On Machine Scheduling with Exponentially Distributed Processing Times

Caroline Jagtenberg
Utrecht University, Dept. of Mathematics

December 20, 2010

Master Thesis
First supervisor: Prof. dr. Marc Uetz, Twente University
Second supervisor: Prof. dr. Roberto Fernandez, Utrecht University

Abstract

We address the problem to minimize the weighted sum of completion times in nonpreemptive parallel machine scheduling. We consider the stochastic variant of this problem in which the processing times are exponentially distributed random variables. We discuss the phenomenon of deliberate idleness. For some cases we prove that optimal policies are idle time free, but the general question remains unanswered. We extensively study the WSEPT list scheduling policy, and show that its perfomance guarantee is not better than 1.243. This constitutes the first lower bound for WSEPT in this setting. In particular, this bound is larger than the (tight) bound for WSPT for deterministic scheduling, which is $\frac{1}{2}(1+\sqrt{2}) \approx 1.207$. Furthermore, we show that for any list scheduling policy, the objective value of the deterministic counterpart is an upper bound for the objective value of the stochastic instance. This result holds for arbitrarily distributed processing times. For a WSEPT policy, we show that the expected objective value for an instance with exponentially distributed processing times is at least $\frac{m}{2m-1}$ times the the objective value of the deterministic couterpart, where m denotes the number of machines. Finally, we conclude with some consideratons about the computational complexity of finding an optimal policy, and of calculating the objective value corresponding to a given policy.

Keywords *stochastic scheduling, WSEPT, exponential distribution, deliberate idleness, computational complexity*

Contents

1 Introduction

In a typical scheduling problem we are given a set of jobs that has to be processed on a number of machines. The goal is to find a feasible schedule that minimizes some sort of cost function. These problems often have nice combinatorial aspects, which makes them interesting for research. The main difference of scheduling problems compared with other combinatorial optimization problems such as graph theory, is that scheduling problems have a "time" dimension. A solution is typically specified via start times of jobs on a time axis, subject to certain combinatorial constraints. Schedules are therefore usually depicted using so-called Gannt-diagrams, where the jobs are represented with boxes on a horizontal time line (see Figure 1).

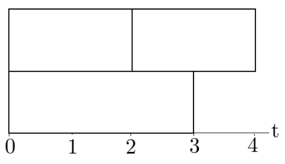

Figure 1: A Gannt diagram of two jobs with length 2 and one job with lengt 3 processed on two machines.

Throughout this thesis, unless stated otherwise, the following model is considered: We are given a set $J = \{1, ..., n\}$ of n jobs that must be processed on m identical parallel machines. Each job j has length p_j and weight w_j. Preemption is not allowed, which means that once we have placed a job on a machine, we cannot take it off until it is finished. The goal is to minimize the weighted sum of completion times. Note that the completion time of a job j depends on boh p_j and its starting time, which is determined by the schedule. Let C_j denote the completion time of the j^{th} job in a given schedule. In the classical 3-field notation for scheduling problems [2], this problem is denoted $P| |\sum w_j C_j$. The first field contains the machine environment, in this case P stands for parallel machines. The second field can contain job characteristics such as release dates or precedence constraints, but in our case it is empty. The third field contains the objective function.

Most of the literature has concentrated on deterministic scheduling. Considerably less attention was drawn to stochastic scheduling. This models a situation where job lengths are not precisely known in advance, but rather are considered a random variable. Clearly this would be a more suitable model for many applications. There was considerable research on stochastic scheduling in the 1970s and 80s. After that, there were few publications in this field until recently. In my opinion stochastic scheduling deserves more attention than it has received so far. The relatively unexploited area leaves plenty of room for new results.

