespan $C_{max} = \max_{j \in N} C_j$.

We will focus on the stochastic online problem denoted by $P|r_j, p_j \sim stoch, online|E[\sum_{j \in N} w_j C_j]$. Unless stated otherwise, the rest of this work will consider the identical machine case and, therefore, processing times p_j independent of the machines.

2.1 Scheduling

In this section, we focus on the general definition and methods used for deterministic offline scheduling without precedence constraints. In the subsequent sections, we will update the definitions for the stochastic and online cases. We refer to [Pin22] for a detailed introduction to scheduling problems. For the problem $P|r_j| \sum_{j \in N} w_j C_j$ we consider an instance $I = \{r, p, w, m\}$ consisting in the number of machines m and the set of jobs N with their release times $r = (r_1, \dots, r_n)$, weights $w = (w_1, \dots, w_n)$ and processing times $p = (p_1, \dots, p_n)$. Various definitions of schedule exist in literature, however for our purposes, the definition employed in [BV21] is the most fitting.

Definition 2.1 (Schedule, Start and Completion Time). *A schedule S for an instance I is characterized by a set of left-continuous functions $\chi^S_{j,i} : [0, \infty) \to \{0, 1\}$, where $\chi^S_{j,i}(t) = 1$ indicates that job j is being processed on machine i at time t. A schedule must satisfy the following constraints:*

- *Job assignment:* $\chi_j^S(t) := \sum_{i=1}^m \chi_{j,i}^S(t) \leq 1 \quad \forall j \in N, \forall t \in [0, \infty)$.
- *Machine allocation:* $\sum_{j=1}^n \chi_{j,i}^S(t) \leq 1 \quad \forall i \in M, \forall t \in [0, \infty)$.
- *Release:* $\chi_{j,i}^S(t) = 0 \quad \forall t < r_j; \quad \forall i \in M, \forall j \in N$.
- *Completion:* $\int_0^\infty \sum_{i=1}^m \chi_{j,i}^S(t)\, dt = p_j \quad \forall j \in N$.
- *Non-migratory:* $\exists i \in M$ *s.t.* $\chi_{j,i}^S(t) = \chi_j^S(t) \quad \forall t \in [0, \infty)$.

The start and completion time of a job j in the schedule S for instance I are defined as $S_j^S(I) := \min support(\chi_j^S)$ and $C_j^S(I) := \max support(\chi_j^S)$, respectively.

The job assignment constraint ensures that each job is processed on no more than one machine at any time, while the machine allocation constraint guarantees that each machine processes at most one job simultaneously. The release constraint ensures that no job is started before its release time. The completion constraint enforces that each job is completely processed, while the non-migratory constraint enforces that each job can only be processed on one machine. Depending on the considered problem, additional constraints can be added. For instance, in non-preemptive scheduling, we require that the support of $\chi_j^s(t)$ is an interval. To simplify the notation, we will usually write C_j^S and S_j^S to refer to job j's completion and start time under the schedule S, dropping the explicit dependence on I in the notation. Assuming an objective function γ dependent on the completion times, we can define the optimal schedule as follows.

Definition 2.2 (Optimal Schedule)**.** *A schedule S^{OPT} is optimal for an instance I, within the schedule family σ, if it holds:*

$$OPT(I) := \gamma(C^{S^{OPT}}(I)) = \inf_{S \in \sigma} \gamma(C^S(I)). \tag{2.1}$$

Where $\gamma(C^S(I))$ is the objective value for instance I under schedule S.

Since the objective function γ is predetermined by the problem, in the remainder of this work, we simplify the notation by indicating with $S(I)$ the objective value of the schedule S for the instance I.
In the next sections, we will need the notion of available jobs and non-delay schedule.

Definition 2.3 (Available Job)**.** *A job j is said to be available at time t if $t \geq r_j$.*

Definition 2.4 (Non-Delay Schedule)**.** *A feasible schedule is said to be non-delay if no machine is kept idle while there is an unscheduled available job,*

As can be seen in Example 2.1, imposing a schedule to be non-delay could prevent us from finding the optimal solutions in the non-preemptive setting. Allowing forced idleness enables the machines to remain free for processing potentially more "important" jobs that are released later. In fact, the additional cost incurred due to deliberate idleness may be negligible when weighed against the advantage of starting a job with higher weights earlier. Schedules that incorporate job-delay strategies, such as considering modified release times, have garnered significant attention in the literature [Goe+02; MT18; MUV06; SS02].

Example 2.1. *Consider two jobs that have to be processed on one machine in order to minimize the total weighted sum of completion times. Jobs 1 and 2 have the following characteristics: $(r_1, p_1, w_1) = (0, 3, 5)$ and $(r_2, p_2, w_2) = (1, 1, 10)$. Scheduling the job 1 at time 0 results in an objective value of $3 \cdot 5 + 4 \cdot 10 = 55$. However, keeping the machine idle and processing Job 2 first results in a weighted completion time of $5 \cdot 5 + 2 \cdot 10 = 45$. Therefore, the non-delay schedule is not optimal in this scenario.*

2.1.1 List Scheduling Policies

In scheduling theory, particular attention has been devoted to the study of heuristics and list scheduling rules. In list scheduling, jobs are processed sequentially according to a priority list L. In Graham's list scheduling algorithm [Gra66], whenever a machine is idle, and there are unassigned available jobs in L, the first one among them is stated in the machines. If the list L remains the same throughout the scheduling process, we say that the list scheduling rule is static. Conversely, the list scheduling is dynamic. The latter is the case in online scheduling. In the deterministic scheduling framework, commonly considered priority lists in Graham's algorithm are the "longest/shortest processing time first" (LPT,SPT) and "weighted shortest processing time first" ($WSPT$) rules. The $WSPT$, also known as the Smith's ratio rule [Smi+56], schedules jobs in order of non-increasing ratios $\frac{w_j}{p_j}$. When all the job's weights equal 1, the $WSPT$ rule simplifies to the SPT. This $WSPT$ is particularly effective for the total weighted completion time objective, as it balances the urgency of high-weighted jobs with the efficiency of shorter tasks. In scenarios where jobs possess equal processing times, the $WSPT$ rule prioritizes jobs with higher weights. On the other hand, when jobs share identical weights, the rule favors those with the shortest processing times. Both cases ensure a lower total weighted completion time. The $WSPT$ rule and its online, stochastic, and preemptive versions play important roles in the analysis and construction of the algorithms addressed in this work.

2.1.2 Approximation Algorithms

Finding an algorithm that efficiently finds the optimal schedule for a given problem is not trivial. For instance, it has been proven that problems $1|r_j, pmtn|\sum_{j\in N} w_jC_j$ [Lab+84], $1|r_j|\sum_{j\in N} C_j$ [LKB77] and $P|(r_j)|\sum_{j\in N} w_jC_j$ [LKB77] are strongly NP-hard. Consequently, research efforts have been directed toward finding approximation algorithms that yield feasible schedules in polynomial time while providing a guarantee on the performance of the outputted schedule.

Definition 2.5 (Deterministic ρ- Approximation Algorithm)**.** *A polynomial-time algorithm A is a deterministic ρ-approximation for a given problem if, for all instances I, it holds that*

$$A(I) \leq \rho \cdot OPT(I), \tag{2.2}$$

where $A(I)$ denotes the objective value of the solution delivered by the algorithm A, and $OPT(I)$ is the optimal value for instance I. The value ρ is called the performance guarantee of algorithm A.

In certain scenarios, deterministic algorithms may struggle to provide good approximation consistently across all instances. Randomized algorithms introduce randomnesses in the decision process, allowing for more adaptability and avoiding instances where the algorithm behaves especially poorly [MR95]. They ensure that, on average, the solution they produce is within a factor of the optimal solution. The performance of these algorithms is evaluated by comparing the expected objective values of the schedule outputted by the algorithm and of the optimal schedule.

Definition 2.6 (Randomized $\rho-$ Approximation Algorithm)**.** *A polynomial-time algorithm A is a randomized ρ-approximation algorithm for a given problem if, for any instance I, it holds that*

$$E_{\sim X}[A(I)] \leq \rho \cdot OPT(I), \tag{2.3}$$

where $OPT(I)$ is the optimal value for instance I. $E_{\sim X}[A(I)]$ is the expected objective value of the solution delivered by the algorithm for instance I and $X = (X_1, \ldots, X_l)$ is the random vector influencing the algorithm's behavior. The value ρ is called the performance guarantee of algorithm A.

2.2 Scheduling with Uncertainty

In this section, we present the models that address uncertainty in scheduling problems. Firstly, we discuss stochastic and online scheduling individually. Subsequently, we explore the integration of these models into a combined framework.

2.2.1 Stochastic Scheduling

In stochastic scheduling problems, jobs' processing times are assumed to be stochastic. We denote by $P = (P_1, \dots, P_n)$ the random vector specifying the processing time of the jobs and by $p = (p_1, \dots, p_n)$ its realization. The actual value of p_j becomes known to the scheduler only when job j is completed. Depending on the problem, different information on P could be known. In some cases, we have access to its distribution, more often, we only have access to its expected value $E[P] = \mu = (\mu_1, \dots, \mu_n)$ and variance $Var(P) = (Var_1, \dots, Var_n)$. In this work, as generally done in literature, we will assume that jobs' processing times are stochastically independent. We refer to [MRW84] and [MRW85] for additional details on the theoretical foundations of stochastic scheduling.

In stochastic scheduling, the concept of coefficient of variation is often used to define bounds on the performance of the solutions. The coefficient of variation gives information on the variability of the random variable with respect to its scale.

Definition 2.7 (Coefficient of Variation)**.** *The coefficient of variation of a random variable P_j is defined as $CV[P_j] = \frac{\sqrt{Var(P_j)}}{E[P_j]}$.*

The solution to a stochastic scheduling problem is a scheduling policy. We will consider its definition in the framework of dynamic optimization [Rad84]. Roughly speaking, a scheduling policy specifies which jobs to schedule at any time for each realization of the processing times.

Definition 2.8 (Scheduling Policy)**.** *A Scheduling policy Π makes a scheduling action at decision times t based on the state of the system.*

- *The **state** of the system at any time t is given by the time t itself and the conditional probability of the jobs' processing times given the observed past. The observed past consists of the set of jobs already completed at time t, with their start and completion times, and the set of jobs already started but not completed at time t with their starting times.*
- *The **action** of a scheduling policy at time t consist of a tentative decision time $t_{tent} > t$ and a set $B(t) \subseteq N$ of jobs that are scheduled at time t. The time t_{tent} is the latest time in which a new policy action must occur, assuming no job is released or completed before t_{tent}.*
- *The **decision time** is the moment a policy takes its action. Given an action at time t, the next decision time is t_{tent}, or the time of the next job release or completion, whatever occurs first.*

We usually assume $t_{tent} = \infty$, which implies that the next action takes place when a job is released or completed. The policies we consider are *non-anticipatory*, as their decision at time t only depends on the current and past state of the system. In other words, the scheduler doesn't know the actual processing times of the jobs until they are completed. In this scenario, we denote an instance as $I = \{r, P, w, m\}$, where here P refers to the information we have on the distributions of the processing times (e.g., μ_j and Var_j or the distribution itself). For an instance I, we will denote the start and completion time of job j under the policy Π as $S_j^{\Pi}(I)$ and $C_j^{\Pi}(I)$, respectively. To shorten the notation, we will also write C_j^{Π} and S_j^{Π}. It is important to note that, unlike in the deterministic case, the start and completion times are random variables.

As shown in Example 2.2, it is unreasonable to expect to find a non-anticipative policy that minimizes apriori the objective value for any realization of the processing times. Therefore, the general goal in stochastic scheduling is to find a policy that minimizes the expectation of the objective value.

Example 2.2. *As in Example 2.1.4 from [Uet02], we consider a single machine problem with two jobs released at time zero that have unit weights. The processing time of the first jobs is either 1 or 3 with probability 0.5. The second job has a deterministic processing time of 2. If Job 1's processing time is 1,* *scheduling it first yields an objective value of 4, whereas scheduling Job 2 first results in an objective value of 5. Conversely, if Job 1's processing time is 3, scheduling it first leads to an objective value of 8, while scheduling Job 2 first results in an objective value of 7. Therefore, the optimal decision depends on the realization of Job 1's processing time, and it cannot be assessed apriori.*

Definition 2.9 (Optimal Policy in Stochastic Scheduling)**.** *Given a family of scheduling policies σ, a policy Π^{OPT} is optimal in expectation for the instance I if*

$$E[OPT(I)] := E_{\sim P}\Big[\Pi^{OPT}(I)\Big] = \inf_{\Pi\in\sigma} E_{\sim P}\Big[\Pi(I)\Big],$$

where $E_{\sim P}[\Pi(I)]$ is the expected objective value of policy Π on the instance I.

When considering the total weighted completion times objective function, the existence of an optimal policy follows from [MRW84]. One could wonder whether considering a deterministic problem that takes the expected values μ_j as processing times would give an insight into how to schedule the jobs. As shown in [MR89], this method fails even for simple problems. However, thanks to Jensen's inequality for convex functions, the deterministic counterpart can still provide a lower bound on the expected objective value. The deterministic scheduling framework can be seen as a special occurrence of the stochastic ones. Therefore, the difficulties in finding the optimal solution usually transfer to the stochastic case. Thus, also in this setting, we need to consider ρ-approximation algorithms. Their definitions are analogous to the ones in the deterministic case, with the only adjustment of considering the expectation of the objective values over the processing times. It is worth noting that there are problems in which the stochastic version can be solved easily, even though the deterministic one is NP-hard. This is a consequence of the different types of input and kinds of solutions we want to find. For instance, this is the case for the problem $P||C_{max}$. While its deterministic version is NP-hard [BCJS74], the stochastic one with exponentially distributed processing time is solved optimally by the "lowest expected processing time first" $(LEPT)$ rule [BDF81].

2.2.2 Online Scheduling

In online scheduling, the algorithm doesn't have access to the full input instance from the beginning. Instead, the input $I = \{r, p, w, m\}$ is revealed piece by piece, and the number and characteristics of jobs are unknown until their arrival. There are two main models of online scheduling: the online-time paradigm and the online-list paradigm. In the online-time model, jobs arrive over time. At each time t, the scheduler must decide which jobs to schedule. We typically work in the clairvoyant setting, where the scheduler learns a job's processing time upon release. On the other hand, in the non-clairvoyant model, denoted by online-time-nclv, we do not gain any information on p_j at r_j. In the online-list paradigm, jobs are organized in a list and are presented to the scheduler in the order dictated by the list at time $t = 0$. Upon presentation, the scheduler acquires complete information on the job's characteristics and must assign it to a specific machine and time slots immediately. Once this assignment is made, it cannot be altered thereafter. Unless diversely specified, we assume an online-time clairvoyant model. We refer to Prujs and Sgall survey for more details on online scheduling [PST04].

Generally, the lack of knowledge of the future prevents an online algorithm from finding an optimal solution. Therefore, the quality of an online algorithm is measured by comparing its performance to the one of an algorithm that has access to more information. In online scheduling, we often compare the developed algorithms to an optimal offline algorithm that knows the full input from the start [ST85].

Definition 2.10 (Deterministic c-Competitive Algorithm)**.** *A deterministic online algorithm A is c-competitive if, for any instance I of the given problem, it holds:*

$$A(I) \leq c \cdot OPT(I),$$ [1]

[1]The general definition is $A(I) \leq c \cdot OPT(I) + b$, for a fixed constant b. However, in the problems we consider, the additive constant can be ignored.

where $A(I)$ is the objective value of the solution delivered by the algorithm A for I and $OPT(I)$ is the optimal offline objective value. The competitive ratio c_A of algorithm A is the infimum c such that A is c-competitive.

Our goal is to find an algorithm A with the smallest possible competitive ratio. Analogously to the definitions of approximate algorithms, this definition focuses on the worst-case instance. This requirement can be relaxed again by considering randomized algorithms and analyzing the ratio between the expected value of the randomized algorithm and the objective value of the optimal offline algorithm. In online algorithms terminology, this corresponds to the oblivious adversary model [BD+94].

Definition 2.11 (Randomized c-Competitive Algorithm)**.** *A randomized online algorithm A is $c-$competitive if, for any instance I of the given problem, it holds:*

$$E_{\sim X}[A(I)] \leq c \cdot OPT(I),$$

where $OPT(I)$ is the optimal offline objective value for the instance I. $E_{\sim X}[A(I)]$ is the expected objective value of algorithm A for instance I, where X represent the random variables that influence the algorithm. The competitive ratio c_A of algorithm A is the infimum c such that A is c-competitive.

2.2.3 Stochastic Online Scheduling

Stochastic Online Scheduling (SOS) problems were introduced by Megow et al. [MUV06] and Chou et al. [Cho+06]. In this setting, jobs arrive online. At their release, the scheduler only learns their attributes and some information on the distribution of their processing time. In this work, we will assume that upon release of the job j, the scheduler learns its weight w_j and the expected value μ_j of its processing time distribution. The goal is to find an approximation algorithm with the lowest possible performance guarantee. For the stochastic online setting, we consider the definition of ρ-approximation algorithm introduced in [MUV06].

Definition 2.12 (Deterministic ρ-Approximation in SOS)**.** *A stochastic online scheduling policy Π is a ρ-approximation for a given problem if, for any instance I, it holds:*

$$E_{\sim P}[\Pi(I)] \leq \rho \cdot E[OPT(I)], \tag{2.4}$$

where $E_{\sim P}[\Pi(I)]$ is the expected objective value of the policy Π for instance I. $E[OPT(I)]$ is the optimal value of an offline stochastic policy, which has apriori knowledge on the set of jobs N, their weights w_j, release times r_j, and processing time distributions P_j. The value ρ is referred to as the performance guarantee of policy Π.

As discussed in the previous sections, in some scenarios, it is convenient to analyze randomized algorithms.

Definition 2.13 (Randomized ρ-Approximation in SOS)**.** *A randomized stochastic online policy Π is a $\rho - approxiation$ if, for any instance I, it holds:*

$$E_{\sim P,X}[\Pi(I)] \leq \rho \cdot E[OPT(I))], \tag{2.5}$$

where $E_{\sim P,X}[\Pi(I)]$ is the expected objective value of the policy Π for instance I, and X denotes the random vector that influences the policy's behavior. $E[OPT(I)]$ is the optimal value of an offline stochastic policy for the instance I which has apriori knowledge on the set of jobs N, their release times r_j, weights w_j and processing time distributions P_j. The value ρ is referred to as the performance guarantee of policy Π.

To lighten the notation from now on, we will denote by $E[\Pi(I)]$ the expected objective value of policy Π for instance I. We omit the subscripts as it's often clear from the context over which variables we are taking the expectation.

2.3 Linear Programming Relaxations

In the context of approximation algorithms, linear programming (LP) relaxations have been shown to be particularly useful in scheduling problems. The idea, introduced by [Hal+97], is to consider a linear programming relaxation of the problem under consideration and prove that it can be solved efficiently. The solution of the relaxation is then used to design a feasible solution for the original problem. For instance, we want to find a policy Π such that

$$\sum_{j\in N} w_j C_j^{\Pi} := \inf\left\{ \sum_{j\in N} w_j C_j \middle| C \in Q \right\}, \tag{2.6}$$

where $Q := \{(E[C_1^{\Pi}], \dots, E[C_n^{\Pi}]) \mid \Pi \text{ non-anticipative policy}\} \subseteq \mathbb{R}^n$ is the performance space. Since we cannot characterize Q or its convex hull in general, we approximate Q by a polyhedron Z, which is defined by valid inequalities for Q, hence $Q \subseteq Z$. We then solve the linear programming relaxation and find $C^{LP} = (C_1^{LP}, \dots, C_n^{LP})$ s.t.

$$\sum_{j\in N} w_j C_j^{LP} = \min\left\{ \sum_{j\in N} w_j C_j \middle| C \in Z \right\}. \tag{2.7}$$

Let's Π^{LP} be a feasible policy for the original problem derived from the solution of the LP-relaxation. For instance, the solution of the LP-relaxation could be used to guide the assignment to the machines [LST90] or to derive a priority order of the jobs for a list scheduling policy [PSW95; SS02]. Clearly, $LP(I) := \sum_{j\in N} w_j C_j^{LP} \leq E[OPT(I)] \leq E\Big[\sum_{j\in N} w_j C_j^{\Pi^{LP}}\Big] = E[\Pi^{LP}(I)]$. Therefore, if we prove that $E[\Pi^{LP}(I)] \leq \rho E[LP(I)]$ for $\rho \geq 1$, it follows that $E[\Pi^{LP}(I)] \leq \rho E[OPT(I)]$, and hence Π^{LP} is a ρ-approximation. Additionally, since we usually don't know the optimal policy, we can use the lower bound $LP(I)$ on its objective value to evaluate the performances of our policies.

2.4 Literature Review

The literature on scheduling is quite extensive and contains a broad range of problems, methodologies, and results. Rather than attempting to make a comprehensive review, we focus on the findings and techniques most pertinent to our discussion or those demonstrating the current best performance. Specifically, we consider the problems $1|r_j|\sum_{j\in N} w_j C_j$ and $P|r_j|\sum_{j\in N} w_j C_j$ across stochastic and deterministic scenarios, considering both offline and online environments. To the best of our knowledge, the state-of-the-art performance guarantees and competitive ratios are summarized in Table 2.1.

One of the first results in scheduling theory is the optimality of the $WSPT$ rule for the single machine problem without release dates $1||\sum_{j\in N} w_j C_j$ [Smi+56]. However, the introduction of non-trivial release dates, or the extension to the multiple machines setting, renders the problem strongly NP-hard [LKB77]. For the identical parallel machine problem with trivial release dates, Kawaguchi et al. demonstrated that the $WSPT$ rule archives a performance guarantee of $\frac{\sqrt{2}+1}{2}$ [KK86]. For the scenarios with non-trivial release dates, Phillips et al. developed the first constant-factor approximation algorithms for the problems $1|r_j|\sum_{j\in N} w_j C_j$ and $P|r_j|\sum_{j\in N} w_j C_j$ [PSW95]. They established a performance guarantee of $16+\epsilon$ and $24+\epsilon$, respectively. Their approach involves using an LP-relaxation to solve the preemptive version of the problem. Subsequently, jobs are scheduled on the non-preemptive machines in order of their completion times in the preemptive schedule, with the additional constraint that no job can be started prior to its release dates. Hall et al. improved these results by employing similar methods [Hal+97; HSW96]. Specifically, they attained a 3-approximation algorithm for the single machine problem and a $(4-\frac{1}{m})$-approximation for the identical parallel machines problem. Moreover, they also presented the first result for the online problem. Partitioning the time horizon in geometrically increasing intervals and using dual fitting techniques, they

Table 2.1 Best-known upper bounds on the performance guarantees/competitive ratios of deterministic (DET.) and randomized (RAND.) approximation algorithms for offline and online problems in the stochastic and deterministic settings.

Problem		**Online**		**Offline**	
		Stochastic	Deterministic	Stochastic	Deterministic
$1\|r_j\|\sum_{j\in N} w_jC_j$	DET.	2.618 [Jäg21] $1.618 + 1.309\Delta^{\dagger}$	2 [AP04]	2 [Jäg21]	$PTAS$ [Afr+99] 1.6853 [Goe+02]
	RAND.	2 [Jäg21]	1.6853 [Goe+02]	2 [Jäg21]	1.6853 [Goe+02]
$P\|r_j\|\sum_{j\in N} w_jC_j$	DET.	$2.309 + 1.309\Delta^{\dagger}$	$1.791 + O(m)$ [Sit10] 2.11 [MT18]	$2 + \Delta$ [Sch08] $3 - \frac{1}{m} + \max\{1, \frac{m-1}{m}\Delta\}$ [MSU99]	$PTAS$ [Afr+99] 2 [SS02]
	RAND.	$2 + \Delta$ [Sch08]	$1.791 + O(m)$ [Sit10] 2 [SS02] $\rho_m \leq 2$ [CW05]	$2 + \Delta$ [Sch08] $3 - \frac{1}{m} + \max\{1, \frac{m-1}{m}\Delta\}$ [MSU99]	2 [SS02]

For some of the stochastic problems, better bounds can be achieved when considering specific processing time distributions. When the deterministic algorithms achieved smaller bounds than the randomized ones for the same problem, we also inserted these bounds in the randomized cell.