The general motivation of this thesis is to compare deterministic and stochastic scheduling problems. We will highlight some of the well-known differences and discuss some results from deterministic scheduling that transfer to the stochastic case. We point out new differences and results that follow from our research, and thereby contribute to a better understanding of stochastic scheduling problems within combinatorial optimization. Using the same approach as for their deterministic couterparts often leads nowhere, as stochastic scheduling problems are conceptually different. In fact, it is not clear whether the set of feasible solutions for a stochastic scheduling problem can be reduced to a discrete set. Therefore it is in general uncertain whether stochastic scheduling problems even qualify as combinatorial optimization problems. We avoid this issue by restricting the problem to exponentially distributed processing times.

1.1 Deterministic scheduling

In deterministic scheduling it is assumed that the weight and length of all jobs is known in advance. The problem for two machines was proven to be NP-complete in the seventies. Later, the problem for m machines was proven to be NP-complete by a reduction from 3-Partition.

Theorem 1.1 [7], [1] *Deterministic scheduling is strongly NP-complete.*

Proof Clearly the problem is in NP. To prove it is NP-hard, we will reduce 3-Partition to it.

A 3-Partition instance is given by a set $S = \{s_1, s_2, ..., s_n\}$ of $n = 3m$ positive integers. The problem is to decide whether S can be partitioned into subsets of size 3 such that the sum of the numbers in each subset is equal.

The scheduling problem asks whether or not there exists a feasible schedule with $\sum w_j C_j \leq k$ for a given number k. Given a 3-Partition instance, every integer is mapped to one job in a scheduling instance, with $p_j = w_j = s_j$ for each job $j \in J$. (Clearly this transformation preserves the *strong* NP-completeness.)
Furthermore, we adopt the following notation: Let J_i be the set of jobs scheduled on machine i, for $i = 1, ..., m$. If job x is scheduled before job j, we write $x < j$. Note that the completion time of job j is the sum of the processing times of all jobs $x \leq j$ that are scheduled on the same machine as j.

Then $\sum_{j \in J} w_j C_j = \sum_{i=1}^{m} (\sum_{j \in J_i} w_j \sum_{r \in J_i, r \leq j} p_r) = \sum_{i=1}^{m} (\sum_{j \in J_i} s_j \sum_{r \in J_i, r \leq j} s_r) = \sum_{i=1}^{m} \frac{1}{2} (\sum_{j \in J_i} s_j)^2 + \frac{1}{2} \sum_{j=1}^{n} s_j^2$
If we let l_i denote the load of machine i,

$$\sum_{j \in J} w_j C_j = \frac{1}{2} \sum_{i=1}^{m} l_i^2 + \frac{1}{2} \sum_{j \in J} p_j^2$$

Note that the last term is independent of the schedule. Also we know that $\sum_{i=1}^{m} l_i = \sum_{j=1}^{n} p_j$ is constant. The objective is therefore a strictly convex function in the l_i's, which is minimized if and only if all l_i's are equal. So when choosing $k = m \frac{1}{2} b^2 + \frac{1}{2} \sum_{j=1}^{n} s_j^2$, yes-instances of 3-Partition are mapped to yes-instances of scheduling, as well as no-instances to no-instances. It is clear that the transformation is computable in polynomial time. ∎

1.2 Stochastic scheduling

In stochastic scheduling instances, the joblengths are random variables. We are only given their distribution, or at least expectation $\mathbb{E}P_j$. The realization of a processing time becomes known upon completion of the job. Therefore we cannot predict what a schedule is going to look like.

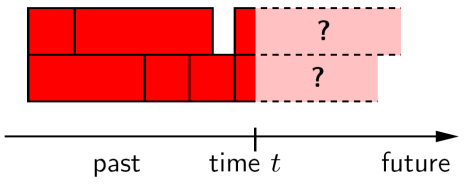

Figure 2: A Gannt diagram for stochastic scheduling.

Solutions are scheduling policies instead of schedules. For example, one can think of an order in which the jobs are placed on available machines. If at any time t that a decision has to be made, the policy is described as a linear ordering $L(t)$ on the set of jobs not started, it is called a list schedule. If the ordering at any decision time t is equal to the ordering at time 0 (restricted to the set of jobs not started before t, of course), the rule is said to be static. However, a policy may use information from realizations in the past to decide what jobs to schedule next. If $L(t)$ varies with t, the rule is said to be dynamic. Note that all list scheduling rules are greedy. Policies in general do not need to be greedy, they can contain deliberate idle time [8].

In general it is not possible to find a policy that minimizes the objective function for all realizations of processing times. We therefore seek a policy that minimizes the objective function in *expectation*. That is, we want to find a policy that attains the minimum of $\mathbb{E}(\sum w_j C_j) = \sum w_j \mathbb{E}(C_j)$ over all policies.

Let P_j denote the random variable corresponding to the processing time of job j. In this thesis we focus on processing times governed by independent, exponentially distributed random variables. That is to say, each job comes with a parameter $\lambda_j > 0$, and

$$\mathrm{P}[P_j > t] = e^{-\lambda_j t}\,.$$

We denote this by writing $P_j \sim \exp(\lambda_j)$. Exponentially distributed processing times somehow represent the cream of stochastic scheduling, in particular when juxtaposing stochastic and deterministic scheduling: The exponential distribution is characterized by the memory-less property, that is,

$$\mathrm{P}\left[P_j > s + t \;|P_j > s\right] = \mathrm{P}\left[P_j > t\right]\,.$$

So for any non-finished job it is irrelevant how much processing it has already received. This is obviously a decisive difference to deterministic scheduling models, and puts stochastic scheduling apart. Next to that, the model with exponentially distributed processing times is attractive because it makes the stochastic model analytically tractable.

Since the exponential distribution is memory-less, we do not gain extra information unless a job finishes. Therefore optimal policies belong to the so-called class of *set strategies* [8]: they only need to make decisions at $t = 0$ and at job completion times. Also, the information on the set of jobs already finished and the set of currently running jobs is sufficient to make optimal decisions. In particular, set strategies do not depend on t.

Attempting to solve stochastic scheduling problems by considering the jobs as if they were deterministic with length $\mathbb{E}P_i$ generally results in an overestimation of the objective value. For example, suppose we (list)schedule three jobs in the order $1, 2, 3$ on two machines. The third job can then start at the minimum of the first two completion times, which is expected at $\frac{1}{\lambda_1+\lambda_2}$. This is clearly earlier than the minimum of $\frac{1}{\lambda_1}$ and $\frac{1}{\lambda_2}$. For the precise result, see Corollary 3.9

The main result of this thesis is a lower bound on the approximation ratio for the WSEPT list scheduling policy (weighted shorted expected processing time first). This result will be published [4], and is found in Section 3.4. The bound is in fact larger than the tight bound for the deterministic variant; WSPT.

We also adress the complexity of stochastic scheduling. Since deterministic scheduling is a special case of stochastic scheduling, finding an optimal policy for stochastic scheduling in general is np-hard. However, it is an open question whether this still holds when we restrict the problem to exponentially distributed processing times. Also, the complexity of calculating the objective value corresponding to a policy is unknown.

The thesis is organized as follows. In Section 2 we discuss the notion of deliberate idleness and whether it is ever needed in an optimal policy. Section 3 is about the performance of the WSEPT heuristic. In Section 4 we try to answer how difficult it is to find an optimal policy, or to calculate the objective value corresponding to a policy. Each section starts with relevant previous work, followed by some new results or conjectures. We end with our conclusions in Section 5 and a discussion of the research in Section 6.

2 Deliberate idleness

Definition Deliberate idleness occurs when one or more machines are left empty while there is at least one unscheduled job available for processing.

Intuitively, the reason for an optimal policy to contain deliberate idleness is the following. If we had more information about the processing times of currently running jobs, we might be able to make a better decision on which job to schedule next. It could then be benifical to wait a small amount of time, in order to gain information about the realizations. An optimal scheduling policy might therefore depend on this information.

This idea is used in the following example.