† Result obtained in this work.

found online algorithms with a $3 + \epsilon$ and $4 + \epsilon$ competitive ratio for the single and parallel machine problems, respectively. Additionally, in the analysis of problems with precedence constraints, they introduced the concept of α-points, denoting the earliest point in time where an α fraction of the job is completed in a preemptive schedule. This concept was later on employed in different problems to create jobs' priority lists based on the α-points of a preemptive relaxation. Goemans proposed to randomly select different α values for each job in order to obtain a more flexible algorithm that potentially gives better approximation results [Goe97]. In the same paper, he introduced the concept of mean-busy-time, denoting the average point in time when a job is processed. Using a suitable distribution for α and considering a LP-relaxation based on the mean-busy-time, Goemans et al. presented a randomized algorithms with a 1.6853 performance guarantee for the problems $1|r_j, (online)|\sum_{j\in N} w_jC_j$ [Goe+02]. Their online algorithm deviates from the offline one by postponing each job's earliest possible starting time from its release time to its α-point. Moreover, they proved a 1.5819 lower bound on the performance ratio of any randomized algorithms for the online problem. Finally, in the offline environment, they showed that their algorithm can be derandomized using the method of conditional probabilities [Spe94].

Anderson and Potts developed a deterministic approximation algorithm with a performance ratio of 2 for the online problem $1|r_j, online|\sum_{j\in N} w_jC_j$ [AP04], matching the established lower bound for the problem [Ves97]. They considered a modified version of the $WSPT$ rule where the available job with the highest ratio is delayed if the current time is smaller than the job's processing time. Afrati et al. found a polynomial time approximation scheme, $PTAS$, for the problems $1|r_j|\sum_{j\in N} w_jC_j$ and $P|r_j|\sum_{j\in N} w_jC_j$ [Afr+99]. Their strategy consisted of simplifying the input without significantly increasing the objective value, and then applying a fast dynamic solution to the modified instance. Subsequently, Schulz and Skutella developed a faster 2-approximation algorithm for the identical machine problem [SS02]. Inspired by the work of Chekuri et al. [Che+01], they considered the LP-relaxation of a preemptive single machine problem with modified processing times $q_j = \frac{p_j}{m}$. Their algorithm randomly assigns the jobs to machines and schedules them within each machine in order of the non-decreasing α-points in the relaxation's solution. Its derandomized version maintains the same performance guarantee. The randomized algorithm, with the additional constraint that no job can start processing before its α-point is reached, also works in the online setting. For $P|r_j, online|\sum_{j\in N} w_jC_j$, a better performance ratio $\rho_m \leq 2$ dependent on m was achieved by Correa and Wagner [CW05] by generalizing the ideas presented in [Goe+02] to the parallel machine problem. Correa and Wagner also proposed a deterministic version of the algorithm with a fixed $\alpha = \frac{\sqrt{5}-1}{2}$, and proved that its competitive ratio is 2.618. For large m, their results were improved by Sitter, who constructed a deterministic online approximation algorithm with performance ratio of $\frac{3e-2}{2e-2} \cdot (1 + \frac{1}{\sqrt{m}})^2$, which converges to 1.791 as m increases [Sit10]. To obtain this result, he considered a modified instance for a fast single machine, applied the non-preemptive $WSPT$ to it, and then scheduled the jobs on the machines in order of their α-points with $\alpha = 0$ for each job. Afterward, Ma and Tao demonstrated that a modified version of the $WSPT$ rule with delayed start times achieves a constant performance ratio of 2.11 [MT18]. The current strongest lower bounds on the performance ratio of deterministic and randomized algorithms for the problem $P|r_j, online| \sum_{j\in N} w_j C_j$ are 1.309 [Ves97] and 1.157 [Sei00], respectively. Therefore, there is still a relatively large performance gap in the online setting for identical parallel machines problems.

In the field of stochastic scheduling, one of the earliest results is the optimality of the preemptive $WSEPT$ for the problem $1||E[\sum_{j\in N} w_j C_j]$ [Rot66]. This result still holds in the presence of non-trivial release dates, when the processing times are exponentially distributed [Pin83]. For the parallel machine problem with trivial release dates, the $WSEPT$ is optimal if the processing times are exponential and the weights w_j and the ratios $\frac{w_j}{\mu_j}$ have the same relative order [Käm87]. However, this is not the case for a generic distribution, as shown in Example 8.2 of [Jäg21]. Generalizing the techniques used in deterministic scheduling, Möhring et al. proposed an LP-relaxation for the stochastic problem dependent on the expected values of the processing times μ_j [MSU99]. Their algorithm schedules the jobs in non-decreasing order of their completion time in solution of the LP-relaxation. They proved that, in the presence of non-trivial release dates, this deterministic algorithm has a performance guarantee of $3 - \frac{1}{m} + \max\{1, \frac{m-1}{m}\Delta\}$, where Δ is an upper bound on the squared coefficient of variation. In the particular case where $m = 1$, this guarantee simplifies to 3. In the same article, they proved that the $WSEPT$ rule has a performance guarantee of $1 + \frac{(\Delta+1)(m-1)}{2m}$ when all jobs are released at time zero.

Megow et al. were the first to analyze stochastic online scheduling problems [MUV06]. They considered processing times with $\gamma - NBUE$ distribution [2] and found a performance guarantee dependent on γ for the problem $P|r_j, p_j \sim stoch, online, |E[\sum_{j\in N} w_j C_j]$. In their algorithm, a job is allocated to the machine where its expected objective value is increased the least, given the previously assigned jobs. When a machine is idle, the available job with the highest $\frac{w_j}{\mu_j}$ among those assigned to that machine is started. For jobs availability, they considered modified release dates $\bar{r}_j = \max\{r_j, \beta \cdot \mu_j\}$, where β is a fixed parameter.

In the paper "Stochastic Online Scheduling Revisited", Schulz transferred the ideas of mean-busy-time based LP-relaxation, α-points, and faster virtual machines to the stochastic setting [Sch08]. In this thesis, we complete the proofs on the performance guarantees of the algorithms described in the paper for the problem $P|r_j, p_j \sim stoch, online|E[\sum_{j\in N} w_j C_j]$. Specifically, we prove that the randomized approximation algorithm developed by Schulz has a $2 + \Delta$ performance ratio. The derandomized version of this algorithm, obtained by the method of conditional probabilities, achieves the same performance guarantee, but it functions only in the offline setting. Schulz proposed a different deterministic algorithm for the online problem. We demonstrate that it achieves a $2.309 + 1.309\Delta$ performance ratio, providing a slight improvement on the ratio stated in the article. To our knowledge, these results remain the best ones for the stochastic online problems with generic processing times' distributions. Additionally, we show that in the single machine problem, the deterministic online algorithm has a $1.618 + 1.309\Delta$ performance guarantee. In his PhD thesis, Jäger applied the policies developed by Schulz to the problem $1|r_j, p_j \sim stoch, online|E[\sum_{j\in N} w_j C_j]$ [Jäg21]. He proved a 2.618 and 2 performance ratio for the deterministic and randomized algorithms, respectively. Using similar techniques to [Goe+02; Gup+21; Gup+20], he also showed that these results can be improved when Δ is known or when the processing times are $\gamma - NBUE$ by choosing the distribution of α accordingly.

[2]A random variable X is $\gamma - NBUE$ for $\gamma \geq 1$ if $E[X - t \mid X > t] \leq \gamma \cdot E[X]$ for all $t \geq 0$.

3 Stochastic Online Scheduling Revisited

In this chapter, we examine the randomized and deterministic polynomial time[1] algorithms developed by Schulz in the conference paper "Stochastic Online Scheduling Revisited" [Sch08] for the problem $P|r_j, p_j \sim stoch, online|E[\sum_{j\in N} w_jC_j]$. Firstly, we complete the proof on the validity of the LP-relaxation presented in the article for the problem $P|r_j, p_j \sim stoch|E[\sum_{j\in N} w_jC_j]$ and observe that it can be efficiently solved online. Using this new relaxation, we then describe the algorithms developed by Schulz and demonstrate the performance ratios stated in the paper. Moreover, we slightly improve the upper bound on the performance guarantee for the deterministic online algorithm. Notably, the algorithms discussed in this chapter rely on the expected processing times of the jobs and do not require additional information on the distribution.

3.1 A LP-Relaxation for Stochastic Scheduling

The first to propose an LP-relaxation for the stochastic problem $P|r_j, p_j \sim stoch|E[\sum_{j\in N} w_jC_j]$ were Möhring et al. [MSU99]. They proved that given any non-anticipatory policy Π, its vector of expected completion time $E[C_j]$ satisfies:

$$\sum_{j\in S}\mu_jC_j \geq \frac{1}{2m}\Big(\sum_{j\in S}\mu_j\Big)^2 - \frac{\Delta-1}{2}\sum_{j\in S}\mu_j^2 \quad \forall S\subseteq N,$$

where m is the number of machines considered, $\mu = (\mu_1, \dots, \mu_n)$ is the expected value of the jobs' processing time P, and Δ is an upper bound on the squared coefficient of variation.

Noticing that no job in the subset S can be started before time $r_{min}(S) := \min_{j\in S} r_j$, Schulz proposed a stronger inequality.

Lemma 3.1. *Given a non-anticipative policy Π for $P|r_j, p_j \sim stoch|E[\sum_{j\in N} w_jC_j]$, its corresponding vector $E[C^{\Pi}]$ of expected completion time satisfies the following inequality:*

$$\sum_{j\in S}\mu_j\Big(C_j + \frac{\Delta-1}{2}\mu_j\Big) \geq r_{min}(S)\sum_{j\in S}\mu_j + \frac{1}{2m}\Big(\sum_{j\in S}\mu_j\Big)^2 \quad \forall S\subseteq N. \tag{3.1}$$

Proof. Taking inspiration from the proof of Theorem 3.1 in [MSU99], we consider a fixed realization $p = (p_1, \dots, p_n)$ of the processing times $P = (P_1, \dots, P_n)$. We denote by $s_j := S_j^{\Pi}(p)$ and $c_j := C_j^{\Pi}(p)$ the start and completion time, respectively, of job j under policy Π when jobs have deterministic processing times p.

Firstly, we show that, for the deterministic problem $P|r_j|\sum_{j\in N} w_jc_j$ with processing times p, it holds that

$$\sum_{j\in S}p_jc_j \geq r_{min}(S)\sum_{j\in S}p_j + \frac{1}{2}\sum_{j\in S}p_j^2 + \frac{1}{2m}\Big(\sum_{j\in S}p_j\Big)^2 \quad \forall S\subseteq N. \tag{3.2}$$

[1] For the algorithms discussed in this thesis, we evaluate their running time in relation to the input size of the corresponding deterministic problems, where job processing times are substituted with their expected values.

For any subset $S \subseteq N$ and any machine $i \in M$, we define $S^i := \{j \in S : j \text{ is processed on } i\}$. For each subset S^i, we apply the following inequality established for the single machine problem $1|r_j|\sum_{j\in N} w_j c_j$ in [Goe96] and [QS95] :

$$\sum_{j\in S^i} p_j c_j \geq r_{min}(S^i)\sum_{j\in S^i} p_j + \frac{1}{2}\Bigg(\sum_{j\in S^i} p_j^2 + \Big(\sum_{j\in S^i} p_j\Big)^2\Bigg) \quad \forall S^i \subseteq N. \tag{3.3}$$

We obtain: $$\sum_{j\in S} p_j c_j = \sum_{i=1}^{m}\sum_{j\in S^i} p_j c_j \overset{(3.3)}{\geq} \sum_{i=1}^{m}\Big(r_{min}(S^i)\sum_{j\in S^i} p_j\Big) + \frac{1}{2}\sum_{i=1}^{m}\sum_{j\in S^i} p_j^2 + \frac{1}{2}\sum_{i=1}^{m}\Big(\sum_{j\in S^i} p_j\Big)^2$$
$$\geq r_{min}(S)\sum_{j\in S} p_j + \frac{1}{2}\sum_{j\in S} p_j^2 + \frac{1}{2m}\Big(\sum_{i=1}^{m}\sum_{j\in S^i} p_j\Big)^2.$$

The second inequality follows from the relation between the quadratic and arithmetic means, applied to the average of $\sum_{j\in S^i} p_j$ across the machines[2] i, and the fact that $r_{min}(S) \leq r_{min}(S^i)$ holds for any machine i.

We now consider $\sum_{j\in N} p_j s_j$. Since we are in the non-preemptive settings, it holds that $c_j = s_j + p_j$ and, for any $S \subseteq N$, we have that

$$\begin{aligned}\sum_{j\in S} p_j s_j &= \sum_{j\in S} p_j(c_j - p_j) = \sum_{j\in S} p_j c_j - \sum_{j\in S} p_j^2 \overset{(3.2)}{\geq} r_{min}(S)\sum_{j\in S} p_j + \frac{1}{2}\sum_{j\in S} p_j^2 + \frac{1}{2m}\Big(\sum_{j\in S} p_j\Big)^2 - \sum_{j\in S} p_j^2\\ &= r_{min}(S)\sum_{j\in S} p_j - \frac{1}{2}\sum_{j\in S} p_j^2 + \frac{1}{2m}\sum_{\substack{j,k\in S,\\ k\neq j}} p_j p_k + \frac{1}{2m}\sum_{j\in S} p_j^2\\ &= r_{min}(S)\sum_{j\in S} p_j + \Big(\frac{1}{2m} - \frac{1}{2}\Big)\sum_{j\in S} p_j^2 + \frac{1}{2m}\sum_{\substack{j,k\in S,\\ k\neq j}} p_j p_k.\end{aligned} \tag{3.4}$$

We take the expectation across the processing times in both sides of (3.4). Since we are considering non-anticipative policies, the start time S_j of a job is independent of the realization of its processing time p_j. In fact, S_j depends on the processing times of the jobs previously assigned; however, from the hypothesis, jobs' processing times are pairwise stochastically independent. Therefore, $\sum_{j\in S} E[P_j S_j] = \sum_{j\in S} E[P_j]E[S_j] = \sum_{j\in S}\mu_j E[S_j]$ and $E[P_j P_k] = E[P_j]E[P_k] = \mu_j\mu_k$. Considering these observations and recalling the propriety $Var_j = E[P_j^2] - \mu_j^2$, we obtain that for any $S \subseteq N$ it holds:

$$\begin{aligned}\sum_{j\in S}\mu_j E[S_j] &\overset{(3.4)}{\geq} r_{min}(S)\sum_{j\in S}\mu_j + \Big(\frac{1}{2m} - \frac{1}{2}\Big)\sum_{j\in S} E[P_j^2] + \frac{1}{2m}\sum_{\substack{j,k\in S,\\ k\neq j}} E[P_j P_k]\\ &= r_{min}(S)\sum_{j\in S}\mu_j + \Big(\frac{1}{2m} - \frac{1}{2}\Big)\sum_{j\in S}(Var_j + \mu_j^2) + \frac{1}{2m}\sum_{\substack{j,k\in S,\\ k\neq j}}\mu_j\mu_k\\ &= r_{min}(S)\sum_{j\in S}\mu_j + \Big(\frac{1}{2m} - \frac{1}{2}\Big)\sum_{j\in S} Var_j + \frac{1}{2m}\Big(\sum_{j\in S}\mu_j\Big)^2 - \frac{1}{2}\sum_{j\in S}\mu_j^2 \qquad (3.5)\\ &\overset{Var_j/\mu_j^2\leq\Delta}{\geq} r_{min}(S)\sum_{j\in S}\mu_j + \Big(\frac{1}{2m} - \frac{1}{2}\Big)\sum_{j\in S}\Delta\mu_j^2 + \frac{1}{2m}\Big(\sum_{j\in S}\mu_j\Big)^2 - \frac{1}{2}\sum_{j\in S}\mu_j^2\\ &= r_{min}(S)\sum_{j\in S}\mu_j + \frac{1}{2m}\Big(\sum_{j\in S}\mu_j\Big)^2 + \Big(\frac{\Delta}{2m} - \frac{\Delta}{2} - \frac{1}{2}\Big)\sum_{j\in S}\mu_j^2.\end{aligned}$$

[2]The quadratic mean is greater than or equal to the arithmetic mean, that is $\sqrt{\frac{\sum_{i=1}^{m} x_i^2}{m}} \geq \frac{\sum_{i=1}^{m} x_i}{m}$. This inequality is applied here with $x_i = \sum_{j\in s_i} p_j$.

Since we are in the non-preemptive setting, $E[C_j] = E[S_j + P_j] = E[S_j] + \mu_j$. Considering $\sum_{j\in S} \mu_j E[C_j]$ and applying (3.5), it follows that for any $S \subseteq N$ we have:

$$\begin{aligned}\sum_{j\in S}\mu_j E[C_j] &= \sum_{j\in S}\mu_j E[S_j] + \sum_{j\in S}\mu_j^2 \overset{(3.5)}{\geq} r_{min}(S)\sum_{j\in S}\mu_j + \frac{1}{2m}\Big(\sum_{j\in S}\mu_j\Big)^2 + \Big(\frac{\Delta}{2m} - \frac{\Delta}{2} - \frac{1}{2} + 1\Big)\sum_{j\in S}\mu_j^2 \\ &= r_{min}(S)\sum_{j\in S}\mu_j + \frac{1}{2m}\Big(\sum_{j\in S}\mu_j\Big)^2 - \frac{\Delta-1}{2}\sum_{j\in S}\mu_j^2 + \frac{\Delta}{2m}\sum_{j\in S}\mu_j^2 \\ &\geq r_{min}(S)\sum_{j\in S}\mu_j + \frac{1}{2m}\Big(\sum_{j\in S}\mu_j\Big)^2 - \frac{\Delta-1}{2}\sum_{j\in S}\mu_j^2.\end{aligned}$$

□

From Lemma 3.1, it follows that a possible linear programming relaxation for the problem $P|r_j, p_j \sim stoch|E[\sum_{j\in N} w_j C_j]$ is

$$\begin{aligned}&\text{minimize } \sum_{j\in N} w_j C_j \\ &\text{such that } \sum_{j\in S}\mu_j\Big(C_j + \frac{\Delta-1}{2}\mu_j\Big) \geq r_{min}(S)\sum_{j\in S}\mu_j + \frac{1}{2m}\Big(\sum_{j\in S}\mu_j\Big)^2 \quad \forall S\subseteq N.\end{aligned} \tag{3.6}$$

Setting $M_j := C_j + \frac{\Delta-1}{2}\mu_j$ and dividing both sides of (3.1) by m, we obtain the following inequalities:

$$\sum_{j\in S}\frac{\mu_j}{m}M_j \geq r_{min}(S)\sum_{j\in S}\frac{\mu_j}{m} + \frac{1}{2}\Big(\sum_{j\in S}\frac{\mu_j}{m}\Big)^2 \quad \forall S\subseteq N. \tag{3.7}$$

Defining $q_j := \frac{\mu_j}{m}$ and dropping the constant term $\frac{1-\Delta}{2}\sum_{j\in S} w_j\mu_j$ from the objective function, we derive the following linear program which is equivalent to (3.6):

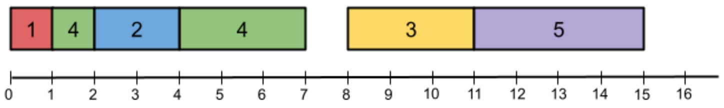
$$\begin{aligned}&\text{minimize } \sum_{j\in N} w_j M_j \\ &\text{such that } \sum_{j\in S} q_j M_j \geq r_{min}(S)\sum_{j\in S} q_j + \frac{1}{2}\Big(\sum_{j\in S} q_j\Big)^2 \quad \forall S\subseteq N.\end{aligned} \tag{3.8}$$

This linear program (3.8) can be also interpreted as a relaxation of the single machine scheduling problem $1|r_j|\sum_{j\in N} w_j C_j$ with deterministic processing times q_j.

In the subsequent analysis, we will use the mean-busy-time variables, a concept introduced by Goemans in [Goe97]. For a single machine problem, the mean-busy-time M_j^S of job j in the schedule S is the average point in time in which the machine is busy processing j. Formally, $M_j^S := \frac{1}{q_j}\int_{r_j}^{\infty}\chi_j^S(t)t\,dt$, where $\chi_j(t)$ is 1 if the machine is processing job j at time t, and 0 otherwise. We define the *LP-schedule* as the preemptive schedule on the single machine that, at any point in time, schedules the available and not yet completed job with the highest ratio of weight w_j to processing times q_j; that is the preemptive $WSPT$ rule on the single machine. Goemans et al. proved that the means-busy-time vector of this schedule optimally solves the linear programming relaxation (3.8) [Theorem 2.5 [Goe+02]]. In the proof of this theorem and in the subsequent sections, we relay on the notion of canonical decomposition introduced by Goemans [Goe96].

Definition 3.1 (Canonical Decomposition)**.** *A partition $S_1, \dots, S_k$ of a set of jobs $S \subseteq N$ is a canonical decomposition of the set S if, given a schedule that processes the jobs in S as soon as possible, the machine is busy processing the jobs in S exactly in the disjoint intervals $[r_{min}(S_l), r_{min}(S_l) + \sum_{j\in S_l} q_j]$ for $l = 1, \dots, k$. A set S is canonical if its decomposition consists of the set S itself.*

Example 3.1. *To gain a better understanding of the concept of canonical decomposition and canonical sets, let's consider an example from [Jäg21]. In this scenario, there are 5 deterministic jobs with unit weights. Their processing times q_j and release times r_j are shown in the table in Figure 3.1. We notice that $\frac{w_1}{q_1} \geq \cdots \geq \frac{w_5}{q_5}$. The figure illustrates the schedule derived from applying the preemptive $WSPT$ rule to these jobs. The canonical set for this schedule are: $\{1\}, \{2\}, \{3\}, \{1,2,4\}, \{3,5\}$. The canonical decomposition of the jobs is comprised of the sets $\{1,2,4\}$ and $\{3,5\}$.*



Jobs	1	2	3	4	5
r_j	0	2	8	0	9
q_j	1	2	3	4	4

Figure 3.1 Schedule generated by applying the preemptive $WSPT$ rule on a single machine for the set of jobs described in the adjacent table.

To leverage Equation (3.1), it is beneficial to express the weighted sum of completion times (mean-busy-times) as a sum over the jobs of q_jC_j ($q_jM_j^{LP}$).

Observation 3.1. *Let $[h]$ denote the subset of jobs $\{1,\dots,h\} \subseteq N$ and let $S_1^h, ..., S_{k(h)}^h$ be a possible decomposition of $[h]$. We assume without loss of generality that $\frac{w_1}{q_1} \geq ... \geq \frac{w_n}{q_n} \geq \frac{w_{n+1}}{q_{n+1}} = 0$. Then, for any vector $X = (X_1, ..., X_n)$ it holds that*

$$\sum_{j\in N} w_jX_j = \sum_{h=1}^{n} \left(\frac{w_h}{q_h} - \frac{w_{h+1}}{q_{h+1}}\right) \sum_{j\in[h]} q_jX_j = \sum_{h=1}^{n} \left(\frac{w_h}{q_h} - \frac{w_{h+1}}{q_{h+1}}\right) \sum_{l=1}^{k(h)} \sum_{j\in S_l^h} q_jX_j. \tag{3.9}$$

The first equality follows from observing that each X_j in the right hand side is multiplied by $\sum_{h=j}^{n} \left(\frac{w_h}{q_h} - \frac{w_{h+1}}{q_{h+1}}\right)q_j = \left(\frac{w_j}{q_j} - \frac{w_{n+1}}{q_{n+1}}\right)q_j = \frac{w_j}{q_j}q_j = w_j$.