Example [16] There are 5 jobs to be scheduled on $m = 2$ machines, with the objective to minimize $\mathbb{E}(\sum w_j C_j)$.

$$
\begin{aligned}
&w_1 = w_2 = 1, && p_1 = p_2 = 1 \\
&w_3 = w_4 = k, k \geq 5, k \in \mathbb{N}, && P_3 = P_4 = \begin{cases} 1 & w.p.\ 1 - 1/k^3 \\ 2k^4 - k^3 + 1 & w.p.\ 1/k^3 \end{cases} \\
&w_5 = 2k^3 && P_5 = \begin{cases} 1/k^5 & w.p.\ 1/2 \\ k^5 & w.p.\ 1/2 \end{cases}
\end{aligned}
$$

Then $\mathbb{E}P_3 = \mathbb{E}P_4 = 2k$. *An optimal policy in this case needs idle time.*

Proof First, consider the situation that jobs $1-4$ are to be scheduled on a single machine. Then a WSEPT policy is optimal [12], say in the order 1,2,3,4. The expected objective value is $6k^2 + 4k + 3$. If one of jobs 3 or 4 is scheduled first, this leads to an expected objective value of at least $6k^2 + 6k + 3$; a difference of $2k$. If jobs $1-4$ are to be scheduled on two machines, a case analysis of list scheduling in the order $3, 4, 1, 2$ shows that the expected objective value is $4k^2 + 4 + o(1)$. A tedious case analysis shows that any policy which does not start with jobs 3 and 4 leads to an expected objective value of at least $4k^2 + k + 3 - o(1)$; a difference of $k - 1 - o(1)$. In other words, the optimal scheduling policy depends on the number of available machines.

Now consider the given instance and a policy that schedules job 5 at time 0, leaves a machine ide until time $1/k^5$, and then, depending on the realization of the processing time of job 5, optimally schedules the remaining 4 jobs. The maximal total processing time of the remaining 4 jobs is $4k^4 - 2k^3 + 4$. Hence, if job 5 turns out to be long (time k^5), irrespective of the realizations of the processing times of the remaining jobs, there is only one machine available for them, since $k \geq 5$. If job 5 turns out to be short (time $1/k^5$), there are to machines available. With the above observations, the expected weighted sum of completion times of this policy is $k^8 + 5k^2 + 2k + 7/2 + o(1)$.

Next, observe that any idle time free policy that schedules job 5 not at time 0, can only schedule it at time 1 or later. This yields an expected objective value of at least $k^8 + 2k^3 - o(1)$. Since $k \geq 5$ this cannot be optimal. So consider a scheduling policy that greedily starts job 5 and some other job at time 0. Then, with probability 1/2 it turns out at time $1/k^5$ that this decision was not the optimal one. According to the above argumentation, this yields an expected weighted sum of completion times of at least $k^8 + 5k^2 + (5/2)k + 3 - o(1)$. In other words, any optimal policy is not idle time free, since it must leave one machine idle until time $1/k^5$ to make the right decision. ∎

Of course, the question arises whether deliberate idleness is ever necessary for a problem with exponentially distributed processing times. For this case decision times of an optimal policy only need to be at 0 or at completion times of jobs. The presence of idleness would therefore imply a machine is left empty for at least one joblength. From now on we are dealing with jobs whose processing times are exponential random variables.

For a few cases we are able to show that optimal policies are idle time free. They are described in the rest of this section.

Theorem 2.1 [14] *When $m = 2$ an optimal policy is idle time free.*

Intuitively, the reason for idle time being suboptimal is the following. With one machine idle, there is only one job processed at a time. The next decision will have to be made upon completion of this job. The set of finished jobs, as well as the set of unscheduled jobs at that time, are already known when we start the idle time. Therefore the only information we gain by waiting, is the completion time of the currently running job. Since a set strategy is optimal, we do not need this information to make an optimal decision.

Proof The proof consists of two steps: in the first step we assume that a policy leaves one machine empty at time 0. We replace the policy by one that does not leave a machine empty before the first job completion, and show it performs better or equal. The theorem then follows by an inductive argument.