Theorem 3.1 (Theorem 2.5 in [Goe+02])**.** *The mean-busy-time vector of the jobs under the LP-schedule is an optimal solution to the linear programming relaxation* (3.8)*.*

Proof. We denote by M^{LP} the mean-busy-time vector of the jobs under the LP-schedule.
We start by showing the feasibility of M^{LP} for (3.8). Given a subset $S \subseteq N$, we rewrite $\sum_{j\in S} q_jM_j^{LP}$ as

$$\sum_{j\in S} q_jM_j^{LP} = \sum_{j\in S} \int_{r_{min}(S)}^{\infty} \chi_j^{LP}(t)\, t\, dt = \int_{r_{min}(S)}^{\infty} \sum_{j\in S} \chi_j^{LP}(t)\, t\, dt.$$

The last integral is minimized among feasible preemptive schedules when all the jobs in S are processed continuously between $r_{min}(S)$ and $r_{min}(S) + \sum_{j\in S} q_j$. Therefore,

$$\sum_{j\in S} q_jM_j^{LP} = \int_{r_{min}(S)}^{\infty} \sum_{j\in S} \chi_j^{LP}(t)\, t\, dt \geq \int_{r_{min}(S)}^{r_{min}(S)+\sum_{j\in S} q_j} t\, dt = r_{min}(S) \sum_{j\in S} q_j + \frac{1}{2}\Big(\sum_{j\in S} q_j\Big)^2. \tag{3.10}$$

We proceed to prove the optimality of M^{LP}. Let $[h] = \{1,\dots,h\} \subseteq N$ be a subset of jobs and $S_1^h, \dots, S_{k(h)}^h$ a canonical decomposition of $[h]$. By construction, the LP-schedule processes the jobs in any of the canonical sets S_l^h continuously from $r_{min}(S_l^h)$ to $r_{min}(S_l^h) + \sum_{j\in S_l^h} q_j$. Therefore, given a feasible M for (3.8) and a canonical set S_l^h it holds:

$$\sum_{j\in S_l^h} q_jM_j \overset{(3.8)}{\geq} r_{min}(S) \sum_{j\in S_l^h} q_j + \frac{1}{2}\Big(\sum_{j\in S_l^h} q_j\Big)^2 = \sum_{j\in S_l^h} q_jM_j^{LP}. \tag{3.11}$$

Without loss of generality, we assume jobs are indexed in non-decreasing order of the ratios $\frac{w_j}{q_j}$. Applying (3.9) to the vectors M and M^{LP} and summing over the canonical sets we obtain:

$$\begin{aligned} \sum_{j\in N} w_j M_j &\overset{(3.9)}{=} \sum_{h=1}^{n} \left(\frac{w_h}{q_h} - \frac{w_{h+1}}{q_{h+1}}\right) \sum_{l=1}^{k(h)} \sum_{j\in S_l^h} q_j M_j \\ &\overset{(3.11)}{\geq} \sum_{h=1}^{n} \left(\frac{w_h}{q_h} - \frac{w_{h+1}}{q_{h+1}}\right) \sum_{l=1}^{k(h)} \sum_{j\in S_l^h} q_j M_j^{LP} \overset{(3.9)}{=} \sum_{j\in N} w_j M_j^{LP}. \end{aligned} \tag{3.12}$$

□

It is worth noting that the LP-schedule can be constructed in $O(n \cdot \log n)$. Let's consider the following dynamic implementation. We maintain a priority list of the currently available and uncompleted jobs, where jobs are ordered in non-decreasing order of the ratio $\frac{w_j}{q_j}$. Additionally, the list keeps track of the remaining processing time for each job. We make a scheduling decision only when a job is released or completed. In the first case, the job is added to the list in the correct position, while in the latter case, the completed job is removed from the list. In either case, the first element in the list is then processed until the next decision time. If the list is empty, we skip to the next decision time. This algorithm consists of $O(n)$ priority list operations, where each of them can be implemented on $O(\log n)$. Hence, it follows that the running time of the algorithm is $O(n \cdot \log n)$.

Observation 3.2. *Clearly, this implementation also works in the online setting, and therefore, it can be used to develop algorithms for online scheduling problems. Additionally, since the LP-schedule solves the relaxation of our problem, it provides the following lower bound on the expected value of the optimal offline policy, $\sum_{j\in N} w_j(M_j^{LP} - \frac{\Delta-1}{2}\mu_j)$.*

3.2 RSOS: A Randomized Algorithm for Stochastic Online Scheduling

The randomized algorithm developed by Schulz for the problem $P|r_j, p_j \sim stoch, online|E[\sum_{j\in N} w_j C_j]$ can be seen as an extension to the stochastic setting of the $RANDOM\ ASSIGNMENT$ algorithm developed in [SS02] and the $NASR$ algorithm developed in [CW05]. The approach involves employing the LP-schedule on a fast virtual machine with processing requirements $q_j = \frac{\mu_j}{m}$ to construct a feasible solution for our problem. The conversion is made leveraging the jobs' α-points, a concept introduced in [Hal+97; HSW96].

Definition 3.2 (α-points)**.** *Given a preemptive schedule S on a single machine and $\alpha \in [0,1]$, the α-point $t_j(\alpha)$ of job j is the first point in time when an α-fraction of the job j has been completed by the schedule. That is,*

$$t_j(\alpha) := \min\left\{t \geq 0 : \int_0^t \chi_j^S(s)\, ds \geq \alpha \cdot q_j\right\}.$$

At its α-point, a job j has been processed for $\alpha \cdot q_j$ units of time. In particular, when α is 0 or 1, the α-point corresponds to the job's start and completion time, respectively.

Observation 3.3. *From the definition of α-points and LP-schedule, it follows that the mean-busy-time M_j^{LP} of any job j is equivalent to the average of its α-point over α. In fact, for any job j the function $\alpha \to t_j(\alpha)$ is piece-wise affine linear with slope q_j. Therefore, applying the transformation $t_j(\alpha) =: s$ and noticing that $\chi_j^{LP}(s)\, ds = q_j\, d\alpha$ we obtain:*

$$\int_0^1 t_j(\alpha)\, d\alpha = \int_0^\infty \frac{s}{q_j} \chi_j^{LP}(s)\, ds = M_j^{LP}. \tag{3.13}$$

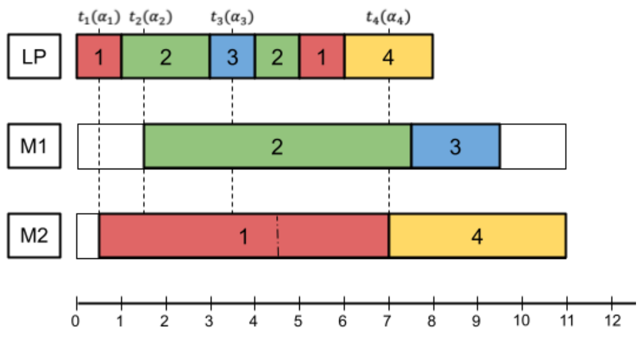
We now proceed to describe the $RSOS$ (Randomized Stochastic Online Scheduling) algorithm developed by Schulz [Algorithm 1]. Similarly to [SS02], we consider a preemptive virtual single machine with deterministic processing requirement $q_j = \frac{\mu_j}{m}$. We simultaneously maintain the preemptive LP-schedule on the virtual single machine and the actual schedule on the m machines. Whenever a job j is released, the $RSOS$ algorithm draws an $\alpha_j \in (0, 1]$ uniformly at random. The job j is then inserted in a priority list L where released jobs are sorted by non-decreasing ratios $\frac{w_j}{\mu_j}$. In the virtual single machine, at any point in time, the first job in L is processed preemptively. As soon as a job j reaches its α_j-point, a machine i is randomly picked with probability $\frac{1}{m}$ and job j is enqueued in the FIFO queue Q_i of the machine i. On each real machine i, the jobs are processed non-preemptively accordingly to Q_i. When a job is completed in the virtual machine, it is deleted from L; when it is completed in one of the real machines i, it is removed from its queue Q_i. In the priority list and queues, ties are broken arbitrarily when inserting a new job. Additionally, the draw of α_j and the random assignment of job j to the machine are assumed to be independent and independent of the ones of the other jobs. For completeness, we point out that, differently from the description of the LP-schedule, we consider a priority list with the equivalent priority order of $\frac{w_j}{\mu_j}$ instead of $\frac{w_j}{q_j}$. The running time of the $RSOS$ algorithm is $O(n \cdot \log n)$. This directly follows from the computational complexity of the previously described algorithm that dynamically constructs the LP-schedule. In fact, in the implementation of the $RSOS$ algorithm, we only consider an additional constant number of operations for every job.

Algorithm 1 RSOS

```
Input: $I = \{r, \mu, w, m\}$ revealed online.
    Initialize:
        • Empty priority list $L$,
        • Empty FIFO queues $Q_1, \dots, Q_m$.
    At every release time $r_j$:
        • Draw $\alpha_j$ uniformly in $(0, 1]$.
        • Insert job $j$ into $L$ according to the priority order determined by non-decreasing values of $\frac{w_j}{\mu_j}$.
        • Process the first job in $L$ on the virtual machine.
    At every $t_j(\alpha_j)$:
        • Draw $i \in \{1, \dots, m\}$ with probability $\frac{1}{m}$.
        • Enqueue $j$ in $Q_i$.
        • Process the first job in $Q_i$ on machine $i$.
    Every time a job $j$ is completed on machine $i$:
        • Dequeue $j$ from $Q_i$.
        • Process the first job in $Q_i$ on machine $i$.
    Every time a job $j$ is completed on the virtual machine:
        • Delete $j$ from $L$.
        • Process the first job in $L$ on the virtual machine.
```

Before delving into the analysis of this algorithm, let's briefly explore the rationale behind its construction. In online settings, delaying job assignments is advantageous for acquiring additional information on future jobs. This strategy is particularly beneficial when a high-weight, short-processing job follows a low-weight, long-processing one. By prioritizing jobs based on their weight to processing time ratio, a standard practice in scheduling, we can effectively prioritize jobs with high weights while considering the potential delay on other job completions. In the virtual single machine, we consider the expected processing requirement divided by m in order to reflect the processing capacity of the actual system. This adjustment ensures that jobs are not excessively delayed, preventing many machines from being deliberately idle for extended periods of time. Additionally, integrating random α values and random job assignments helps to mitigate scenarios where the algorithm demonstrates especially unfavorable behavior and better tackles uncertainty. In Example 3.2, we can observe the positive effect of these randomization steps.

Example 3.2. *To gain a better understanding of the $RSOS$ algorithm, we illustrate its behavior on the instance defined in the table in Figure 3.2. The table also contains the realization p_j of the random variables P_j and α_j for each job. In the figure, the resulting schedule alongside the LP-schedule is shown. The objective value of the schedule depicted in the picture, for the specific realization of the processing times in this example, is $7 \cdot 4 + 7.5 \cdot 12 + 9.5 \cdot 6 + 11 \cdot 16 = 351$. In our scenario, at $t_3(\alpha_3)$, Job 3 is randomly assigned to machine $M1$. By employing diverse assignment rules, such as assigning to the machine with the lowest expected load or the machine that becomes idle first, Job 3 would be assigned to machine $M2$ and job 4 to machine $M1$. This alternative assignment results in an objective value of $7 \cdot 4 + 7.5 \cdot 12 + 9 \cdot 6 + 11.5 \cdot 16 = 356$. Thus, we can observe how, in practice, the randomization steps can enable us to address the uncertainty in the problem more effectively.*



Jobs	r_j	w_j	μ_j	w_j/μ_j	q_j	p_j	α_j
1	0	4	4	1	2	6.5	1/4
2	1	12	6	2	6	6	1/6
3	3	6	2	3	3	2	1/2
4	6	16	4	4	2	4	1/2

Figure 3.2 The LP-schedule and the schedule resulting from applying the $RSOS$ algorithm on two machines, $M1$ and $M2$, when considering the instance described in the adjacent table. The dashed line within the box corresponding to Job 1 represents the expected completion time of the job at its starting time.

Theorem 3.2. *Let I be an instance for the problem $P|r_j, p_j \sim stoch, online|E[\sum_{j\in N} w_j C_j]$. Then,*

$$E[RSOS(I)] \leq (2 + \Delta)E[OPT(I)],$$

where $E[OPT(I)]$ is the expected objective value of the optimal offline non-anticipative scheduling policy for I and Δ is an upper bound on the squared coefficient of variation of the jobs' processing times.

Proof. To lighten the notation, we will write C_j, instead of C_j^{RSOS}, to indicate the completion times of job j under the scheduling policy $RSOS$. Let j be an arbitrary job. Initially, we consider a fixed realization a_j of α_j and a fixed assignment of job j to machine i. Since $t_j(a_j) \geq r_j$, job j could be started as soon as it is assigned to the machine at time $t_j(a_j)$. However, if other jobs with smaller α-point have been already assigned to i, job j has to wait for their completion. Therefore, the expected completion time of job j conditioned on $\alpha_j = a_j$ and on the assignment to machine i, can be bounded as

$$E[C_j|\alpha_j = a_j, j \to i] \leq t_j(a_j) + \mu_j + \sum_{k\in N/\{j\}} \mu_k P(k \to i \text{ before } j|\alpha_j = a_j). \tag{3.14}$$

Where $k \to i$ indicates that job k is assigned to machine i. From the definition of the $RSOS$ policy, the probability that a job is assigned to a machine i is $\frac{1}{m}$, and it is independent of the previous assignment. The probability of a job being assigned before job j depends only on the order in the α-points in LP-schedule, which are influenced by α, r, μ, w and are independent from the realization of processing time. Therefore, we can rewrite the probability of a job being assigned before j as

$$P(k \text{ assigned before } j|\alpha_j = a_j) = P\Big(t_k(\alpha_k) < t_j(\alpha_j)\Big|\alpha_j = a_j\Big)$$
$$= P\left(\frac{1}{q_k}\int_0^{t_j(\alpha_j)} \chi_k(t)\,dt \geq \alpha_k \middle| \alpha_j = a_j\right) = \frac{1}{q_k}\int_0^{t_j(a_j)} \chi_k(t)\,dt, \tag{3.15}$$

where the last equality holds since for the uniform distribution in $[0,1]$ we have that $P(\alpha \leq c) = \int_0^c 1\,d\alpha = c$. By construction, the order in which jobs are assigned, i.e., the order of the α-points, and the assignment to a specific machine are independent. Therefore, we can rewrite (3.14) as

$$\begin{aligned} E[C_j|\alpha_j = a_j, j \to i] &\leq t_j(a_j) + \mu_j + \sum_{k\in N/\{j\}} \mu_k \frac{1}{m}\frac{1}{q_k}\int_0^{t_j(a_j)} \chi_k(t)\,dt \\ &\overset{q_k = \frac{\mu_k}{m}}{=} t_j(a_j) + \mu_j + \int_0^{t_j(a_j)} \Bigg(\sum_{k\in N/\{j\}} \chi_k(t)\Bigg)\,dt \leq t_j(a_j) + \mu_j + \int_0^{t_j(a_j)} 1\,dt \\ &= 2t_j(a_j) + \mu_j. \end{aligned} \tag{3.16}$$

The last inequality follows from $\sum_{k\in N/\{j\}} \chi_k(t) \leq 1$. This holds since, in the single virtual machine, we can process at most one job at a time. Deconditioning over a_j and the assignment to machine i we obtain:

$$\begin{aligned} E[C_j] &= \sum_{i=1}^m \frac{1}{m}\int_0^1 E[C_j|\alpha_j = a_j, j \to i]\,da_j \overset{(3.16)}{\leq} \sum_{i=1}^m \frac{1}{m}\int_0^1 2t_j(a_j) + \mu_j\,da_j \\ &= \int_0^1 2t_j(a_j) + \mu_j\,da_j \overset{(3.13)}{=} 2M_j^{LP} + \mu_j. \end{aligned} \tag{3.17}$$

Taking the weighted sum of the expected completion time, we have:

$$\begin{aligned} E[RSOS(I)] &= E\Big[\sum_{j\in N} w_j C_j\Big] \overset{(3.17)}{\leq} 2\sum_{j\in N} w_j M_j^{LP} + \sum_{j\in N} w_j\mu_j \\ &= 2\sum_{j\in N} w_j\Big(M_j^{LP} - \frac{\Delta - 1}{2}\mu_j\Big) + \Delta\sum_{j\in N} w_j\mu_j \overset{(3.2)}{\leq} 2E[OPT(I)] + \Delta E[OPT(I)]. \end{aligned} \tag{3.18}$$

The last inequality follows from Observation (3.2) and the fact that $\sum_{j\in N} w_j\mu_j$ is a lower bound of the optimal value, since it is the value we would obtain if we could process all jobs at the same time starting at $t = 0$. □

3.2.1 DSS: A Derandomized Algorithm for Stochastic Offline Scheduling

When a guarantee of consistent behavior is necessary for every application of the algorithm, a deterministic version may be preferred over a randomized one. In the offline setting, we can derive a deterministic algorithm from $RSOS$ that has the same performance guarantee. To construct it, we need to get rid of the two randomized steps: the choice of α_j and the assignment to the machines. At the time $t_j(\alpha_j)$, instead of randomly picking a machine, job j is assigned to the machine i with currently the lowest expected load [3], $\sum_{k:t_k(\alpha_k)\leq t_j(\alpha_j),k\to i} \mu_k$. The selection of a deterministic α is less straightforward. Similarly to [Goe+02], the idea is to use the LP-schedule and the method of conditional probabilities [Spe94] to select a fixed $\alpha = (\alpha_1, \ldots, \alpha_n)$ such that the resulting algorithm has the same performance guarantee of the randomized one. We will denote the resulting algorithm as DSS (Deterministic Stochastic Scheduling). Before describing the algorithm, we need to make some preliminary observations.

[3]Another possible assignment rule consists in starting the unscheduled job with the lowest reached α-point at any time a machine becomes idle. This strategy results in the same theoretical performance guarantee.

Observation 3.4. *In the LP-schedule a preemption can happen only when a job is released. Hence, we have at most $n-1$ preemptions, and the processing of any job j is divided into $l_j \leq n$ pieces. We denote the set of intervals for α_j associated to the l_j pieces of work by $A_j = \{A_j^1, \dots, A_j^{l_j}\}$ with $\cup_{k=1}^{l_j} A_j^k = [0,1]$ and $A_j^k \cap A_j^s = \varnothing$ for any $s \neq k$. The order in which jobs are assigned to the machines is influenced only by the intervals A_j^d in which the selected α_j lies and not specific α_j value. For instance, let's consider $\alpha_j^1, \alpha_j^2 \in A_j^d$ s.t. $\alpha_j^1 \leq \alpha_j^2$. By construction, no other job k is processed on the virtual machine between $t_j(\alpha_j^1)$ and $t_j(\alpha_j^2)$; therefore, no other job k can reach its α-point during this time interval.*

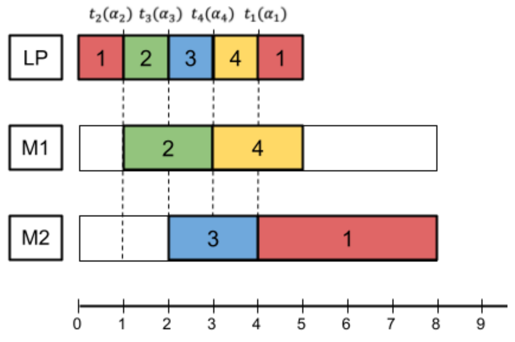
Taking into account this observation, we now proceed to describe the DSS algorithm [Algorithm 2]. Firstly, it constructs the LP-schedule at time $t=0$; this is possible since we are considering an offline environment. Then, it iteratively selects A_k^d that satisfies the following equation:

$$E\Big[UB(\alpha)\Big|\alpha_1 \in A_1^d, ..., \alpha_{k-1} \in A_{k-1}^d\Big] \geq E\Big[UB(\alpha)\Big|\alpha_1 \in A_1^d, ..., \alpha_{k-1} \in A_{k-1}^d, \alpha_k \in A_k^d\Big], \tag{3.19}$$

where $UB(\alpha) := \sum_{j \in N} w_j \Big(t_j(\alpha_j) + \mu_j + \frac{1}{m}\sum_{k: t_k(\alpha_k) < t_j(\alpha_j)} \mu_k\Big)$. For each job j it fixes $\bar{\alpha}_j$ as the minimum value within A_j^d. When its $\bar{\alpha}_j$-point is reached, job j is assigned to the machine with the current lowest expected load. In each machine, the jobs are processed in order of their α-points.

As observed in [Goe+02], for every $j \in \{1 \dots, n\}$ to check if A_j^d satisfies (3.19) we need to evaluate $O(n)$ terms, each of which can be computed in constant time. The number of job j's pieces, l_j, is equivalent to the number of times the job j is preempted plus 1. Since there are at most $n-1$ preemptions, we can bound the number of possible intervals as $\sum_{j=1}^n l_j \leq n - 1 + n = 2n-1$. Therefore, selecting $A_1^d, \dots, A_n^d$ that satisfy (3.19) takes $O(n^2)$ operations. As observed before, constricting the LP-schedule is $O(n \cdot \log n)$. For each job j, finding the machine with the current lowest expected load at $t_j(\alpha_j)$ takes $O(m)$ operations. Since it is reasonable to assume $m \leq n$, it follows that the DSS algorithm has a $O(n^2)$ running time.

Example 3.3. *To gain a better understanding of the DSS algorithm, we illustrate its behavior on the instance defined in the table in Figure 3.3. The table also contains the realization p_j of the processing times. In the picture, the resulting schedule alongside the LP-schedule is shown. In the latter, Jobs 2,3 and 4 are processed without preemption. Therefore, the DSS algorithm selects $\alpha_2 = \alpha_3 = \alpha_4 = 0$. On the other hand, the processing of Job 1 is divided into two pieces, and we need to select A_1^d, between $(0, 0.5)$ and $[0.5, 1]$, such that (3.19) is satisfied. Opting for the first interval, results in $\alpha_1 = 0$ which yields an objective value of $4 \cdot 1 + 3 \cdot 1 + 5 \cdot 10 + 6 \cdot 2 = 69$. Alternatively, selecting $\alpha_1 = 0.5$ results in an objective value of $8 \cdot 1 + 3 \cdot 1 + 4 \cdot 10 + 5 \cdot 2 = 61$. Consequently, the DSS algorithm chooses $A_1^d = [0.5, 1]$ and $\alpha_1 = 0.5$. This example also underscores the advantage of delaying jobs to keep the machine available for higher-weighted jobs.*


$t_2(\alpha_2)$ $t_3(\alpha_3)$ $t_4(\alpha_4)$ $t_1(\alpha_1)$
LP
1
2
3
4
1
M1
2
4
M2
3
1
0
1
2
3
4
5
6
7
8
9

Jobs	r_j	w_j	μ_j	w_j/μ_j	q_j	p_j
1	0	1	4	$1/4$	2	4
2	1	1	2	$1/2$	1	2
3	2	10	2	5	1	2
4	3	2	2	1	1	2

Figure 3.3 The LP-schedule and the schedule resulting from applying the DSS algorithm on two machines, $M1$ and $M2$, when considering the instance described in the adjacent table.

Algorithm 2 DSS

```
Input: $I = \{r, \mu, w, m\}$.
    Initialize: Empty FIFO queues $Q_1, \dots, Q_m$.
    At time $t = 0$:
        • Construct the preemptive $LP$-schedule on the virtual single machine with processing requirement
          $q_j = \frac{\mu_j}{m}$ by employing the preemptive $WSPT$ rule.
        • For $j = 1$ to $n$:
            – Determine the interval $A_j^d$ satisfying (3.19).
            – Set $\bar{\alpha}_j$ as the minimum value in $A_j^d$.
    At every $t_j(\bar{\alpha}_j)$-point:
        • Select machine $i$ with the current lowest expected load $\sum_{\substack{k:t_k(\alpha_k)\le t_j(\alpha_j)\\ k\to i}} \mu_k$.
        • Enqueue $j$ in $Q_i$.
        • Process the first job in $Q_i$ on machine $i$.
    Every time a job $j$ is completed on machine $i$:
        • Dequeue $j$ from $Q_i$.
        • Process the first job in $Q_i$ on machine $i$.
```

Corollary 3.1. *Let I be an instance for the problem $P|r_j, p_j \sim stoch|E[\sum_{j\in N} w_jC_j]$. Then,*

$$E[DSS(I)] \le (2 + \Delta)E[OPT(I)],$$

where $E[OPT(I)]$ is the expected objective value of the optimal offline non-anticipate scheduling policy for I and Δ is an upper bound on the squared coefficient of variation of the jobs' processing times.

Proof. For additional clarity, in this proof, we will specify when we are taking the expectation over the random variable α.

As in the $RSOS$, we suppose α_j is uniformly distributed in $[0, 1]$. Firstly, we consider a fixed job j and a fixed realization $a = (a_1, \dots, a_n)$ of $\alpha = (\alpha_1, \dots, \alpha_n)$. We denote by C_j^a the completion time of job j in the policy that, based on the LP-schedule, assigns the jobs at their a-points to the machine with the current lowest expected load and, in each machine, processes the jobs in the order of their assignment. $E[C_j^a]$ is equal to the sum of the expected processing time μ_j of job j and the expected time that the job has to wait before it can be started on the machine i to which it has been assigned. The latter can be bounded by the sum of $t_j(a_j)$ and the expected load of machine i before the assignment of j, which, in turn, can be bounded by the average expected load on the machines $\frac{1}{m}\sum_{k:t_k(a_k)<t_j(a_j)} \mu_k$ before the assignment. Therefore it holds that

$$E[C_j^a] = E_{\sim\alpha}[C_j^\alpha|\alpha = a] \le t_j(a_j) + \mu_j + \frac{1}{m}\sum_{k:t_k(a_k)<t_j(a_j)} \mu_k. \tag{3.20}$$

As in the proof of Theorem 3.2, deconditioning on the value a we obtain $E_{\sim\alpha}[C_j^\alpha|\alpha_j = a_j] \le 2t_j(a_j) + \mu_j$ and $E_{\sim\alpha}[C_j^\alpha] \le 2M_j^{LP} + \mu_j$. Summing over the jobs and taking the expectation over a, we obtain:

$$E_{\sim\alpha}\Big[\sum_{j\in N} w_jC_j^\alpha\Big] \overset{(3.20)}{\le} E_{\sim\alpha}\Big[UB(\alpha)\Big] \le \sum_{j\in N} w_j(2M_j^{LP} + \mu_j) \overset{(3.2)}{\le} (2 + \Delta)E[OPT(I)], \tag{3.21}$$

where $UB(\alpha) := \sum_{j\in N} w_j\Big(t_j(\alpha_j) + \mu_j + \frac{1}{m}\sum_{k:t_k(\alpha_k)<t_j(\alpha_j)} \mu_k\Big)$. From the law of total expectation and noticing that $\sum_{s=1}^{l_1} P(\alpha_1 \in A_1^s) = 1$ it follows that

$$\begin{aligned} E_{\sim\alpha}\Big[UB(\alpha)\Big] &= \sum_{s=1}^{l_1} P(\alpha_1 \in A_1^s) E_{\sim\alpha}\Big[UB(\alpha)\Big|\alpha_1 \in A_1^s\Big] \\ &\geq \min_{s=1,\dots,l_1} E_{\sim\alpha}\Big[UB(\alpha)\Big|\alpha_1 \in A_1^s\Big] = E_{\sim\alpha}\Big[UB(\alpha)\Big|\alpha_1 \in A_1^d\Big], \end{aligned} \tag{3.22}$$

where $A_1^d \subseteq [0,1]$ is the interval realizing the minima. Iterating this procedure for all the jobs $j = 1,\dots,n$ we obtain that

$$\begin{aligned} E_{\sim\alpha}\Big[UB(\alpha)\Big] &\geq \cdots \geq E_{\sim\alpha}\Big[UB(\alpha)\Big|\alpha_1 \in A_1^d, ..., \alpha_{k-1} \in A_{k-1}^d\Big] \\ &= \sum_{s=1}^{l_k} P\Big(\alpha_k \in A_k^s\Big) E_{\sim\alpha}\Big[UB(\alpha)\Big|\alpha_1 \in A_1^d, ..., \alpha_{k-1} \in A_{k-1}^d, \alpha_k \in A_k^s\Big] \\ &\geq \min_{s=1,\dots,l_k} E_{\sim\alpha}\Big[UB(\alpha)\Big|\alpha_1 \in A_1^d, ..., \alpha_{k-1} \in A_{k-1}^d, \alpha_k \in A_k^s\Big] \\ &= E_{\sim\alpha}\Big[UB(\alpha)\Big|\alpha_1 \in A_1^d, ..., \alpha_{k-1} \in A_{k-1}^d, \alpha_k \in A_k^d\Big] \\ &\geq \cdots \geq E_{\sim\alpha}\Big[UB(\alpha)\Big|\alpha_1 \in A_1^d, ..., \alpha_n \in A_n^d\Big]. \end{aligned} \tag{3.23}$$

We select $\bar\alpha = (\bar\alpha_1,\dots,\bar\alpha_n) \in A_1^d \times\cdots\times A_n^d$ s.t. $\bar\alpha_j$ is the minimum value in the interval A_j^d. This allows us to assign the jobs and, consequently, also start them as soon as possible while still maintaining the order dictated by the choices of the intervals A_j^d. This procedure for the selection of $\bar\alpha$ and for the assignment to the machines is the one prescribed in the DSS algorithm, therefore we have that

$$\begin{aligned} E[DSS(I)] = E\Big[\sum_{j\in N} w_j C_j^{\bar\alpha}\Big] &\overset{(3.20)}{\leq} UB(\bar\alpha) \leq E_{\sim\alpha}\Big[UB(\alpha)\Big|\alpha_1 \in A_1^d, ..., \alpha_n \in A_n^d\Big] \\ &\overset{(3.23)}{\leq} E_{\sim\alpha}[UB(\alpha)] \overset{(3.21)}{\leq} (2+\Delta)E[OPT(I)]. \end{aligned}$$

□

We point out that, for $\Delta \leq m-1$, the DSS algorithm improves the results of Mohring et al. who developed a $(3-\frac{1}{m}+\max\{1, \frac{m-1}{m}\Delta\})$-approximation algorithm for the offline problem $P|r_j, p_j \sim stoch|E[\sum_{j\in N} w_j C_j]$ [MSU99].

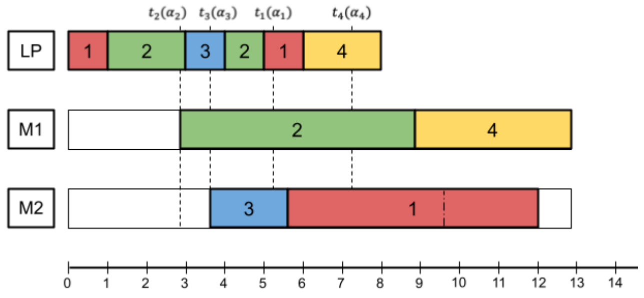
3.3 DSOS: A Deterministic Algorithm for Stochastic Online Scheduling

Schulz proposed a different deterministic version of the $RSOS$ policy that works also in the online setting [Algorithm 3]. The $DSOS$ (Deterministic Stochastic Online scheduling) algorithm differs from $RSOS$ in the choice of $\alpha = (\alpha_1,\dots,\alpha_n)$ and in the assignment rule to the machines. Instead of randomly drawing each α_j from the uniform distribution, their values are fixed beforehand to $\alpha_j = \alpha = \phi - 1 = \frac{\sqrt{5}-1}{2} = 0.618$, where ϕ is the golden ratio. The choice of this value α is motivated by its property $1 + \frac{1}{\alpha} = 2 + \alpha$, which yields the most favorable performance guarantee in the proof. At their α-points, jobs enter a FIFO queue instead of being assigned to a machine at random. Whenever a machine becomes idle, the first unscheduled job in the queue is started. Similarly to the $RSOS$ algorithm, the $DSOS$ has a time complexity of $O(n \cdot \log n)$. In [Sch08] it is stated that the $DSOS$ policy has performance ratio of $\max\{\alpha+2, \frac{\alpha+2}{2}\Delta + \frac{\alpha+4}{2}\}$. We refine this guarantee and demonstrate a performance ratio of $(\frac{\alpha+2}{2}\Delta + \frac{\alpha+4}{2})$. This new bound improves the previous one for $\Delta \leq 0.2361$. Our proof is inspired by the one of Theorem 3.2 in [CW05] for the deterministic online setting.

Algorithm 3 DSOS

```
Input: $I = \{r, \mu, w, m\}$ reveled online.
    Initialize:
        • Empty priority list $L$,
        • Empty FIFO queue $Q$,
        • $\alpha = \phi - 1$.
    At every release time $r_j$:
        • Insert job $j$ into $L$ accordingly to the priority order determined by non-decreasing values of $\frac{w_j}{\mu_j}$.
        • Process the first job in $L$ on the virtual machine.
    At every $t_j(\alpha)$:
        • Enqueue $j$ into $Q$.
    Every time a job $j$ is completed on machine $i$:
        • Dequeue $j$ from $Q$.
        • Process the first unscheduled job in $Q$ on machine $i$
    Every time a job $j$ is completed on the virtual machine:
        • Delete $j$ from $L$.
        • Process the first job in $L$ on the virtual machine.
```

Example 3.4. *To gain a better understanding of the $DSOS$ algorithm, we illustrate its behavior on the instance described in the table of Figure 3.4, which is the same one as Example 3.2. The table also contains the realization p_j of the processing times. In the picture, the resulting schedule alongside the LP-schedule is shown. The objective value of the schedule depicted in the figure is* $12.12 \cdot 4 + 8.85 \cdot 12 + 5.62 \cdot 6 + 12.85 \cdot 16 = 394.1$*. We notice that, in this specific instance, the $RSOS$ policy performs better.*



Jobs	r_j	w_j	μ_j	w_j/μ_j	q_j	p_j
1	0	4	4	1	2	6.5
2	1	12	6	2	6	6
3	3	6	2	3	3	2
4	6	16	4	4	2	4

Figure 3.4 The LP-schedule and the schedule resulting from applying the $DSOS$ algorithm on two machines, $M1$ and $M2$, when considering the instance described in the adjacent table. The dashed line within the box corresponding to Job 1 represents the expected completion of the job at its starting time.

Theorem 3.3. *Let I be an instance for the problem $P|r_j, p_j \sim stoch, online|E[\sum_{j \in N} w_j C_j]$. Then,*

$$E[DSOS(I)] \leq \left(\frac{2+\alpha}{2} \Delta + \frac{4+\alpha}{2} \right) E[OPT(I)],$$

where $\alpha = \phi - 1$, Δ is an upper bound on the squared coefficient of variation of the jobs' processing times and $E[OPT(I)]$ is the expected objective value of the optimal offline non-anticipate scheduling policy for I.

Proof. To lighten the notation, we let $C_j = C_j^{DSOS}$ denote the completion time of job j under the $DSOS$ policy, and let $C_j(p) = C_j^{DSOS}(p)$ denote its completion time for a fixed realization $p = (p_1, \dots, p_n)$ of the processing times. We define $\eta_k^j = \eta_k(\alpha)$ as the fraction of job k's processing requirement q_k completed in the virtual machine by time $t_j(\alpha_j)$. Clearly, if $\eta_k^j > \alpha$, jobs k is started on the real machines before job j. We note that $t_j(\alpha)$ and $\{k : \alpha \leq n_k^j\}$ are deterministically determined when α is fixed. In fact, they are influenced only by the values of r, $\frac{\mu}{m}, w, \alpha$ and don't depend on the actual realization of the processing times p.

Let's consider a fixed canonical set $S = \{1, ..., l\} \subseteq N$. From the definition of canonical sets it holds that $t_j(\alpha) = r_{min}(S) + \sum_{k \in S} \eta_k^j q_k$ for any $j \in S$. Assuming without loss of generality that $t_1(\alpha) \leq \ldots \leq t_l(\alpha)$, we have $\alpha \leq n_k^j \leq 1$ for $k < j$, and $n_k^j \leq \alpha$ for $k \geq j$. Therefore, we can bound $t_j(\alpha)$ as follows[4]:

$$t_j(\alpha) \leq r_{min}(S) + \sum_{k=1}^{j-1} q_k + \sum_{k=j}^{l} \alpha \cdot q_k = r_{min}(S) + (1-\alpha) \sum_{k=1}^{j-1} q_k + \alpha \sum_{k \in S} q_k. \tag{3.24}$$

We define $R := \{k : t_k(\alpha) < r_{min}(S)\}$ as the set of jobs that have reached their α-point before $r_{min}(S)$. Therefore, at $r_{min}(S)$, any job k in R has already been processed for at least $\alpha \cdot q_k$ units of time and we can use R to define a lower-bound for $r_{min}(S)$:

$$\alpha \sum_{k \in R} q_k \leq r_{min}(S). \tag{3.25}$$

By construction of the $DSOS$ algorithm, the expected completion time of a job j is equivalent to the sum of the time it has to wait to enter the queue Q, the expected time it waits in the queue and its expected processing time μ_j. The first two terms can be bounded from above by $t_j(\alpha) + \frac{1}{m} \sum_{k:\alpha \leq \eta_k^j, k \neq j} \mu_k$, which corresponds to the worst case scenario where all jobs start being processed after time $t_j(\alpha_j)$ and all the machines become idle at the same time after having processed an expected amount of work $\frac{1}{m} \sum_{k:\alpha \leq \eta_k^j, k \neq j} \mu_k$. Therefore we have that

$$E[C_j] \leq t_j(\alpha) + \frac{1}{m} \sum_{\substack{k:\alpha \leq \eta_k^j, \\ k \neq j}} \mu_k + \mu_j \overset{q_j = \frac{\mu_j}{m}}{=} t_j(\alpha) + \sum_{\substack{k:\alpha \leq \eta_k^j, \\ k \neq j}} q_k + m \cdot q_j. \tag{3.26}$$

Substituting (3.24) in (3.26), and noticing that $\{k : \alpha \leq \eta_k^j\} = R \cup \{1, ..., j\}$ we obtain:

$$\begin{aligned} E[C_j] &\overset{(3.24)}{\leq} r_{min}(S) + (1-\alpha) \sum_{k=1}^{j-1} q_k + \alpha \sum_{k \in S} q_k + \sum_{\substack{k:\alpha \leq \eta_k^j, \\ k \neq j}} q_k + m \cdot q_j \\ &= r_{min}(S) + (1-\alpha) \sum_{k=1}^{j-1} q_k + \alpha \sum_{k \in S} q_k + \sum_{k \in R} q_k + \sum_{k=1}^{j-1} q_k + m \cdot q_j \\ &= r_{min}(S) + (2-\alpha) \sum_{k=1}^{j-1} q_k + \alpha \sum_{k \in S} q_k + \sum_{k \in R} q_k + m \cdot q_j \\ &\overset{(3.25)}{\leq} r_{min}(S) \left(1 + \frac{1}{\alpha}\right) + (2-\alpha) \sum_{k=1}^{j-1} q_k + \alpha \sum_{k \in S} q_k + m \cdot q_j, \end{aligned} \tag{3.27}$$

Multiplying both sides by q_j and summing over the canonical set S we have that

$$\sum_{j \in S} q_j E[C_j] \leq r_{min}(S) \left(1 + \frac{1}{\alpha}\right) \sum_{j \in S} q_j + (2-\alpha) \sum_{j \in S} q_j \sum_{k=1}^{j-1} q_k + \alpha \Big(\sum_{j \in S} q_j\Big)^2 + m \sum_{j \in S} q_j^2. \tag{3.28}$$

[4]For $j = 1$, $\sum_{k=1}^{j-1} q_k$ is set to zero.

Observing that

$$\sum_{j\in S}\sum_{k=1}^{j-1} q_k q_j = \sum_{j\in S}\sum_{k=1}^{j-1} q_k q_j + \frac{1}{2}\sum_{j\in S} q_j^2 - \frac{1}{2}\sum_{j\in S} q_j^2 = \frac{1}{2}\sum_{j\in S}\Big(\sum_{k\in S} q_k q_j - q_j^2\Big) = \frac{1}{2}\Big(\sum_{j\in S} q_j\Big)^2 - \frac{1}{2}\sum_{j\in S} q_j^2,$$

we obtain:

$$\begin{aligned}
\sum_{j\in S} q_j E[C_j] &\le r_{min}(S)\left(1+\frac{1}{\alpha}\right)\sum_{j\in S} q_j + (2-\alpha)\left(\frac{1}{2}\Big(\sum_{j\in S} q_j\Big)^2 - \frac{1}{2}\sum_{j\in S} q_j^2\right) + \alpha\Big(\sum_{j\in S} q_j\Big)^2 + m\sum_{j\in S} q_j^2 \\
&= r_{min}(S)\left(1+\frac{1}{\alpha}\right)\sum_{j\in S} q_j + \left(\frac{2-\alpha}{2}+\alpha\right)\Big(\sum_{j\in S} q_j\Big)^2 + \left(m - \frac{2-\alpha}{2}\right)\sum_{j\in S} q_j^2 \\
&= r_{min}(S)\left(1+\frac{1}{\alpha}\right)\sum_{j\in S} q_j + \frac{2+\alpha}{2}\Big(\sum_{j\in S} q_j\Big)^2 + \left(m - \frac{2-\alpha}{2}\right)\sum_{j\in S} q_j^2 \\
&\le \max\left\{2+\alpha, 1+\frac{1}{\alpha}\right\}\Big(r_{min}(S) + \frac{1}{2}\sum_{j\in S} q_j\Big)\sum_{j\in S} q_j + \left(m - \frac{2-\alpha}{2}\right)\sum_{j\in S} q_j^2. \\
&\overset{(3.8)}{\le} \max\left\{2+\alpha, 1+\frac{1}{\alpha}\right\}\sum_{j\in S} q_j M_j^{LP} + \left(m - \frac{2-\alpha}{2}\right)\sum_{j\in S} q_j^2,
\end{aligned} \tag{3.29}$$

where the last inequality follows from the definition of the LP-relaxation. Since $\alpha = \phi - 1$ and $\phi = \frac{\sqrt{5}+1}{2}$, we have that[5] $2+\alpha = 1+\frac{1}{\alpha}$. Using this observation and multiplying both sides of the inequality (3.29) by m, we have that

$$\begin{aligned}
\sum_{j\in S}\mu_j E[C_j] &\le (2+\alpha)\sum_{j\in S}\mu_j M_j^{LP} + \left(1-\frac{2-\alpha}{2m}\right)\sum_{j\in S}\mu_j^2 \\
&= (2+\alpha)\sum_{j\in S}\mu_j\left(M_j^{LP} - \frac{\Delta-1}{2}\mu_j + \frac{\Delta-1}{2}\mu_j\right) + \left(1-\frac{2-\alpha}{2m}\right)\sum_{j\in S}\mu_j^2 \\
&= (2+\alpha)\sum_{j\in S}\mu_j\left(M_j^{LP} - \frac{\Delta-1}{2}\mu_j\right) + \left((2+\alpha)\frac{\Delta-1}{2} + 1 - \frac{2-\alpha}{2m}\right)\sum_{j\in S}\mu_j^2 \\
&= \sum_{j\in S}\mu_j\left[(2+\alpha)\left(M_j^{LP} - \frac{\Delta-1}{2}\mu_j\right) + \left((2+\alpha)\frac{\Delta-1}{2} + 1 - \frac{2-\alpha}{2m}\right)\sum_{j\in S}\mu_j\right].
\end{aligned} \tag{3.30}$$

Without loss of generality we assume that $\frac{w_1}{q_1} \ge \cdots \ge \frac{w_n}{q_n} = \frac{w_{n+1}}{q_{n+1}} = 0$. We denote with $S_1^h, \ldots, S_{k(h)}^h$ a canonical decomposition of the subset $[h] = \{1,\ldots,h\} \subseteq N$. Using Observation 3.1 with $E[C] = (E[C_1], \ldots E[C_n])$ and applying (3.30) to every canonical set S_l^i we obtain:

$$\begin{aligned}
&\sum_{j\in N} w_j E[C_j] \overset{(3.9)}{=} \sum_{i=1}^{n}\left(\frac{w_i}{q_i} - \frac{w_{i+1}}{q_{i+1}}\right)\sum_{l=1}^{k(i)}\sum_{j\in S_l^i}\frac{\mu_j}{m}E[C_j] \le \\
&\overset{(3.30)}{\le} \sum_{i=1}^{n}\left(\frac{w_i}{p_i} - \frac{w_{i+1}}{p_{i+1}}\right)\sum_{l=1}^{k(i)}\frac{1}{m}\sum_{j\in S_l^i}\mu_j\left[(2+\alpha)\left(M_j^{LP} - \frac{\Delta-1}{2}\mu_j\right) + \left((2+\alpha)\frac{\Delta-1}{2} + 1 - \frac{2-\alpha}{2m}\right)\mu_j\right] \\
&\overset{(3.9)}{=} \sum_{j\in N} w_j\left[(2+\alpha)\left(M_j^{LP} - \frac{\Delta-1}{2}\mu_j\right) + \left((2+\alpha)\frac{\Delta-1}{2} + 1 - \frac{2-\alpha}{2m}\right)\mu_j\right].
\end{aligned} \tag{3.31}$$

[5] We point out that the choice of $\alpha = \phi - 1$ in the $DSOS$ algorithm is motivated by its minimization of $\max\left\{2+\alpha, 1+\frac{1}{\alpha}\right\}$ within the interval $[0, 1]$.

Since, from Observation 3.2, $\sum_{j\in N} w_j\mu_j$ and $\sum_{j\in N} w_j\left(M_j^{LP} - \frac{\Delta-1}{2}\mu_j\right)$ are lower bounds on the expected value of the optimal policy, it follows that

$$\begin{aligned}
E[DSOS(I)] = \sum_{j\in N} w_j E[C_j] &\overset{(3.31)}{\leq} (2+\alpha)E[OPT(I)] + \left((2+\alpha)\frac{\Delta-1}{2} + 1 - \frac{2-\alpha}{2m}\right)E[OPT(I)] \\
&= \left((2+\alpha) + (2+\alpha)\frac{\Delta-1}{2} + 1 - \frac{2-\alpha}{2m}\right)E[OPT(I)] \qquad (3.32)\\
&\leq \left((2+\alpha) + (2+\alpha)\frac{\Delta-1}{2} + 1\right)E[OPT(I)] \qquad (3.33)\\
&= \left(\frac{2+\alpha}{2}\Delta + \frac{4+\alpha}{2}\right)E[OPT(I)].
\end{aligned}$$

□

For the sake of completeness, we point out that $\alpha = \phi - 1$ is the optimal choice even when considering the additional α-dependent term $\frac{\alpha}{2m}$ in (3.32). In in the proof's step from (3.32) to (3.33) we used that $1 - \frac{2-\alpha}{2m} \leq 1$. Since $1 - \frac{2-\alpha}{2m}$ tends to 1 as m increases, we cannot improve this bound while considering a generic number of machines. However, we can refine the performance ratio when m is fixed. For instance, when $m = 1$, we get the following performance guarantee:

$$\left((2+\alpha)\frac{\Delta+1}{2} + 1 - \frac{2-\alpha}{2m}\right) \overset{m=1}{=} \left((2+\alpha)\frac{\Delta+1}{2} + 1 - 1 + \frac{\alpha}{2}\right) = \left(\frac{2+\alpha}{2}\Delta + (1+\alpha)\right). \qquad (3.34)$$

Recently, Jäger proved that the $DSOS$ algorithm applied to one machine has a performance ratio $\phi + 1$ [Jäg21]. Which is smaller than (3.34) for $\Delta \geq \frac{2}{\alpha+2} = 0.764$.