Step 1 Assume that policy Π leaves one machine empty at time 0. W.l.o.g. let k be the job that starts at $t = 0$. The next decision is made at the completion of k. As there is no new information gained by then, the decision on what job to schedule next will be the same as it would have been at time 0. As it is clearly suboptimal for a policy to leave all machines idle, at least one job has to be started. Let i be that job. Whether or not Π leaves the second machine empty again is does not matter, we will write '*' for this position. We write '**' for what is scheduled at the completion time of i.
We define a policy Π^1 by:

- Start jobs k and i at time 0.
- If k finishes first, continue as in Π (with '*')
- If i finishes first, leave one machine empty until k finishes. As soon as k has finished, continue as in Π (with '$* \cup **$')

Let J be the set of all jobs. Then

$$\mathbb{E}(\sum_{j\in J} w_j C_j^{\Pi}) = \frac{\sum_{j\in J} w_j}{\lambda_k} + \varrho(J - \{k\})$$

where $\varrho(I)$ is the objective value of an optimal policy for job set I.
Assuming Π was optimal, it is also optimal for the subproblem $J - \{k\}$. For Π^1 then holds:

$$\mathbb{E}(\sum_{j\in J} w_j C_j^{\Pi^1}) = \frac{\sum_{j\in J} w_j}{\lambda_k + \lambda_i} + \frac{\lambda_k}{\lambda_k + \lambda_i}\varrho(J - \{k\}) + \frac{\lambda_i}{\lambda_k + \lambda_i}\left(\frac{\sum_{j\in J-\{i\}} w_j}{\lambda_k} + \varrho(J - \{k, i\})\right)$$

Which yields

$$\mathbb{E}(\sum_{j\in J} w_j C_j^{\Pi^1}) - \mathbb{E}(\sum_{j\in J} w_j C_j^{\Pi}) =$$

$$\frac{1}{\lambda_k+\lambda_i}\Bigg(\sum_{j\in J} w_j \;+\; \lambda_k\ \varrho(J-\{k\}) + \frac{\lambda_i}{\lambda_k}\sum_{j\in J-\{i\}} w_j \;+\; \lambda_i\ \varrho(J-\{k,i\})$$

$$-\frac{\lambda_k+\lambda_i}{\lambda_k}\sum_{j\in J} w_j - (\lambda_k+\lambda_i)\varrho(J-\{k\})\Bigg) =$$

$$\frac{1}{\lambda_k+\lambda_i}\Bigg(\frac{\lambda_i}{\lambda_k}\ (\sum_{j\in J-\{i\}} w_j - \sum_{j\in J} w_j) + \lambda_i\ (\varrho(J-\{k,i\})\ -\ \varrho(J-\{k\}))\Bigg) \le$$

$$\frac{1}{\lambda_k+\lambda_i}\Bigg(-\frac{\lambda_i}{\lambda_k}\ w_i - \lambda_i(w_i P_i)\Bigg) \le 0$$

Therefore Π^1 is better or equal than Π.

Step 2 Π^1 is optimal and leaves no machine idle at time 0. Starting from Π^1, we can use a construction as in step 1, to obtain a policy Π^2 that also after the first job completion leaves no machine idle. In case Π^1 leaves a machine idle, we simply repeat the above construction. This way we obtain a finite series of policies with

$$\mathbb{E}(\sum w_j C_j^{\Pi}) \ge \mathbb{E}(\sum w_j C_j^{\Pi^1}) \ge \mathbb{E}(\sum w_j C_j^{\Pi^2}) \ge ... \ge \mathbb{E}(\sum w_j C_j^{\Pi^j})$$

Where finally Π^j is an idle time free policy. This gives us a solution without deliberate idleness that is at least as good as Π. ∎

The same does not hold for $m > 2$. In that case we can leave one machine idle while two or more jobs are running. The completion of a job then gives us information on the set of finished jobs, that we did not have before. This information might be used in an optimal policy.