Similarly, when $m = 2$ we have that

$$\left((2+\alpha)\frac{\Delta+1}{2} + 1 - \frac{2-\alpha}{2m}\right) = \left(2 + \alpha\frac{\Delta+1}{2} + \frac{2+\alpha}{4}\right) = \frac{2+\alpha}{2}\left(\Delta + \frac{3}{2}\right). \qquad (3.35)$$

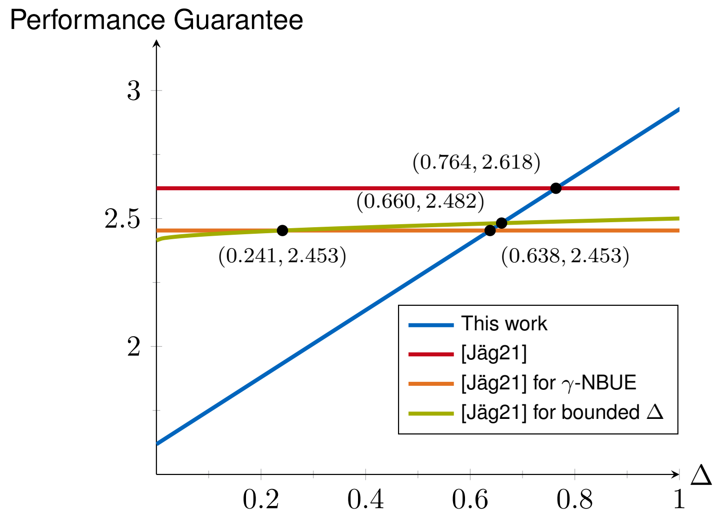


Figure 3.5 Comparison of the performance guarantee results for the $DSOS$ policy and its variations applied to the single machine problem.

For generic Δ, selecting $\alpha = \phi - 1$ gives the best performances. However, when the value of Δ is known in advance, we could use the additional information to find lower ratios. In the single machine problem, Jäger [Jäg21] showed that for a fixed Δ, the lowest performance ratio is obtained when choosing α that satisfies

$$(1 + g(\Delta)) \cdot \alpha^2 + (1 - g(\Delta)) \cdot \alpha = 1, \text{ where } g(\Delta) := \begin{cases} \frac{2-\sqrt{\Delta}}{2} \text{ if } \Delta \leq 1; \\ \frac{1}{\Delta+1} \text{ if } \Delta \geq 1. \end{cases}$$

In particular, he showed a performance guarantee of $c := \begin{cases} 1 + \frac{\sqrt{\Delta}+\sqrt{32-8\sqrt{\Delta}+\Delta}}{4} \text{ if } \Delta \leq 1; \\ 1 + \frac{2(2+\Delta)}{\sqrt{8+12\Delta+5\Delta^2}-\Delta} \text{ if } \Delta \geq 1. \end{cases}$

His results improves ours for $\Delta > 0.66$, and converges from below to $1 + \frac{2}{\sqrt{5}-1} = \phi + 1$ as Δ increases. When the processing times are $\gamma - NBUE$, Jäger improved this bound even further for $\Delta \geq 0.638$, proving a performance guarantee of 2.453 for $\alpha = 0.688$. The comparison between these bounds is depicted in Figure 3.5. Moreover, Jäger conducted a similar analysis for the application of the $RSOS$ policy to a single machine. He showed that, for a generic Δ, the $RSOS$ has a performance ratio of 2. Additionally, he demonstrated that when Δ is known or when the processing requirements follow a $\gamma - NBUE$ distribution, this result can be improved by choosing the distribution of α accordingly. The conditions he gave for these distributions are similar to the ones described in [Goe+02]. In future research, it would be interesting to extend these findings and methodologies to the parallel machines problem.

4 Computational Study

In the previous chapter we have derived theoretical guarantees for the performance of the $DSOS$ and $RSOS$ policy. In this chapter, we present an experimental study to assess the performance of these algorithms on simulated data. While the literature on scheduling theory is vast, experimental studies on algorithms based on LP-relaxation are relatively limited. Notably, one of the first computational studies in this realm is illustrated in [SUW05], where a comparison of different heuristics and approximation algorithms was conducted for the problem $1|r_j|\sum_{j\in N} w_jC_j$. Albers and Schröder considered an online problem in a parallel machine setting with the objective of minimizing the makespan [AS02]. Correa and Wagner conducted the first experiments for the problem $P|r_j, online|\sum_{j\in N} w_jC_j$ [CW05]. More recently, Buchem and Vredeveld presented an experimental study for policies for the stochastic online problem on uniform machines [BV21]. Our implementation was partially inspired by their work. In the next sections, we will illustrate the implementation of the simulations and discuss the obtained results.

4.1 Implementation

In the simulation, we considered various combinations of the number of machines $m \in \{1, 2, 5, 10\}$, the number of jobs $n \in \{10, 20, 50, 100, 500, 1000\}$, and the distributions of jobs' processing times. We analyzed different scenarios by varying release time conditions, weights, and expected processing time parameters. Jobs's weights w_j were selected uniformly in $(0, W)$. The expected value of each job's processing time μ_j was drawn uniformly from the integer numbers $\{1, \ldots, F\}$. We tested for different values of F and W. In order to analyze the results for distinct values of Δ, we simulated the realizations p_j of job j's processing times using four different distributions parameterized by μ_j:

- **Deterministic setting** $p_j = \mu_j$:
  $E[P_j] = \mu_j, \quad Var_j = 0 \implies CV[P_j]^2 = \Delta = 0.$
- **Uniform distribution** $p_j \sim Uniform(0, 2\mu_j)$:
  $E[P_j] = \mu_j, \quad Var_j = \frac{(4\mu_j^2)}{12} = \frac{\mu_j^2}{3} \implies CV[P_j]^2 = \Delta = \frac{1}{3}.$
- **Exponential distribution** $p_j \sim Exponential\Big(\frac{1}{\mu_j}\Big)$:
  $E[P_j] = \mu_j, \quad Var_j = \frac{1}{1/\mu_j^2} = \mu_j^2 \implies CV[P_j]^2 = \Delta = 1.$
- **Log-normal distribution** $p_j \sim Lognormal(\mu_j, \sigma^2)$:
  $E[P_j] = exp(\mu_j + \frac{\sigma^2}{2}), \quad Var_j = (exp(\sigma^2) - 1)exp(2\mu_j + \sigma^2)$
  $\implies CV[P_j]" = \frac{(exp(\sigma^2)-1)exp(2\mu_j+\sigma^2)}{exp(2\mu_j+\sigma^2)} = (exp(\sigma^2) - 1);$
  We considered $\sigma^2 = ln(11)$, and consequently $CV[P_j]^2 = \Delta = 10.$

Finally, the arrival time r_j of each job was drawn uniformly between 0 and $R(n, m)$, with the upper limit being either a fixed value R or varying with the number of jobs and machines. The case with R fixed was the one originally considered in [BV21]. However, fixing R may result in degenerate problems. If the value of R is too small compared to n, the algorithm will process most jobs after R. In this case, we don't have a "real" online arrival anymore, and the jobs are processed without any preemption in the virtual single machine. Conversely, if R is too large, the algorithm will process and finish most jobs before the next one arrives. Therefore, it is reasonable to consider an upper bound $R(n, m) = R \cdot n$ that depends on the number of jobs n. Additionally, after a first analysis of our results, we considered an upper bound on the release times dependent also on the number of machines in the scenario, $R(n,m) = \frac{R \cdot n}{m}\frac{F}{2}$. This new bound allowed us to test the performance for different numbers of machines m in similar tightness conditions.

For every unique combination of m, n, and processing time distribution, we generated 50 distinct instances with different weights, release times, and expected values of jobs' processing times. Since we are interested in the expected objective value, for each instance I, we conducted 50 simulations, generating different realizations of processing times, and executed the $RSOS$ ($DSOS$) algorithm on each of them. We then calculated the average objective value across these realizations to approximate $E[RSOS(I)]$ ($E[DSOS(I)]$). We are interested in the ratio between these expected objective values and the expected objective value of the optimal offline policy $E[OPT(I)]$. However, since we don't have access to the value of the latter, we used its lower bound $LB(I) = \max\left\{\sum_{j\in N} w_j\left(M_j^{LP} - \frac{\Delta-1}{2}\mu_j\right); \sum_{j\in N} w_j\mu_j\right\}$ to approximate the ratio. Therefore, the ratios computed in these experiments are greater than or equal to the actual ratios.

The simulations were implemented in Python version 3.12.0 on a macOS Sonoma 14.3.1 operating system, utilizing a 1.4 GHz Intel Core i5 quad-core processor. We refer to the code repository[1] for additional details on the simulation environment and implementation.

4.2 Results

Since we didn't observe any significant qualitative difference in the results when considering different values for F and W, we only show the experiments with $W = 30$ and $F = 50$. Most simulations show similar behavior, therefore, we focus our analysis on three specific settings that give useful insights on the influence of the different parameters. The results for $R(n,m) = 0.6 \cdot \frac{n \cdot F}{2m}$ are shown in Figure 4.1 and Table 4.1, while the results for $R(n,m) = 1.5 \cdot \frac{n \cdot F}{2m}$ are shown in Figure 4.2 and Table 4.2. Finally, the results of the simulations for $R(n,m) = 3 \cdot n$ are shown in Figure 4.3 and Table 4.3. Additional scenarios can be found in appendix A.1 and in the code repository. In order to visualize the general behavior of the ratios, we display the average ratio across the instances of the same scenario in the plots. In the tables, we report the average value of the ratio as well as the maximum observed ratio for the specific scenario.

Analyzing the results of the simulation, the first thing to notice is that the performance improves as the number of jobs increases, regardless of whether the release time upper bound is fixed or variable. This behavior is consistent across all simulations, independently of the parameters R, F, W, m. The results of our experiments suggest that the $RSOS$ and the $DSOS$ may be asymptotically optimal. Notably, asymptotic results that have been proved for related policies. For instance, when processing times are uniformly bounded, Gu and Lu [GL11] proved that $WSEPT$ is asymptotically optimal for $Q|r_j, p_j \sim stoch, online|E[\sum_{j\in N} w_jC_j]$. However, this condition doesn't hold for exponential and log-normal processing time distributions. Considering the insights from our simulations and relevant literature, it could be interesting to explore the asymptotic behavior of the algorithms presented here in future research.

In some of the scenarios, we observe that as n increases, the ratio increases before starting to decline. This is due to $\sum_{j\in N} w_j(M_j^{LP} - \frac{\Delta-1}{2}\mu_j)$ being particularly small, or even negative, when n is small and Δ is large. In these cases, $LB(I)$ coincides with $\sum_{j\in N} w_j\mu_j$, which is a substantial underestimation of the optimal objective value. As n increases, the discrepancy between $\sum_{j\in N} w_j\mu_j$ and the optimal value also increases. Therefore, the calculated ratio is significantly larger than the one we are approximating. The lower bound $\sum_{j\in N} w_j(M_j^{LP} - \frac{\Delta-1}{2}\mu_j)$, which is more complex and relevant to our problem, increases faster than $\sum_{j\in N} w_j\mu_j$ as n increases. After a certain value of n, $LB(I)$ coincides with $\sum_{j\in N} w_j(M_j^{LP} - \frac{\Delta-1}{2}\mu_j)$ and the ratio starts decreasing with n.

[1]https://github.com/MartaPip/Stochastic-Online-Scheduling/blob/main/README.md

To investigate the behavior of the algorithms under different degrees of uncertainty, we examine the performance ratios as Δ varies. As expected, the performance ratio increases as Δ increases. On one hand, $LB(I)$ remains the same across all the distributions considered, as it does not depend on the actual realization of the processing times. On the other hand, due to the higher uncertainty on the processing times, $E[RSOS(I)]$ and $E[DSOS(I)]$ increase with Δ. We notice that the performance of the $RSOS$ algorithm is often inferior to that of the $DSOS$ algorithm. This is likely due to the random machine assignment in the $RSOS$ policy, which does not take into account the current workload of the machines. This assumption is supported by the similar performances of the two policies for $m = 1$. As expected, for small m, the gap between the two policies decreases as Δ increases. In these cases, the $RSOS$'s random selection of α_j better handles the higher uncertainty in the processing times. In light of these considerations, we point out that, in practice, it may be interesting to consider a policy that randomly chooses values for α while adhering to the $DSOS$ method of machine allocation.

Examining the influence of the number of machines in the simulations, we observe that the performance ratio generally increases with m. This is a consequence of how varying m affects the lower bounds and the algorithms' results. When considering more machines, the more complex lower bound utilizes the additional resources to their full potential. In contrast, the $RSOS$ and $DSOS$ policies can fully exploit the additional resources only when m jobs are often processed simultaneously. Additionally, we can notice that the gap in the performance for different numbers of machines is more significant when employing the $RSOS$ policy; as discussed before, this is a result of the random assignment. When considering release time parameters $R(n, m)$ depending on m, we see that the performance ratio always increases with larger values of m. However, this relationship does not always hold when the upper bound on the release time is not influenced by the number of machines, especially when the release time is relatively large. For instance, in figure 4.3, we observe a higher ratio when $m = 5$, whereas other orders are displayed in different scenarios (Figure A.2, Table A.2). These differences can be explained by observing how changes in the release time conditions affect the lower bounds and the objective values of the $RSOS$ and $DSOS$ policies. When the value of $R(n, m)$ is large and independent from the number of machines, the lower bounds for different values of m tend to be closer. This occurs because, for large r_j, the variation in $p_j = \frac{\mu_j}{m}$ across different m is negligible compared to the magnitude of r_j. On the other hand, when there is greater variability in the arrival times, the gap in the performances of the $RSOS(DSOS)$ algorithm between scenarios with different m also decreases. However, this decrease is not as fast as the one of the lower bound. In fact, even though we don't use the additional capacity, scenarios with more machines still have the advantage that α-points are reached earlier, allowing jobs to start processing sooner.

As expected, all the ratios fall below the derived theoretical bounds. Additionally, in the deterministic setting and in the single machine scenario, they never exceed the performance guarantees proved for different algorithms and summarized in Table 2.1. The values of the worst-case instances suggest that there is potential for improving the performance guarantees of the $DSOS$ algorithm and of other algorithms tailored for the single machine problem. However, it is important to notice that we didn't test the algorithms on specifically designed instances in which they would behave particularly badly (e.g., an instance where jobs with higher priority keep being released before the previous ones reach their α-points, pathologically delaying the start of our jobs). Examining the result for the $RSOS$, we notice that in some scenarios the observed worst-case ratio is close to the theoretical upper bound (e.g., for the uniformly distributed processing time, $m = 10$ and $R(n, m) = 0.6 \cdot \frac{n \cdot F}{2m}$, we have observed a worst-case expected ratio of 2.145 where our upper bound is 2.333, Table A.1).

In future analysis, it would be interesting to compare the performance of the policies analyzed in this work to the $WSEPT$ policy. This type of study was undergone for the deterministic online problem in [AS02] and in [SUW05] for the offline one. Both papers showed that simple heuristics often outperform more complex algorithms when dealing with "average instances". However, this is not the case when considering particularly hard or irregular instances such as the ones that can be found in real data. Therefore, evaluating the $RSOS$ and $DSOS$ algorithms on both real-world data and specifically crafted instances while comparing their performances to those of different policies could offer valuable additional insights into their performances in practice.

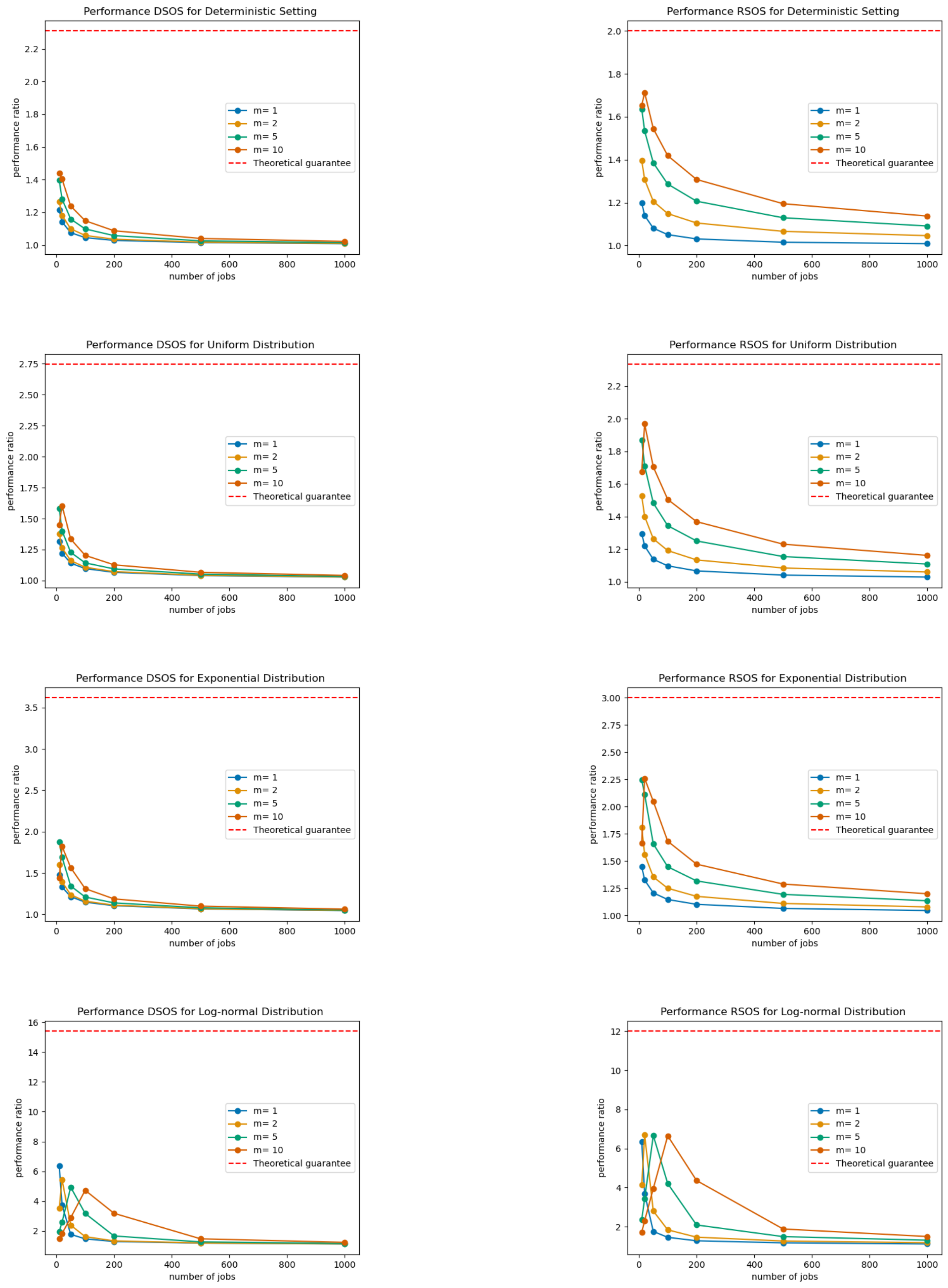


Figure 4.1 Average performance ratios over 50 instances for the $DSOS$ and $RSOS$ policies in the simulations with parameters $F = 50$, $W = 30$ and $R(n, m) = 0.6 \cdot \frac{n \cdot F}{2m} \cdot n$ for different processing times distributions.

(a) $DSOS$ policy with deterministic processing times.

m/n	10	20	50	100	200	500	1000
1	1.213 (1.355)	1.141 (1.225)	1.076 (1.128)	1.045 (1.065)	1.028 (1.040)	1.015 (1.020)	1.008 (1.011)
2	1.265 (1.467)	1.182 (1.254)	1.099 (1.132)	1.059 (1.079)	1.036 (1.047)	1.018 (1.023)	1.010 (1.012)
5	1.397 (1.496)	1.281 (1.336)	1.159 (1.183)	1.099 (1.133)	1.057 (1.079)	1.026 (1.030)	1.014 (1.017)
10	1.439 (1.524)	1.403 (1.462)	1.238 (1.286)	1.149 (1.177)	1.087 (1.104)	1.040 (1.043)	1.022 (1.025)

(b) $RSOS$ policy with deterministic processing times.

m/n	10	20	50	100	200	500	1000
1	1.200 (1.294)	1.140 (1.179)	1.081 (1.100)	1.051 (1.058)	1.031 (1.036)	1.015 (1.017)	1.009 (1.010)
2	1.396 (1.514)	1.308 (1.373)	1.206 (1.231)	1.149 (1.165)	1.105 (1.114)	1.066 (1.071)	1.046 (1.049)
5	1.637 (1.771)	1.534 (1.646)	1.385 (1.430)	1.287 (1.314)	1.207 (1.224)	1.129 (1.137)	1.091 (1.094)
10	1.654 (1.806)	1.713 (1.805)	1.545 (1.601)	1.419 (1.449)	1.308 (1.332)	1.195 (1.203)	1.137 (1.142)

(c) $DSOS$ policy with uniformly distributed processing times.

m/n	10	20	50	100	200	500	1000
1	1.318 (1.511)	1.221 (1.310)	1.141 (1.180)	1.097 (1.117)	1.066 (1.079)	1.040 (1.045)	1.028 (1.032)
2	1.380 (1.586)	1.263 (1.351)	1.161 (1.227)	1.107 (1.123)	1.072 (1.083)	1.043 (1.048)	1.029 (1.033)
5	1.583 (1.742)	1.400 (1.496)	1.227 (1.265)	1.143 (1.174)	1.094 (1.107)	1.052 (1.059)	1.033 (1.037)
10	1.451 (1.588)	1.604 (1.716)	1.335 (1.390)	1.203 (1.228)	1.127 (1.139)	1.067 (1.073)	1.041 (1.045)

(d) $RSOS$ policy with uniformly distributed processing times.

m/n	10	20	50	100	200	500	1000
1	1.295 (1.392)	1.219 (1.277)	1.137 (1.195)	1.097 (1.115)	1.066 (1.077)	1.040 (1.046)	1.028 (1.032)
2	1.525 (1.695)	1.400 (1.478)	1.263 (1.295)	1.191 (1.210)	1.133 (1.144)	1.084 (1.092)	1.059 (1.066)
5	1.870 (2.105)	1.711 (1.867)	1.482 (1.551)	1.343 (1.373)	1.250 (1.266)	1.154 (1.166)	1.108 (1.115)
10	1.674 (1.874)	1.968 (2.145)	1.704 (1.814)	1.505 (1.556)	1.368 (1.391)	1.230 (1.244)	1.161 (1.166)

(e) $DSOS$ policy with exponentially distributed processing times.

m/n	10	20	50	100	200	500	1000
1	1.476 (1.731)	1.329 (1.449)	1.213 (1.259)	1.147 (1.186)	1.104 (1.128)	1.065 (1.077)	1.046 (1.053)
2	1.601 (1.916)	1.392 (1.523)	1.236 (1.318)	1.159 (1.179)	1.110 (1.135)	1.068 (1.079)	1.047 (1.055)
5	1.877 (2.114)	1.690 (1.917)	1.342 (1.416)	1.209 (1.252)	1.138 (1.160)	1.079 (1.090)	1.051 (1.058)
10	1.440 (1.602)	1.822 (2.047)	1.564 (1.683)	1.310 (1.348)	1.186 (1.207)	1.099 (1.110)	1.062 (1.070)

(f) $RSOS$ policy with exponentially distributed processing times.

m/n	10	20	50	100	200	500	1000
1	1.449 (1.656)	1.325 (1.426)	1.208 (1.276)	1.146 (1.181)	1.102 (1.124)	1.065 (1.079)	1.045 (1.054)
2	1.810 (2.232)	1.562 (1.696)	1.354 (1.419)	1.250 (1.280)	1.175 (1.201)	1.110 (1.124)	1.078 (1.087)
5	2.244 (2.722)	2.115 (2.342)	1.658 (1.815)	1.447 (1.509)	1.317 (1.348)	1.194 (1.215)	1.135 (1.145)
10	1.667 (1.917)	2.261 (2.607)	2.048 (2.296)	1.682 (1.769)	1.471 (1.502)	1.289 (1.312)	1.199 (1.209)

(g) $DSOS$ policy with log-normally distributed processing times.

m/n	10	20	50	100	200	500	1000
1	6.381 (9.718)	3.729 (5.946)	1.765 (2.071)	1.465 (1.662)	1.284 (1.349)	1.179 (1.236)	1.122 (1.150)
2	3.498 (6.677)	5.446 (7.182)	2.383 (2.780)	1.612 (1.714)	1.327 (1.381)	1.187 (1.236)	1.128 (1.154)
5	1.944 (2.354)	2.597 (3.069)	4.929 (5.621)	3.169 (4.049)	1.657 (1.742)	1.260 (1.295)	1.155 (1.187)
10	1.478 (2.564)	1.824 (2.305)	2.885 (3.331)	4.725 (5.252)	3.179 (3.575)	1.467 (1.509)	1.228 (1.243)

(h) $RSOS$ policy with log-normally distributed processing times.

m/n	10	20	50	100	200	500	1000
1	6.341 (9.539)	3.698 (5.877)	1.761 (2.059)	1.460 (1.656)	1.283 (1.345)	1.178 (1.234)	1.122 (1.149)
2	4.142 (7.633)	6.692 (9.877)	2.807 (3.366)	1.843 (2.082)	1.467 (1.568)	1.271 (1.393)	1.184 (1.226)
5	2.366 (3.079)	3.434 (4.373)	6.665 (8.228)	4.210 (5.747)	2.092 (2.374)	1.496 (1.585)	1.312 (1.364)
10	1.721 (2.945)	2.312 (2.969)	3.956 (4.748)	6.651 (7.602)	4.365 (5.054)	1.886 (1.968)	1.501 (1.538)

Table 4.1 Performance ratios of $DSOS$ and $RSOS$ algorithms in the simulations with parameters $F = 50$, $W = 30$, $R(n,m) = 0.6 \cdot \frac{n \cdot F}{2m}$. The first number represents the average ratio over 50 instances, while the number in parentheses indicates the maximum ratio observed among the instances.

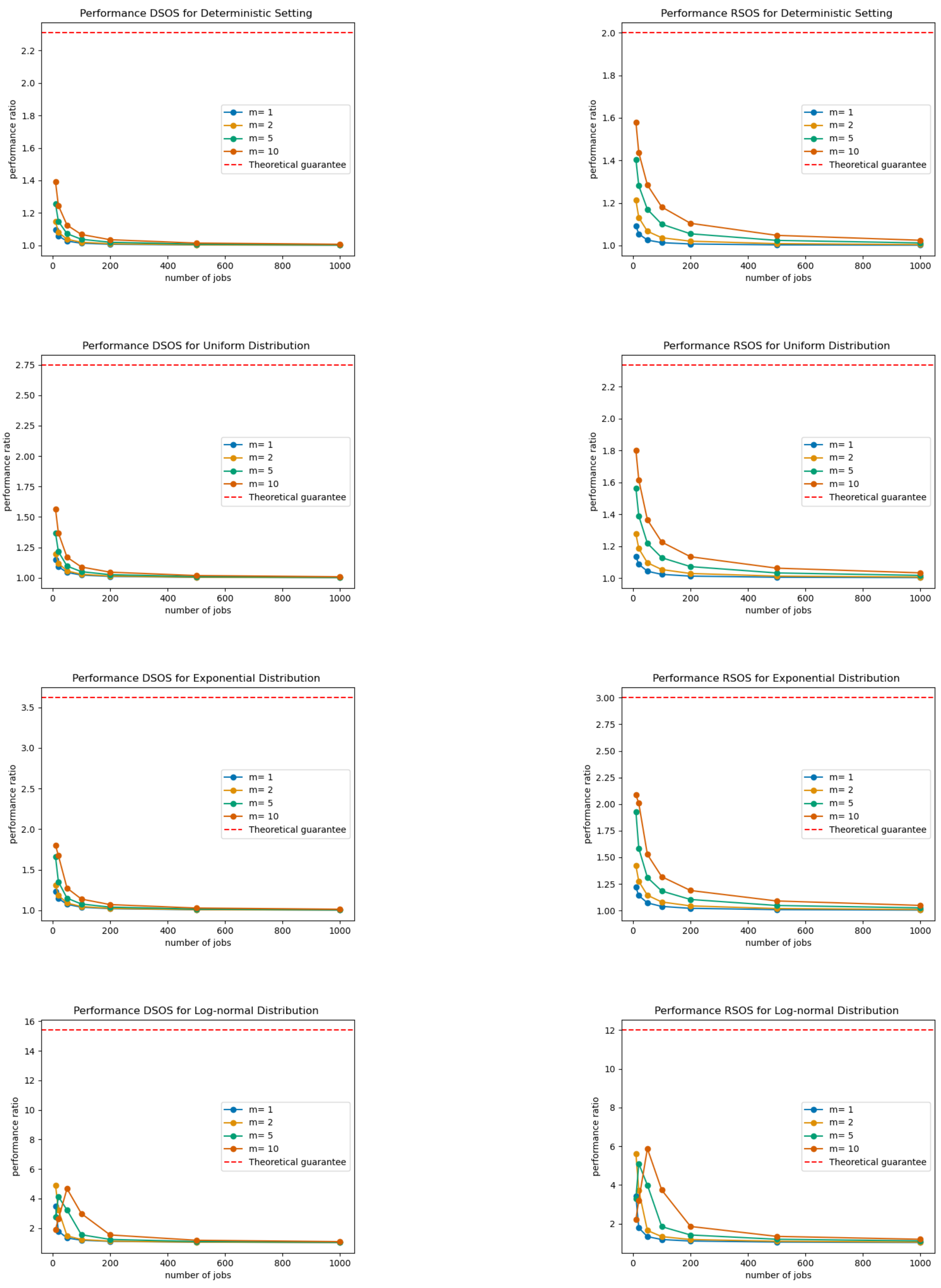


Figure 4.2 Average performance ratios over 50 instances for the $DSOS$ and $RSOS$ policies in the simulations with parameters $F = 50$, $W = 30$ and $R(n,m) = 1.5 \cdot \frac{n \cdot F}{2m} \cdot n$ for different processing times distributions.

(a) $DSOS$ policy with deterministic processing times.

m/n	10	20	50	100	200	500	1000
1	1.097 (1.224)	1.058 (1.088)	1.026 (1.038)	1.014 (1.019)	1.007 (1.009)	1.003 (1.003)	1.001 (1.002)
2	1.148 (1.245)	1.081 (1.122)	1.038 (1.051)	1.019 (1.026)	1.010 (1.012)	1.004 (1.005)	1.002 (1.002)
5	1.255 (1.425)	1.148 (1.225)	1.072 (1.101)	1.038 (1.046)	1.020 (1.024)	1.008 (1.009)	1.004 (1.004)
10	1.391 (1.486)	1.245 (1.351)	1.125 (1.154)	1.067 (1.080)	1.035 (1.041)	1.014 (1.016)	1.007 (1.008)

(b) $RSOS$ policy with deterministic processing times.

m/n	10	20	50	100	200	500	1000
1	1.092 (1.234)	1.055 (1.086)	1.025 (1.035)	1.013 (1.018)	1.007 (1.009)	1.003 (1.004)	1.001 (1.002)
2	1.214 (1.412)	1.131 (1.241)	1.067 (1.093)	1.036 (1.050)	1.020 (1.027)	1.009 (1.011)	1.004 (1.005)
5	1.404 (1.669)	1.281 (1.444)	1.170 (1.257)	1.100 (1.134)	1.055 (1.079)	1.024 (1.030)	1.012 (1.014)
10	1.581 (1.763)	1.437 (1.634)	1.285 (1.390)	1.181 (1.218)	1.104 (1.137)	1.048 (1.061)	1.024 (1.027)

(c) $DSOS$ policy with uniformly distributed processing times.

m/n	10	20	50	100	200	500	1000
1	1.147 (1.293)	1.092 (1.186)	1.045 (1.076)	1.024 (1.035)	1.012 (1.018)	1.005 (1.007)	1.003 (1.003)
2	1.197 (1.373)	1.121 (1.230)	1.058 (1.090)	1.030 (1.044)	1.015 (1.021)	1.006 (1.008)	1.003 (1.004)
5	1.367 (1.566)	1.216 (1.327)	1.097 (1.141)	1.051 (1.073)	1.027 (1.035)	1.011 (1.013)	1.006 (1.006)
10	1.565 (1.715)	1.368 (1.550)	1.170 (1.225)	1.090 (1.117)	1.047 (1.059)	1.019 (1.022)	1.010 (1.011)

(d) $RSOS$ policy with uniformly distributed processing times.

m/n	10	20	50	100	200	500	1000
1	1.135 (1.270)	1.086 (1.175)	1.042 (1.071)	1.023 (1.034)	1.012 (1.017)	1.005 (1.007)	1.002 (1.003)
2	1.277 (1.492)	1.184 (1.355)	1.096 (1.153)	1.053 (1.076)	1.028 (1.039)	1.012 (1.016)	1.006 (1.007)
5	1.563 (1.953)	1.390 (1.524)	1.218 (1.335)	1.127 (1.186)	1.071 (1.101)	1.032 (1.040)	1.016 (1.019)
10	1.801 (2.032)	1.615 (1.832)	1.365 (1.538)	1.225 (1.313)	1.133 (1.177)	1.062 (1.077)	1.032 (1.037)

(e) $DSOS$ policy with exponentially distributed processing times.

m/n	10	20	50	100	200	500	1000
1	1.231 (1.460)	1.147 (1.291)	1.074 (1.135)	1.039 (1.057)	1.021 (1.030)	1.009 (1.011)	1.004 (1.005)
2	1.310 (1.605)	1.186 (1.338)	1.090 (1.152)	1.047 (1.067)	1.025 (1.034)	1.010 (1.013)	1.005 (1.006)
5	1.660 (2.077)	1.352 (1.583)	1.151 (1.216)	1.078 (1.109)	1.041 (1.058)	1.017 (1.020)	1.008 (1.010)
10	1.801 (2.094)	1.674 (2.125)	1.275 (1.358)	1.139 (1.181)	1.072 (1.091)	1.029 (1.033)	1.015 (1.016)

(f) $RSOS$ policy with exponentially distributed processing times.

m/n	10	20	50	100	200	500	1000
1	1.218 (1.436)	1.140 (1.278)	1.071 (1.133)	1.038 (1.054)	1.020 (1.029)	1.008 (1.011)	1.004 (1.005)
2	1.422 (1.766)	1.274 (1.485)	1.143 (1.235)	1.080 (1.112)	1.043 (1.058)	1.018 (1.024)	1.009 (1.011)
5	1.928 (2.660)	1.585 (1.890)	1.310 (1.444)	1.183 (1.259)	1.104 (1.156)	1.047 (1.061)	1.025 (1.029)
10	2.086 (2.479)	2.008 (2.589)	1.527 (1.765)	1.318 (1.440)	1.188 (1.247)	1.090 (1.110)	1.048 (1.056)

(g) $DSOS$ policy with log-normally distributed processing times.

m/n	10	20	50	100	200	500	1000
1	3.460 (7.007)	1.788 (2.484)	1.330 (1.547)	1.179 (1.327)	1.100 (1.236)	1.048 (1.072)	1.025 (1.041)
2	4.901 (6.735)	3.230 (6.627)	1.470 (1.872)	1.225 (1.309)	1.113 (1.156)	1.051 (1.064)	1.026 (1.036)
5	2.767 (4.313)	4.127 (5.788)	3.223 (6.179)	1.548 (1.783)	1.232 (1.266)	1.090 (1.099)	1.044 (1.052)
10	1.908 (2.622)	2.625 (3.415)	4.666 (6.234)	2.980 (4.881)	1.535 (1.623)	1.175 (1.194)	1.082 (1.087)

(h) $RSOS$ policy with log-normally distributed processing times.

m/n	10	20	50	100	200	500	1000
1	3.419 (6.925)	1.779 (2.458)	1.327 (1.539)	1.178 (1.325)	1.099 (1.236)	1.048 (1.071)	1.024 (1.041)
2	5.623 (10.140)	3.714 (7.703)	1.646 (2.210)	1.328 (1.535)	1.178 (1.338)	1.087 (1.114)	1.047 (1.066)
5	3.303 (4.794)	5.102 (6.734)	3.992 (8.311)	1.842 (2.262)	1.413 (1.601)	1.191 (1.229)	1.106 (1.125)
10	2.205 (2.964)	3.205 (4.472)	5.877 (7.481)	3.733 (6.168)	1.854 (2.012)	1.340 (1.393)	1.189 (1.212)

Table 4.2 Performance ratios of $DSOS$ and $RSOS$ algorithms in the simulations with parameters $F = 50$, $W = 30$, $R(n,m) = 1.5 \cdot \frac{n \cdot F}{2m}$. The first number represents the average ratio over 50 instances, while the number in parentheses indicates the maximum ratio observed among the instances.

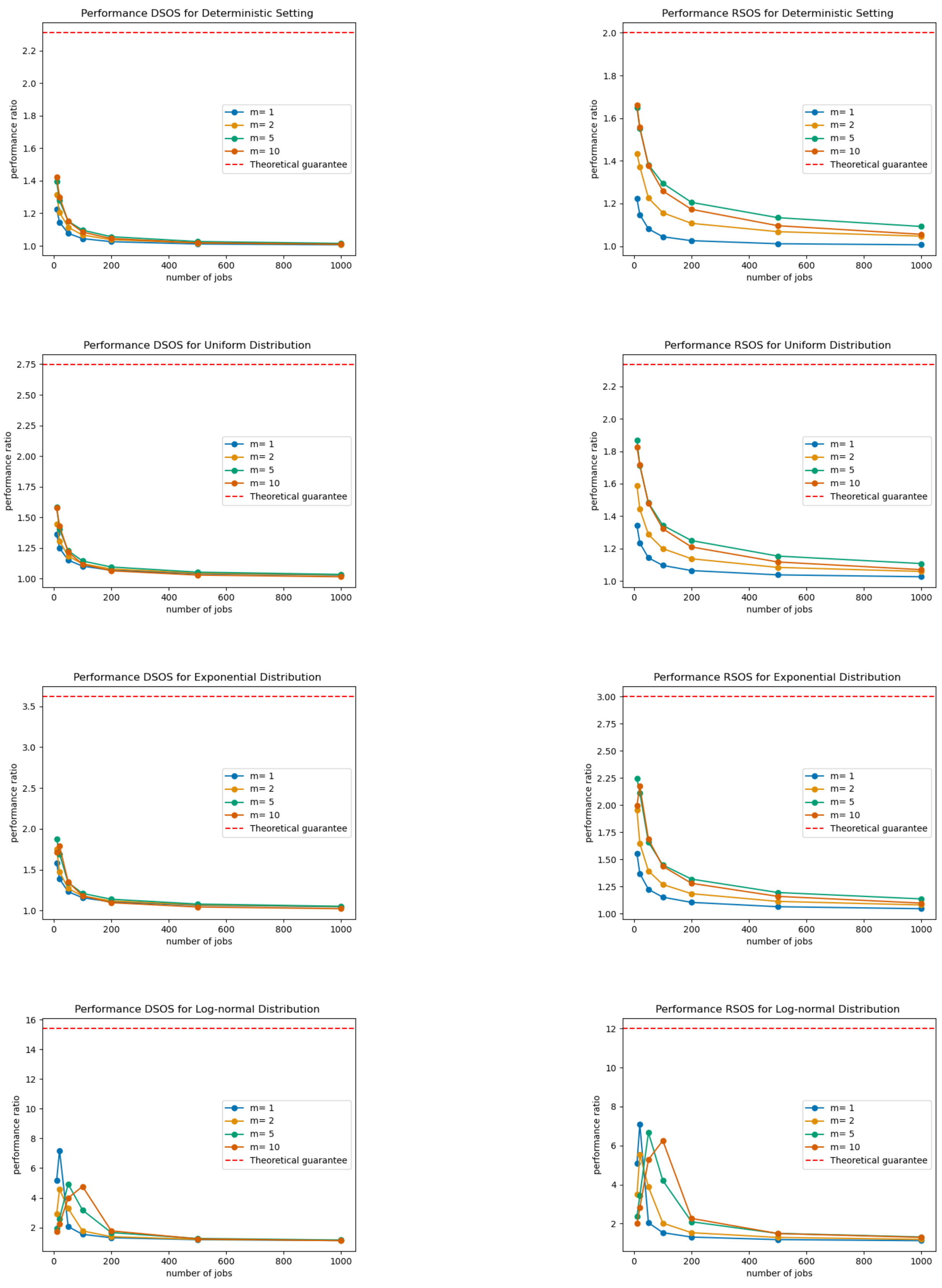


Figure 4.3 Average performance ratios over 50 instances for the $DSOS$ and $RSOS$ policies in the simulations with parameters $F = 50$, $W = 30$ and $R(n, m) = 3 \cdot n$ for different processing times distributions.

(a) $DSOS$ policy with deterministic processing times.

m/n	10	20	50	100	200	500	1000
1	1.235 (1.358)	1.146 (1.189)	1.075 (1.127)	1.044 (1.064)	1.025 (1.037)	1.011 (1.020)	1.006 (1.014)
2	1.315 (1.428)	1.211 (1.283)	1.114 (1.168)	1.069 (1.108)	1.038 (1.053)	1.018 (1.025)	1.010 (1.014)
5	1.397 (1.496)	1.281 (1.336)	1.159 (1.183)	1.099 (1.133)	1.057 (1.079)	1.026 (1.030)	1.014 (1.017)
10	1.430 (1.517)	1.297 (1.386)	1.153 (1.190)	1.084 (1.099)	1.046 (1.054)	1.019 (1.022)	1.010 (1.011)

(b) $RSOS$ policy with deterministic processing times.

m/n	10	20	50	100	200	500	1000
1	1.230 (1.319)	1.147 (1.183)	1.078 (1.114)	1.046 (1.059)	1.026 (1.031)	1.012 (1.013)	1.006 (1.008)
2	1.442 (1.537)	1.335 (1.413)	1.222 (1.252)	1.156 (1.175)	1.109 (1.117)	1.066 (1.073)	1.046 (1.048)
5	1.637 (1.771)	1.534 (1.646)	1.385 (1.430)	1.287 (1.314)	1.207 (1.224)	1.129 (1.137)	1.091 (1.094)
10	1.648 (1.860)	1.554 (1.713)	1.377 (1.483)	1.256 (1.323)	1.171 (1.228)	1.096 (1.123)	1.057 (1.076)

(c) $DSOS$ policy with uniformly distributed processing times.

m/n	10	20	50	100	200	500	1000
1	1.360 (1.450)	1.248 (1.340)	1.150 (1.187)	1.101 (1.128)	1.067 (1.081)	1.040 (1.045)	1.028 (1.031)
2	1.446 (1.559)	1.307 (1.382)	1.182 (1.227)	1.120 (1.141)	1.078 (1.091)	1.044 (1.049)	1.030 (1.034)
5	1.583 (1.742)	1.400 (1.496)	1.227 (1.265)	1.143 (1.174)	1.094 (1.107)	1.052 (1.059)	1.033 (1.037)
10	1.576 (1.698)	1.428 (1.585)	1.215 (1.297)	1.117 (1.164)	1.063 (1.077)	1.028 (1.043)	1.014 (1.017)

(d) $RSOS$ policy with uniformly distributed processing times.

m/n	10	20	50	100	200	500	1000
1	1.342 (1.447)	1.235 (1.287)	1.143 (1.177)	1.096 (1.109)	1.065 (1.074)	1.038 (1.043)	1.027 (1.030)
2	1.590 (1.764)	1.443 (1.517)	1.287 (1.320)	1.200 (1.234)	1.137 (1.150)	1.084 (1.091)	1.059 (1.063)
5	1.870 (2.105)	1.711 (1.867)	1.482 (1.551)	1.343 (1.373)	1.250 (1.266)	1.154 (1.166)	1.108 (1.115)
10	1.827 (1.995)	1.717 (1.985)	1.478 (1.645)	1.323 (1.431)	1.210 (1.255)	1.118 (1.167)	1.070 (1.088)

(e) $DSOS$ policy with exponentially distributed processing times.

m/n	10	20	50	100	200	500	1000
1	1.578 (1.710)	1.386 (1.494)	1.231 (1.292)	1.156 (1.183)	1.106 (1.129)	1.064 (1.074)	1.046 (1.055)
2	1.748 (1.977)	1.473 (1.553)	1.270 (1.314)	1.178 (1.201)	1.117 (1.140)	1.070 (1.080)	1.048 (1.057)
5	1.877 (2.114)	1.690 (1.917)	1.342 (1.416)	1.209 (1.252)	1.138 (1.160)	1.079 (1.090)	1.051 (1.058)
10	1.712 (2.057)	1.787 (2.026)	1.349 (1.490)	1.180 (1.252)	1.096 (1.116)	1.043 (1.064)	1.022 (1.027)

(f) $RSOS$ policy with exponentially distributed processing times.

m/n	10	20	50	100	200	500	1000
1	1.555 (1.817)	1.368 (1.465)	1.223 (1.272)	1.151 (1.175)	1.102 (1.123)	1.063 (1.071)	1.044 (1.052)
2	1.955 (2.274)	1.646 (1.760)	1.393 (1.448)	1.267 (1.317)	1.183 (1.209)	1.112 (1.123)	1.079 (1.088)
5	2.244 (2.722)	2.115 (2.342)	1.658 (1.815)	1.447 (1.509)	1.317 (1.348)	1.194 (1.215)	1.135 (1.145)
10	1.997 (2.553)	2.173 (2.600)	1.685 (1.928)	1.435 (1.573)	1.278 (1.333)	1.158 (1.214)	1.096 (1.116)

(g) $DSOS$ policy with log-normally distributed processing times.

m/n	10	20	50	100	200	500	1000
1	5.189 (7.628)	7.189 (9.995)	2.061 (2.440)	1.541 (1.909)	1.312 (1.412)	1.179 (1.241)	1.126 (1.156)
2	2.901 (3.701)	4.582 (5.522)	3.281 (3.859)	1.770 (1.889)	1.383 (1.446)	1.203 (1.229)	1.133 (1.163)
5	1.944 (2.354)	2.597 (3.069)	4.929 (5.621)	3.169 (4.049)	1.657 (1.742)	1.260 (1.295)	1.155 (1.187)
10	1.713 (2.163)	2.233 (3.061)	3.991 (5.168)	4.766 (6.696)	1.773 (1.937)	1.235 (1.272)	1.115 (1.130)

(h) $RSOS$ policy with log-normally distributed processing times.

m/n	10	20	50	100	200	500	1000
1	5.077 (7.203)	7.095 (10.080)	2.043 (2.422)	1.534 (1.902)	1.308 (1.409)	1.178 (1.242)	1.125 (1.154)
2	3.487 (5.140)	5.534 (7.451)	3.876 (4.702)	2.026 (2.178)	1.531 (1.647)	1.288 (1.375)	1.188 (1.232)
5	2.366 (3.079)	3.434 (4.373)	6.665 (8.228)	4.210 (5.747)	2.092 (2.374)	1.496 (1.585)	1.312 (1.364)
10	2.001 (2.577)	2.811 (3.739)	5.285 (6.844)	6.257 (9.518)	2.265 (2.547)	1.489 (1.628)	1.292 (1.334)

Table 4.3 Performance ratios of $DSOS$ and $RSOS$ algorithms in the simulations with parameters $F = 50$, $W = 30$, $R(n, m) = 3 \cdot n$. The first number represents the average ratio over 50 instances, while the number in parentheses indicates the maximum ratio observed among the instances.

5 Conclusion

In this thesis, we investigated the stochastic online scheduling problem on identical parallel machines $P|r_j, p_j \sim stoch, online|E[\sum_{j\in N} w_j C_j]$. Specifically, we thoroughly analyzed the $RSOS$ and $DSOS$ policies introduced by Schulz in the conference paper "Stochastic Online Scheduling Revisited" [Sch08]. We started by demonstrating the validity of the LP-relaxation introduced in the paper. Leveraging this relaxation, we completed and comprehensively illustrated the proofs of the performance guarantees claimed in the article for the $RSOS$ and $DSOS$ algorithms. Moreover, we successfully reduced the upper bound on the performance guarantee of the $DSOS$ policy. Notably, the guarantees proved in this thesis are the best-known ones for the problem with generic processing times' distributions. We concluded our work with a computational study on simulated data, to evaluate the performance of the policies in practice. The results of these experiments confirmed the policies' adherence to the established theoretical bounds.

In future research, drawing inspiration from [Goe+02] and [Jäg21], it would be interesting to explore alternative distributions for α within the $RSOS$ algorithm and optimize the choice of α based on the processing times' distribution in the $DSOS$ algorithm. We believe that such an investigation could lead to improved performance guarantees under restricted conditions. Additionally, given the observed asymptotic behavior of the policies in our simulations, another compelling direction for future research would be the rigorous analysis of the $RSOS$ and $DSOS$ asymptotic behavior.

In the computational study, we noticed that the experimental performance ratios for the $DSOS$ policy were considerably lower than the theoretical guarantee, indicating a potential for further improvement on the theoretical bound. However, it is essential to note that our experiments were confined to simulated data, typically consisting of average-case instances rather than worst-case scenarios. Therefore, in further work, in order to comprehensively evaluate the performance of the policies, it would be interesting to test them on deliberately crafted challenging instances as well as on real-world datasets. Furthermore, as discussed in [AS02; SUW05], comparative analyses with the $WSEPT$ policy could provide additional valuable insights on the utility of the algorithms analyzed in this work. Moreover, exploring variations of the $RSOS$ and $DSOS$ algorithms could reveal alternative approaches that may yield, depending on the problem characteristics, better outcomes in practice. For instance, our analysis of the simulations' results suggested the potential effectiveness of a policy randomly selecting α_j while using the assignment strategy of the $DSOS$ algorithms. Nonetheless, there are numerous other variations that might merit consideration.

In summary, this thesis provides a comprehensive examination of the $RSOS$ and $DSOS$ policies, offering valuable insights into their performances and suggesting potential directions for further theoretical and applied advancements.

A Appendix

A.1 Additional Simulation Results

In order to better understand the behavior of the algorithms in our simulations, we also ran the $RSOS$ and $DSOS$ in two degenerated cases. In the first scenario, all jobs are released at time 0 (Figure A.1, Table A.1), whereas in the second one, we consider a large upper bound on the release time $R(n, m) = 10 \cdot n$ (Figure A.2, Table A.2). For completeness, we also show the result of a simulation with a fixed upper bound on the release times $R(n, m) = 30$ independent of m and n (Figure A.3, Table A.3). In all cases, we observe behavior consistent with the previous discussion and interpretation of the simulations. When considering $R(n, m) = 10 \cdot n$, the effect of the parameter m reflects our previous analyses (Figure A.2). Additionally, examining the differences between the various scenarios, we notice that, as expected, the ratio decreases as the release time upper bound increases. In fact, when jobs' arrivals are extremely dilated, the jobs are processed directly on the virtual machine, and their starting time on the real machines is delayed by $\alpha_j \frac{\mu_j}{m}$, which is negligible when r_j is large.

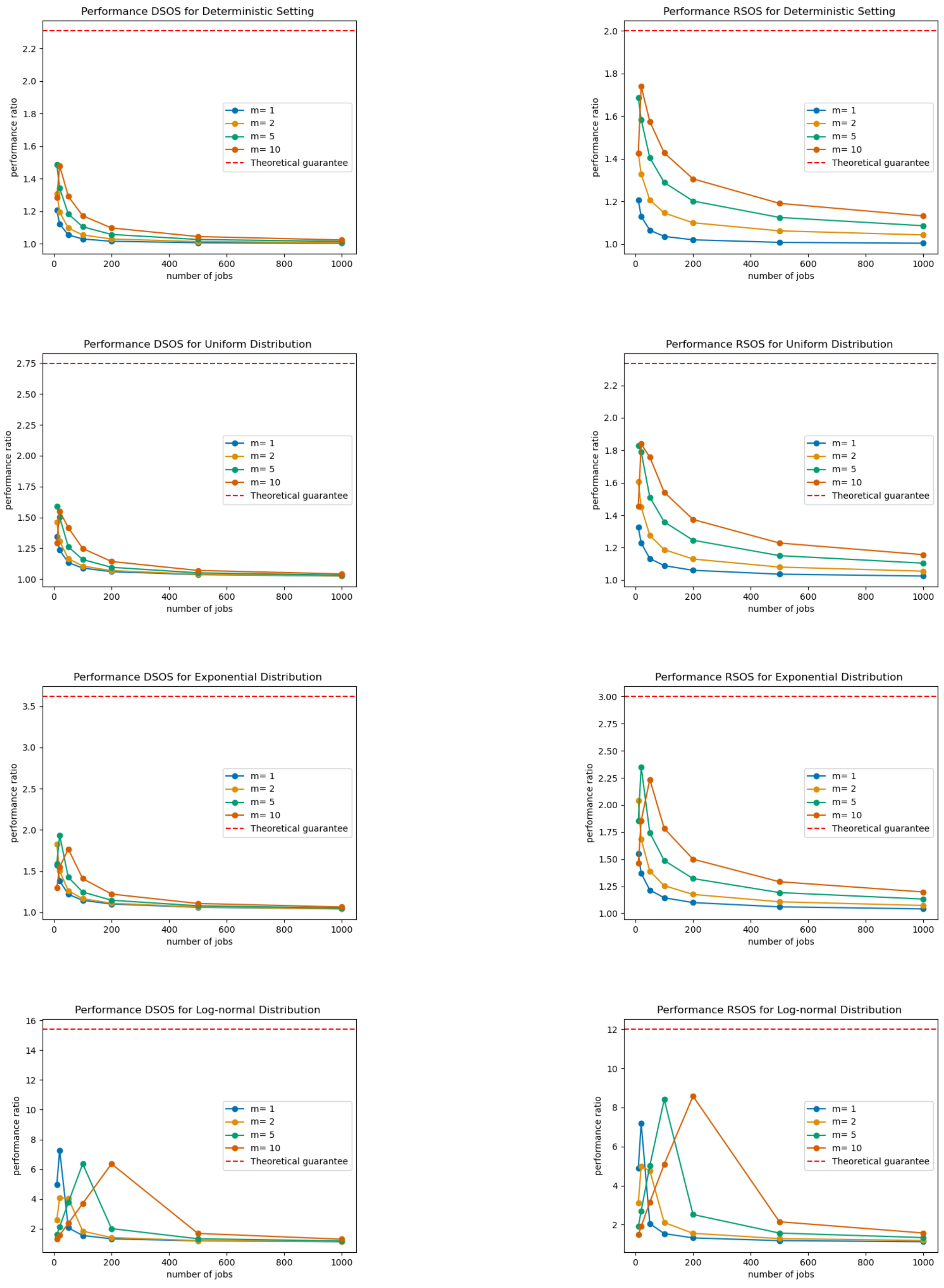


Figure A.1 Average performance ratios over 50 instances for the $DSOS$ and $RSOS$ policies in the simulations with parameters $F = 50$, $W = 30$ and $R(n, m) = 0$ for different processing times distributions.

(a) $DSOS$ policy with deterministic processing times.

m/n	10	20	50	100	200	500	1000
1	1.206 (1.247)	1.121 (1.150)	1.054 (1.062)	1.029 (1.033)	1.015 (1.016)	1.006 (1.006)	1.003 (1.003)
2	1.309 (1.369)	1.194 (1.231)	1.096 (1.108)	1.054 (1.060)	1.029 (1.032)	1.013 (1.014)	1.007 (1.007)
5	1.488 (1.527)	1.341 (1.392)	1.183 (1.201)	1.104 (1.116)	1.057 (1.061)	1.026 (1.027)	1.014 (1.014)
10	1.283 (1.348)	1.480 (1.511)	1.291 (1.317)	1.172 (1.187)	1.097 (1.104)	1.044 (1.046)	1.023 (1.024)

(b) $RSOS$ policy with deterministic processing times.

m/n	10	20	50	100	200	500	1000
1	1.206 (1.249)	1.130 (1.155)	1.065 (1.075)	1.036 (1.040)	1.020 (1.021)	1.008 (1.009)	1.004 (1.005)
2	1.427 (1.487)	1.328 (1.374)	1.207 (1.232)	1.146 (1.165)	1.100 (1.108)	1.062 (1.066)	1.043 (1.046)
5	1.687 (1.788)	1.584 (1.648)	1.405 (1.432)	1.289 (1.311)	1.202 (1.215)	1.125 (1.129)	1.086 (1.090)
10	1.427 (1.559)	1.740 (1.821)	1.575 (1.610)	1.428 (1.455)	1.306 (1.322)	1.191 (1.199)	1.132 (1.137)

(c) $DSOS$ policy with uniformly distributed processing times.

m/n	10	20	50	100	200	500	1000
1	1.342 (1.402)	1.234 (1.290)	1.134 (1.151)	1.089 (1.101)	1.060 (1.072)	1.036 (1.044)	1.025 (1.029)
2	1.463 (1.538)	1.308 (1.388)	1.166 (1.186)	1.106 (1.119)	1.069 (1.080)	1.040 (1.047)	1.026 (1.031)
5	1.588 (1.723)	1.502 (1.620)	1.261 (1.296)	1.159 (1.180)	1.096 (1.104)	1.051 (1.058)	1.032 (1.036)
10	1.294 (1.389)	1.551 (1.660)	1.417 (1.469)	1.248 (1.285)	1.143 (1.152)	1.070 (1.077)	1.042 (1.045)

(d) $RSOS$ policy with uniformly distributed processing times.

m/n	10	20	50	100	200	500	1000
1	1.325 (1.381)	1.228 (1.277)	1.132 (1.150)	1.089 (1.102)	1.060 (1.072)	1.036 (1.044)	1.025 (1.029)
2	1.606 (1.680)	1.450 (1.534)	1.275 (1.312)	1.188 (1.217)	1.130 (1.141)	1.080 (1.090)	1.055 (1.060)
5	1.830 (2.075)	1.791 (1.893)	1.511 (1.569)	1.359 (1.389)	1.246 (1.258)	1.151 (1.158)	1.104 (1.107)
10	1.455 (1.633)	1.838 (2.012)	1.757 (1.845)	1.542 (1.589)	1.373 (1.392)	1.228 (1.242)	1.156 (1.162)

(e) $DSOS$ policy with exponentially distributed processing times.

m/n	10	20	50	100	200	500	1000
1	1.575 (1.717)	1.381 (1.466)	1.218 (1.263)	1.145 (1.170)	1.100 (1.131)	1.061 (1.073)	1.041 (1.048)
2	1.828 (2.026)	1.502 (1.648)	1.261 (1.306)	1.166 (1.191)	1.110 (1.140)	1.065 (1.077)	1.043 (1.050)
5	1.590 (1.763)	1.935 (2.060)	1.425 (1.493)	1.246 (1.282)	1.146 (1.160)	1.079 (1.088)	1.051 (1.056)
10	1.294 (1.433)	1.550 (1.707)	1.768 (1.894)	1.409 (1.474)	1.221 (1.238)	1.107 (1.116)	1.065 (1.069)

(f) $RSOS$ policy with exponentially distributed processing times.

m/n	10	20	50	100	200	500	1000
1	1.550 (1.696)	1.370 (1.449)	1.215 (1.259)	1.144 (1.168)	1.099 (1.130)	1.060 (1.072)	1.041 (1.047)
2	2.042 (2.289)	1.684 (1.840)	1.390 (1.468)	1.256 (1.297)	1.175 (1.200)	1.107 (1.119)	1.073 (1.080)
5	1.853 (2.165)	2.351 (2.553)	1.744 (1.838)	1.488 (1.557)	1.321 (1.344)	1.192 (1.203)	1.132 (1.138)
10	1.462 (1.651)	1.855 (2.126)	2.236 (2.404)	1.783 (1.864)	1.500 (1.531)	1.292 (1.306)	1.197 (1.205)

(g) $DSOS$ policy with log-normally distributed processing times.

m/n	10	20	50	100	200	500	1000
1	4.965 (9.174)	7.271 (10.742)	2.055 (2.397)	1.543 (1.685)	1.320 (1.432)	1.180 (1.246)	1.123 (1.158)
2	2.592 (3.533)	4.087 (5.088)	4.050 (5.687)	1.835 (1.943)	1.403 (1.503)	1.200 (1.238)	1.131 (1.160)
5	1.604 (3.680)	2.124 (2.761)	3.758 (4.328)	6.363 (6.977)	2.004 (2.114)	1.323 (1.350)	1.176 (1.196)
10	1.308 (3.370)	1.573 (2.301)	2.356 (2.697)	3.692 (4.059)	6.362 (6.599)	1.680 (1.721)	1.293 (1.310)

(h) $RSOS$ policy with log-normally distributed processing times.

m/n	10	20	50	100	200	500	1000
1	4.876 (9.100)	7.195 (10.661)	2.047 (2.393)	1.540 (1.683)	1.318 (1.430)	1.179 (1.245)	1.123 (1.158)
2	3.103 (6.362)	4.979 (6.208)	4.761 (6.741)	2.109 (2.301)	1.554 (1.687)	1.283 (1.353)	1.185 (1.228)
5	1.899 (5.319)	2.702 (3.825)	5.014 (6.061)	8.406 (9.954)	2.517 (2.743)	1.566 (1.640)	1.335 (1.366)
10	1.479 (3.673)	1.906 (2.853)	3.147 (3.649)	5.094 (6.099)	8.580 (9.385)	2.146 (2.269)	1.570 (1.609)

Table A.1 Performance ratios of $DSOS$ and $RSOS$ algorithms in the simulations with parameters $F = 50$, $W = 30$, $R(n, m) = 0$. The first number represents the average ratio over 50 instances, while the number in parentheses indicates the maximum ratio observed among the instances.

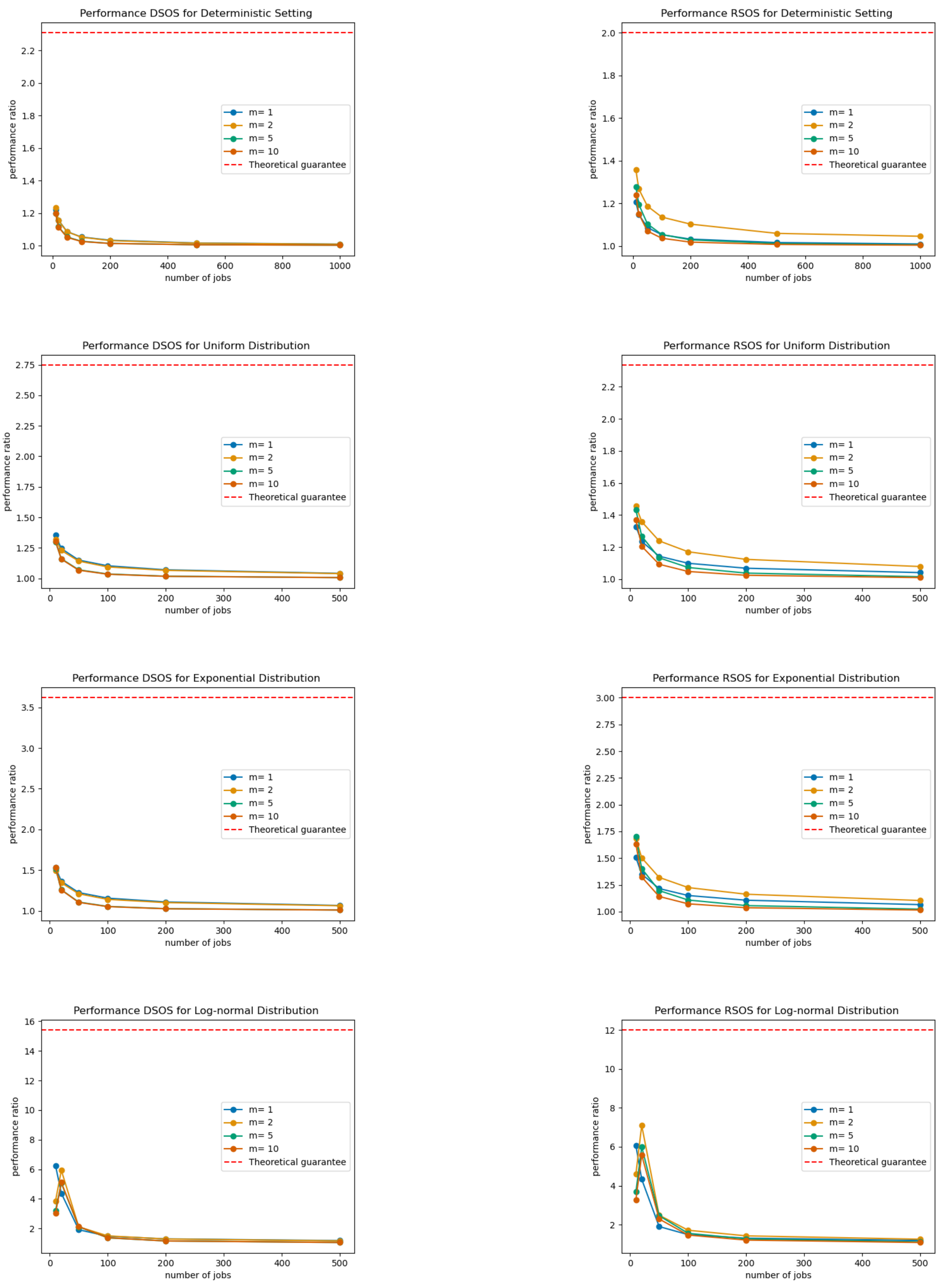


Figure A.2 Average performance ratios over 50 instances for the $DSOS$ and $RSOS$ policies in the simulations with parameters $F = 50$, $W = 30$ and $R(n, m) = 10 \cdot n$ for different processing times distributions.

(a) $DSOS$ policy with deterministic processing times.

m/n	10	20	50	100	200	500	1000
1	1.227 (1.387)	1.155 (1.263)	1.086 (1.127)	1.051 (1.080)	1.032 (1.048)	1.015 (1.021)	1.009 (1.015)
2	1.242 (1.345)	1.156 (1.223)	1.086 (1.117)	1.053 (1.070)	1.032 (1.041)	1.014 (1.017)	1.009 (1.012)
5	1.205 (1.301)	1.115 (1.152)	1.054 (1.079)	1.027 (1.036)	1.014 (1.016)	1.005 (1.006)	1.003 (1.003)
10	1.204 (1.298)	1.113 (1.161)	1.052 (1.073)	1.026 (1.035)	1.013 (1.015)	1.005 (1.006)	1.003 (1.003)

(b) $RSOS$ policy with deterministic processing times.

m/n	10	20	50	100	200	500	1000
1	1.217 (1.313)	1.154 (1.277)	1.086 (1.103)	1.052 (1.065)	1.032 (1.037)	1.016 (1.019)	1.009 (1.010)
2	1.358 (1.481)	1.277 (1.341)	1.187 (1.220)	1.136 (1.156)	1.098 (1.112)	1.061 (1.068)	1.043 (1.046)
5	1.312 (1.584)	1.197 (1.278)	1.105 (1.193)	1.055 (1.076)	1.028 (1.037)	1.011 (1.013)	1.006 (1.006)
10	1.259 (1.438)	1.148 (1.210)	1.071 (1.119)	1.036 (1.049)	1.018 (1.022)	1.007 (1.008)	1.004 (1.004)

(c) $DSOS$ policy with uniformly distributed processing times.

m/n	10	20	50	100	200	500	1000
1	1.355 (1.640)	1.245 (1.321)	1.150 (1.192)	1.104 (1.125)	1.071 (1.082)	1.042 (1.047)	1.028 (1.033)
2	1.319 (1.440)	1.229 (1.296)	1.142 (1.198)	1.094 (1.127)	1.066 (1.082)	1.040 (1.046)	1.027 (1.030)
5	1.301 (1.437)	1.161 (1.222)	1.071 (1.093)	1.037 (1.046)	1.018 (1.021)	1.007 (1.008)	1.004 (1.004)
10	1.304 (1.440)	1.158 (1.216)	1.067 (1.088)	1.035 (1.043)	1.017 (1.019)	1.007 (1.008)	1.003 (1.004)

(d) $RSOS$ policy with uniformly distributed processing times.

m/n	10	20	50	100	200	500	1000
1	1.328 (1.543)	1.235 (1.336)	1.143 (1.169)	1.099 (1.127)	1.068 (1.080)	1.041 (1.047)	1.027 (1.033)
2	1.456 (1.644)	1.358 (1.451)	1.239 (1.300)	1.171 (1.210)	1.123 (1.139)	1.079 (1.089)	1.055 (1.060)
5	1.432 (1.651)	1.268 (1.389)	1.133 (1.190)	1.073 (1.096)	1.038 (1.046)	1.015 (1.018)	1.008 (1.009)
10	1.372 (1.587)	1.205 (1.272)	1.093 (1.125)	1.048 (1.061)	1.024 (1.028)	1.010 (1.011)	1.005 (1.005)

(e) $DSOS$ policy with exponentially distributed processing times.

m/n	10	20	50	100	200	500	1000
1	1.534 (1.843)	1.364 (1.467)	1.225 (1.285)	1.156 (1.203)	1.110 (1.138)	1.067 (1.078)	1.046 (1.052)
2	1.498 (1.739)	1.345 (1.452)	1.210 (1.306)	1.141 (1.180)	1.102 (1.127)	1.063 (1.076)	1.043 (1.048)
5	1.522 (1.775)	1.259 (1.364)	1.108 (1.143)	1.055 (1.071)	1.027 (1.031)	1.011 (1.012)	1.005 (1.006)
10	1.535 (1.806)	1.257 (1.364)	1.104 (1.134)	1.052 (1.066)	1.026 (1.029)	1.010 (1.012)	1.005 (1.006)

(f) $RSOS$ policy with exponentially distributed processing times.

m/n	10	20	50	100	200	500	1000
1	1.506 (1.766)	1.350 (1.482)	1.216 (1.250)	1.150 (1.204)	1.107 (1.133)	1.065 (1.077)	1.045 (1.050)
2	1.683 (1.975)	1.503 (1.634)	1.320 (1.417)	1.225 (1.269)	1.163 (1.188)	1.104 (1.117)	1.073 (1.080)
5	1.702 (2.080)	1.403 (1.579)	1.196 (1.278)	1.108 (1.148)	1.056 (1.068)	1.023 (1.027)	1.012 (1.014)
10	1.631 (2.024)	1.324 (1.469)	1.142 (1.194)	1.073 (1.095)	1.036 (1.043)	1.015 (1.017)	1.007 (1.008)

(g) $DSOS$ policy with log-normally distributed processing times.

m/n	10	20	50	100	200	500	1000
1	6.251 (9.608)	4.383 (7.280)	1.918 (2.247)	1.483 (1.672)	1.303 (1.497)	1.187 (1.243)	1.124 (1.160)
2	3.866 (6.126)	5.933 (7.448)	2.115 (2.670)	1.505 (1.722)	1.293 (1.360)	1.170 (1.211)	1.114 (1.145)
5	3.208 (5.488)	5.073 (6.484)	2.118 (2.984)	1.373 (1.527)	1.164 (1.202)	1.060 (1.068)	1.030 (1.033)
10	3.032 (5.390)	5.111 (6.507)	2.128 (3.022)	1.381 (1.525)	1.160 (1.195)	1.059 (1.064)	1.029 (1.031)

(h) $RSOS$ policy with log-normally distributed processing times.

m/n	10	20	50	100	200	500	1000
1	6.067 (8.963)	4.331 (7.012)	1.899 (2.240)	1.481 (1.679)	1.299 (1.475)	1.185 (1.243)	1.123 (1.158)
2	4.598 (7.517)	7.103 (9.741)	2.484 (3.392)	1.708 (2.043)	1.419 (1.528)	1.247 (1.299)	1.164 (1.204)
5	3.680 (6.212)	6.001 (7.667)	2.456 (3.580)	1.551 (1.987)	1.266 (1.317)	1.110 (1.138)	1.060 (1.087)
10	3.282 (6.143)	5.580 (6.854)	2.288 (3.251)	1.461 (1.625)	1.202 (1.242)	1.079 (1.092)	1.040 (1.052)

Table A.2 Performance ratios of $DSOS$ and $RSOS$ algorithms in the simulations with parameters $F = 50$, $W = 30$, $R(n,m) = 10 \cdot n$. The first number represents the average ratio over 50 instances, while the number in parentheses indicates the maximum ratio observed among the instances.

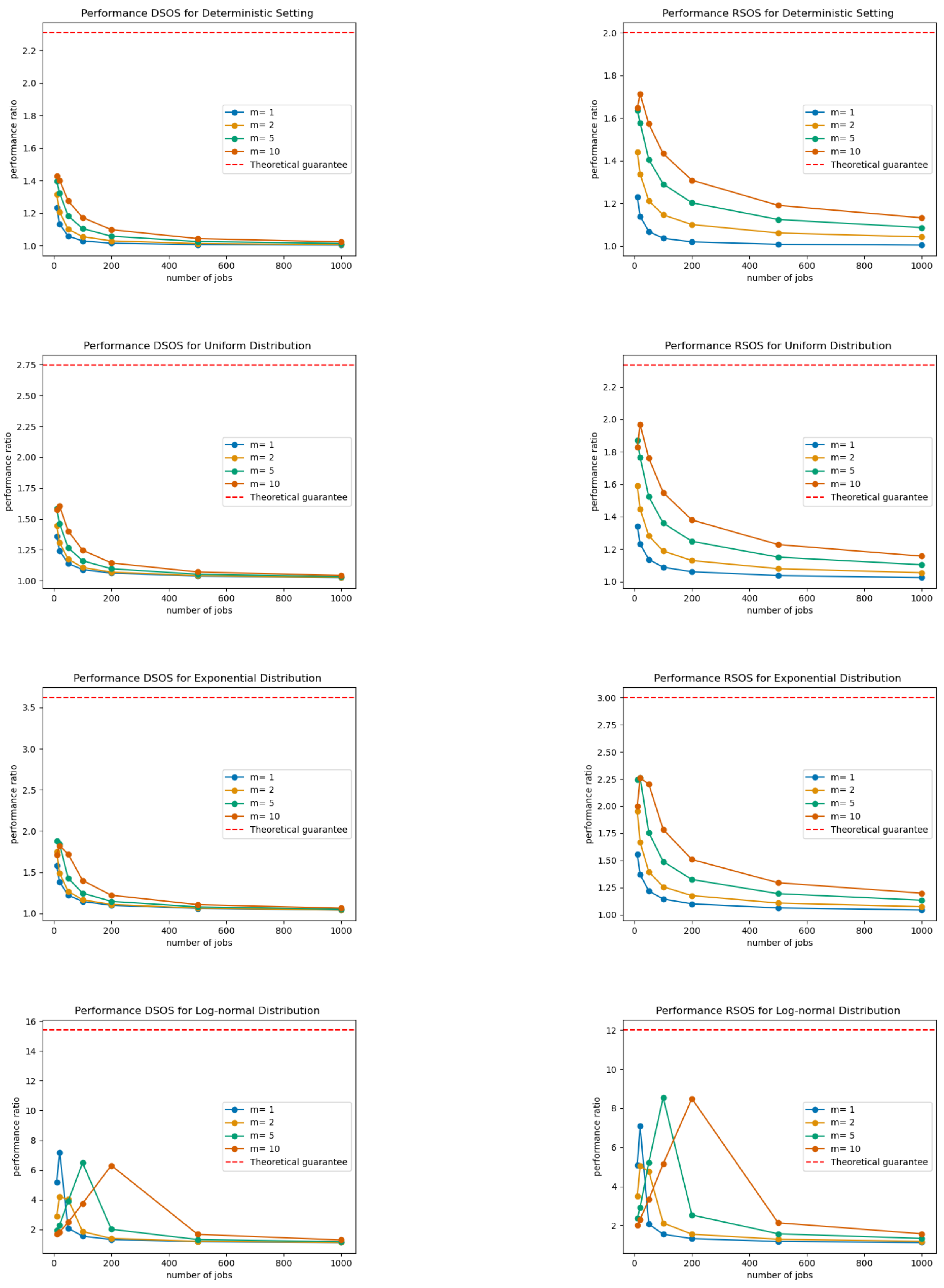


Figure A.3 Average performance ratios over 50 instances for the $DSOS$ and $RSOS$ policies in the simulations with parameters $F = 50$, $W = 30$ and $R(n, m) = 30$ for different processing times distributions.

(a) $DSOS$ policy with deterministic processing times.

m/n	10	20	50	100	200	500	1000
1	1.235 (1.358)	1.133 (1.167)	1.059 (1.072)	1.029 (1.031)	1.015 (1.016)	1.006 (1.006)	1.003 (1.003)
2	1.315 (1.428)	1.207 (1.252)	1.101 (1.125)	1.056 (1.060)	1.030 (1.032)	1.013 (1.014)	1.007 (1.007)
5	1.397 (1.496)	1.324 (1.377)	1.183 (1.207)	1.105 (1.113)	1.058 (1.064)	1.026 (1.028)	1.014 (1.014)
10	1.430 (1.517)	1.403 (1.462)	1.275 (1.297)	1.173 (1.195)	1.098 (1.105)	1.044 (1.047)	1.023 (1.024)

(b) $RSOS$ policy with deterministic processing times.

m/n	10	20	50	100	200	500	1000
1	1.230 (1.319)	1.140 (1.170)	1.068 (1.079)	1.037 (1.040)	1.020 (1.022)	1.009 (1.009)	1.004 (1.005)
2	1.442 (1.537)	1.339 (1.407)	1.213 (1.238)	1.147 (1.164)	1.101 (1.109)	1.062 (1.066)	1.043 (1.046)
5	1.637 (1.771)	1.579 (1.652)	1.406 (1.458)	1.291 (1.306)	1.203 (1.213)	1.125 (1.131)	1.087 (1.090)
10	1.648 (1.860)	1.713 (1.805)	1.573 (1.614)	1.434 (1.453)	1.309 (1.323)	1.191 (1.197)	1.133 (1.137)

(c) $DSOS$ policy with uniformly distributed processing times.

m/n	10	20	50	100	200	500	1000
1	1.360 (1.450)	1.243 (1.328)	1.139 (1.163)	1.090 (1.100)	1.061 (1.071)	1.037 (1.042)	1.024 (1.028)
2	1.446 (1.559)	1.309 (1.404)	1.174 (1.219)	1.107 (1.121)	1.069 (1.078)	1.040 (1.046)	1.026 (1.029)
5	1.583 (1.742)	1.461 (1.551)	1.267 (1.300)	1.161 (1.188)	1.096 (1.106)	1.051 (1.056)	1.032 (1.036)
10	1.576 (1.698)	1.604 (1.716)	1.402 (1.441)	1.247 (1.280)	1.143 (1.158)	1.070 (1.075)	1.042 (1.045)

(d) $RSOS$ policy with uniformly distributed processing times.

m/n	10	20	50	100	200	500	1000
1	1.342 (1.447)	1.232 (1.284)	1.136 (1.162)	1.089 (1.099)	1.061 (1.070)	1.037 (1.042)	1.024 (1.028)
2	1.590 (1.764)	1.449 (1.563)	1.282 (1.339)	1.189 (1.206)	1.129 (1.145)	1.079 (1.088)	1.055 (1.061)
5	1.870 (2.105)	1.767 (1.914)	1.523 (1.576)	1.361 (1.393)	1.248 (1.269)	1.151 (1.160)	1.104 (1.111)
10	1.827 (1.995)	1.968 (2.145)	1.761 (1.834)	1.549 (1.599)	1.380 (1.400)	1.228 (1.238)	1.157 (1.162)

(e) $DSOS$ policy with exponentially distributed processing times.

m/n	10	20	50	100	200	500	1000
1	1.578 (1.710)	1.383 (1.499)	1.222 (1.257)	1.145 (1.168)	1.099 (1.116)	1.061 (1.073)	1.042 (1.048)
2	1.748 (1.977)	1.488 (1.610)	1.269 (1.320)	1.166 (1.190)	1.110 (1.126)	1.065 (1.075)	1.043 (1.048)
5	1.877 (2.114)	1.840 (2.049)	1.428 (1.478)	1.247 (1.291)	1.147 (1.161)	1.080 (1.090)	1.051 (1.059)
10	1.712 (2.057)	1.822 (2.047)	1.721 (1.793)	1.401 (1.458)	1.221 (1.243)	1.107 (1.117)	1.064 (1.070)

(f) $RSOS$ policy with exponentially distributed processing times.

m/n	10	20	50	100	200	500	1000
1	1.555 (1.817)	1.369 (1.480)	1.218 (1.251)	1.143 (1.167)	1.098 (1.115)	1.061 (1.073)	1.042 (1.048)
2	1.955 (2.274)	1.667 (1.805)	1.395 (1.457)	1.256 (1.284)	1.174 (1.192)	1.106 (1.117)	1.073 (1.078)
5	2.244 (2.722)	2.264 (2.612)	1.757 (1.861)	1.489 (1.543)	1.323 (1.347)	1.193 (1.209)	1.131 (1.140)
10	1.997 (2.553)	2.261 (2.607)	2.203 (2.326)	1.785 (1.871)	1.507 (1.541)	1.293 (1.310)	1.197 (1.208)

(g) $DSOS$ policy with log-normally distributed processing times.

m/n	10	20	50	100	200	500	1000
1	5.189 (7.628)	7.180 (10.414)	2.078 (2.700)	1.550 (1.748)	1.325 (1.450)	1.178 (1.234)	1.125 (1.163)
2	2.901 (3.701)	4.200 (4.933)	4.024 (5.587)	1.856 (2.006)	1.403 (1.459)	1.203 (1.255)	1.131 (1.175)
5	1.944 (2.354)	2.303 (2.697)	3.894 (4.401)	6.479 (6.828)	2.014 (2.183)	1.324 (1.346)	1.174 (1.192)
10	1.713 (2.163)	1.824 (2.305)	2.483 (2.784)	3.747 (4.015)	6.303 (6.609)	1.676 (1.723)	1.292 (1.311)

(h) $RSOS$ policy with log-normally distributed processing times.

m/n	10	20	50	100	200	500	1000
1	5.077 (7.203)	7.097 (10.317)	2.068 (2.693)	1.548 (1.745)	1.323 (1.448)	1.177 (1.233)	1.125 (1.163)
2	3.487 (5.140)	5.043 (8.157)	4.765 (6.505)	2.122 (2.299)	1.550 (1.646)	1.290 (1.372)	1.185 (1.248)
5	2.366 (3.079)	2.915 (3.566)	5.224 (6.291)	8.550 (9.367)	2.534 (2.800)	1.571 (1.681)	1.333 (1.365)
10	2.001 (2.577)	2.312 (2.969)	3.325 (4.051)	5.153 (5.787)	8.492 (9.151)	2.134 (2.224)	1.570 (1.620)

Table A.3 Performance ratios of $DSOS$ and $RSOS$ algorithms in the simulations with parameters $F = 50$, $W = 30$, $R(n, m) = 30$. The first number represents the average ratio over 50 instances, while the number in parentheses indicates the maximum ratio observed among the instances.

List of Figures

List of Tables

Bibliography

[Afr+99] F. Afrati, E. Bampis, C. Chekuri, D. Karger, C. Kenyon, S. Khanna, I. Milis, M. Queyranne, M. Skutella, C. Stein, et al. “Approximation schemes for minimizing average weighted completion time with release dates”. In: *40th annual symposium on foundations of computer science (Cat. No. 99CB37039)*. IEEE. 1999, pp. 32–43.

[AS02] S. Albers and B. Schröder. “An experimental study of online scheduling algorithms”. In: *Journal of Experimental Algorithmics (JEA)* 7 (2002), p. 3.

[AP04] E. J. Anderson and C. N. Potts. “Online scheduling of a single machine to minimize total weighted completion time”. In: *Mathematics of Operations Research* 29.3 (2004), pp. 686–697.

[BD+94] S. Ben-David, A. Borodin, R. Karp, G. Tardos, and A. Wigderson. “On the power of randomization in on-line algorithms”. In: *Algorithmica* 11 (1994), pp. 2–14.

[BCJS74] J. Bruno, E. G. Coffman Jr, and R. Sethi. “Scheduling independent tasks to reduce mean finishing time”. In: *Communications of the ACM* 17.7 (1974), pp. 382–387.

[BDF81] J. Bruno, P. Downey, and G. N. Frederickson. “Sequencing tasks with exponential service times to minimize the expected flow time or makespan”. In: *Journal of the ACM (JACM)* 28.1 (1981), pp. 100–113.

[BV21] M. Buchem and T. Vredeveld. “Performance analysis of fixed assignment policies for stochastic online scheduling on uniform parallel machines”. In: *Computers & Operations Research* 125 (2021), p. 105093.

[Che+96] C. Chekuri, R. Johnson, R. Motwani, B. Natarajan, B. Ran, and M. Schlansker. “Profile-driven instruction level parallel scheduling with application to super blocks”. In: *Proceedings of the 29th Annual IEEE/ACM International Symposium on Microarchitecture. MICRO 29*. IEEE. 1996, pp. 58–67.

[Che+01] C. Chekuri, R. Motwani, B. Natarajan, and C. Stein. “Approximation techniques for average completion time scheduling”. In: *SIAM Journal on Computing* 31.1 (2001), pp. 146–166.

[Cho+06] M. C. Chou, H. Liu, M. Queyranne, and D. Simchi-Levi. “On the asymptotic optimality of a simple on-line algorithm for the stochastic single-machine weighted completion time problem and its extensions”. In: *Operations Research* 54.3 (2006), pp. 464–474.

[CW05] J. R. Correa and M. R. Wagner. “LP-Based Online Scheduling: From Single to Parallel Machines”. In: *Integer Programming and Combinatorial Optimization*. Ed. by M. Jünger and V. Kaibel. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 196–209. ISBN: 978-3-540-32102-6.

[FST94] A. Feldmann, J. Sgall, and S.-H. Teng. “Dynamic scheduling on parallel machines”. In: *Theoretical Computer Science* 130.1 (1994), pp. 49–72.

[GW93] G. Galambos and G. J. Woeginger. “An on-line scheduling heuristic with better worst-case ratio than Graham’s list scheduling”. In: *SIAM Journal on Computing* 22.2 (1993), pp. 349–355.

[Goe96] M. X. Goemans. “A supermodular relaxation for scheduling with release dates”. In: *International Conference on Integer Programming and Combinatorial Optimization*. Springer. 1996, pp. 288–300.

[Goe97] M. X. Goemans. “Improved approximation algorthims for scheduling with release dates”. In: *Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms*. 1997, pp. 591–598.

[Goe+02] M. X. Goemans, M. Queyranne, A. S. Schulz, M. Skutella, and Y. Wang. “Single machine scheduling with release dates”. In: *SIAM Journal on Discrete Mathematics* 15.2 (2002), pp. 165–192.

[Gra66] R. L. Graham. “Bounds for certain multiprocessing anomalies”. In: *Bell system technical journal* 45.9 (1966), pp. 1563–1581.

[Gra+79] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. R. Kan. “Optimization and approximation in deterministic sequencing and scheduling: a survey”. In: *Annals of discrete mathematics*. Vol. 5. Elsevier, 1979, pp. 287–326.

[GL11] M. Gu and X. Lu. “Asymptotical optimality of WSEPT for stochastic online scheduling on uniform machines”. In: *Annals of Operations Research* 191 (2011), pp. 97–113.

[Gup+21] V. Gupta, B. Moseley, M. Uetz, and Q. Xie. “Corrigendum: Greed works—online algorithms for unrelated machine stochastic scheduling”. In: *Mathematics of operations research* 46.3 (2021), pp. 1230–1234.

[Gup+20] V. Gupta, B. Moseley, M. Uetz, and Q. Xie. “Greed works—online algorithms for unrelated machine stochastic scheduling”. In: *Mathematics of operations research* 45.2 (2020), pp. 497–516.

[Hal+97] L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein. “Scheduling to minimize average completion time: Off-line and on-line approximation algorithms”. In: *Mathematics of operations research* 22.3 (1997), pp. 513–544.

[HSW96] L. A. Hall, D. B. Shmoys, and J. Wein. “Scheduling to minimize average completion time: off-line and on-line algorithms”. In: *Proceedings of the seventh annual ACM-SIAM symposium on Discrete algorithms*. 1996, pp. 142–151.

[JS18] S. Jäger and M. Skutella. “Generalizing the Kawaguchi-Kyan bound to stochastic parallel machine scheduling”. In: *arXiv preprint arXiv:1801.01105* (2018).

[Jäg21] S. J. Jäger. “Approximation in deterministic and stochastic machine scheduling”. In: (2021).

[Käm87] T. Kämpke. “On the optimality of static priority policies in stochastic scheduling on parallel machines”. In: *Journal of Applied Probability* 24.2 (1987), pp. 430–448.

[KK86] T. Kawaguchi and S. Kyan. “Worst case bound of an LRF schedule for the mean weighted flow-time problem”. In: *SIAM Journal on Computing* 15.4 (1986), pp. 1119–1129.

[Lab+84] J. Labetoulle, E. L. Lawler, J. K. Lenstra, and A. R. Kan. “Preemptive scheduling of uniform machines subject to release dates”. In: *Progress in combinatorial optimization*. Elsevier, 1984, pp. 245–261.

[LKB77] J. K. Lenstra, A. R. Kan, and P. Brucker. “Complexity of machine scheduling problems”. In: *Annals of discrete mathematics*. Vol. 1. Elsevier, 1977, pp. 343–362.

[LST90] J. K. Lenstra, D. B. Shmoys, and É. Tardos. “Approximation algorithms for scheduling unrelated parallel machines”. In: *Mathematical programming* 46 (1990), pp. 259–271.

[MT18] R. Ma and J. Tao. “An improved 2.11-competitive algorithm for online scheduling on parallel machines to minimize total weighted completion time”. In: *Journal of Industrial & Management Optimization* 14.2 (2018).

[MUV06] N. Megow, M. Uetz, and T. Vredeveld. “Models and algorithms for stochastic online scheduling”. In: *Mathematics of Operations Research* 31.3 (2006), pp. 513–525.

[MR89] R. H. Möhring and F. J. Radermacher. “The order-theoretic approach to scheduling: the stochastic case”. In: *Advances in project scheduling*. Elsevier, 1989, pp. 497–531.

[MRW85] R. H. Möhring, F. J. Radermacher, and G. Weiss. “Stochastic scheduling problems II-set strategies”. In: *Zeitschrift für Operations Research* 29 (1985), pp. 65–104.

[MRW84] R. H. Möhring, F. J. Radermacher, and G. Weiss. “Stochastic scheduling problems I—General strategies”. In: *Zeitschrift für Operations Research* 28 (1984), pp. 193–260.

[MSU99] R. H. Möhring, A. S. Schulz, and M. Uetz. “Approximation in stochastic scheduling: the power of LP-based priority policies”. In: *Journal of the ACM (JACM)* 46.6 (1999), pp. 924–942.

[MR95] R. Motwani and P. Raghavan. *Randomized algorithms*. Cambridge university press, 1995.

[PSW95] C. Phillips, C. Stein, and J. Wein. “Scheduling jobs that arrive over time”. In: *Algorithms and Data Structures: 4th International Workshop, WADS'95 Kingston, Canada, August 16–18, 1995 Proceedings 4*. Springer. 1995, pp. 86–97.

[Pin83] M. Pinedo. “Stochastic scheduling with release dates and due dates”. In: *Operations Research* 31.3 (1983), pp. 559–572.

[Pin22] M. L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer Nature, 2022.

[PST04] K. Pruhs, J. Sgall, and E. Torng. *Online scheduling.* 2004.

[QS95] M. Queyranne and A. S. Schulz. “Scheduling unit jobs with compatible release dates on parallel machines with nonstationary speeds”. In: *Integer Programming and Combinatorial Optimization*. Ed. by E. Balas and J. Clausen. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 307–320. ISBN: 978-3-540-49245-0.

[Rad84] F. J. Radermacher. “Optimale Strategien für stochastische Scheduling-Probleme”. PhD thesis. RWTH, 1984.

[Rot66] M. H. Rothkopf. “Scheduling with random service times”. In: *Management Science* 12.9 (1966), pp. 707–713.

[SUW05] M. W. Savelsbergh, R. Uma, and J. Wein. “An experimental study of LP-based approximation algorithms for scheduling problems”. In: *INFORMS Journal on Computing* 17.1 (2005), pp. 123–136.

[Sch08] A. S. Schulz. “Stochastic Online Scheduling Revisited”. In: *Combinatorial Optimization and Applications*. Ed. by B. Yang, D.-Z. Du, and C. A. Wang. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 448–457. ISBN: 978-3-540-85097-7.

[SS02] A. S. Schulz and M. Skutella. “Scheduling unrelated machines by randomized rounding”. In: *SIAM journal on discrete mathematics* 15.4 (2002), pp. 450–469.

[Sei00] S. S. Seiden. “A guessing game and randomized online algorithms”. In: *Proceedings of the thirty-second annual ACM symposium on Theory of computing*. 2000, pp. 592–601.

[SWW95] D. B. Shmoys, J. Wein, and D. P. Williamson. “Scheduling parallel machines on-line”. In: *SIAM journal on computing* 24.6 (1995), pp. 1313–1331.

[Sit10] R. Sitters. “Efficient Algorithms for Average Completion Time Scheduling”. In: *Integer Programming and Combinatorial Optimization*. Ed. by F. Eisenbrand and F. B. Shepherd. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 411–423. ISBN: 978-3-642-13036-6.

[ST85] D. D. Sleator and R. E. Tarjan. “Amortized efficiency of list update and paging rules”. In: *Communications of the ACM* 28.2 (1985), pp. 202–208.

[Smi+56] W. E. Smith et al. “Various optimizers for single-stage production”. In: *Naval Research Logistics Quarterly* 3.1-2 (1956), pp. 59–66.

[Spe94] J. Spencer. *Ten lectures on the probabilistic method*. SIAM, 1994.

[SWW19] H. Su, G. Wan, and S. Wang. “Online scheduling for outpatient services with heterogeneous patients and physicians”. In: *Journal of Combinatorial Optimization* 37 (2019), pp. 123–149.

[Uet02] M. Uetz. *Algorithms for deterministic and stochastic scheduling*. Cuvillier Verlag, 2002.

[Ves97] A. P. Vestjens. *On-line machine scheduling*. Citeseer, 1997.