Lemma 2.2 *For any time t holds; if at the next completion time there are enough machines available to process all unscheduled jobs, idleness at time t is suboptimal.*

Proof Assume there exists an optimal policy Π that leaves at least one machine deliberately idle at time t. An optimal policy must schedule all remaining jobs at the next completion time, since there are enough machines available by then. Scheduling any one of these remaining jobs at time t would decrease the objective value. To see this, say that job k is one of the jobs scheduled at the next completion time t', and note that $t' > t$. More precisely, $t' = t + (\sum_{j\in J(t)} \lambda_j)^{-1}$, where $J(t)$ is the set of jobs that is in process after time t. When scheduling k at time t instead, the expected completion time of k itself decreases by $t' - t > 0$. Moreover, the expected next completion time now becomes $t'' = t + (\sum_{j\in J(t)\cup\{k\}} \lambda_j)^{-1}$. Clearly $t < t'' < t'$, hence our new scheduling policy is expected to start all remaining jobs (including k) earlier than Π. This is an obvious contradiction to Π being optimal.

Corollary 2.3 *A policy with deliberate idle time is suboptimal when $n \le m+1$.*

Theorem 2.4 *When $m = 3$ and $n = 5$, an optimal policy is idle time free.*

Proof Inserting idleness when only one job is left unscheduled is clearly suboptimal, as there is only one decision possible. Starting an idle period when there are only two jobs left unscheduled is also suboptimal, by means of Lemma 2.2.
Therefore we can restrict ourselves to policies where the only idle time starts at $t = 0$. Assuming

such a policy is optimal, it starts two jobs at the first completion time. Which two jobs these are, may depend on which job finished first. (If not, the proof is trivial.) W.l.o.g., assume jobs 1 and 2 are started at time 0. If job 1 ends first, w.l.o.g. say the optimal policy continues with jobs $3 \cup 4$. If job 2 ends first, say the optimal policy continues with jobs $4 \cup 5$. Since $(3 \cup 4) \cap (4 \cup 5) = 4$, a policy that schedules job 4 at time 0 is better than any policy that leaves a machine idle. ∎

Note that this proof does not extend to larger cases, as the intersection of the two job sets can then be empty.

This implies that when looking for an instance that requires deliberate idle time, the smallest option is $m = 3$ and $n = 6$. Our attempt to find such an instance has been unsuccessful.

Open question 2.5 *When scheduling exponentially distributed jobs for* $P| \,|\mathbb{E}\sum w_j C_j$*, do optimal policies ever require idleness?*

3 The WSEPT list scheduling policy

As long as the question of deliberate idleness is unanswered, we won't be able to find optimal policies for all instances. Instead, we consider a well known list scheduling policy:

Definition WSEPT greedily schedules all jobs in order of decreasing $\frac{w_i}{\mathbb{E}P_i}$, i.e. weighted shortest expected processing time first.

We will now discuss some of the previous work on WSEPT and add some new results. We will focus on comparing WSEPT with WSPT, the variant for deterministic scheduling. Some of our proofs require properties of the exponential distribution, others are valid for stochastic scheduling in general. In order to quantify the performance of WSEPT, we need to define approximation for stochastic problems:

Definition A policy is called an α-approximation if its expected performance is within a factor α from the expected objective value of an optimal policy.

For exponentially distributed processing times, WSEPT turns out to be a $2 - \frac{1}{m}$-approximation. This bound is determined by using an LP relaxation, as described in Theorem 3.7. In some cases it can even be proved that WSEPT is optimal.

3.1 Special cases where WSEPT is optimal

Theorem 3.1 [13] *For deterministic scheduling, WSPT (weighted shortest processing time first) is optimal for* $1| \,|\sum w_j C_j$

The proof is based on a simple job exchange argument.

Theorem 3.2 [12] *For general distributions of processing times, WSEPT is optimal for* $1| \,|\mathbb{E}\sum w_j C_j$

Note that the processing times are stochastically independent and independent of the start times of the jobs. Therefore the proof of Theorem 3.1 transfers directly to the stochastic case.

Theorem 3.3 [5] *Consider jobs with* $P_j \sim \exp(\lambda_j)$*. If there exists an ordering of the jobs, such that:*

$$\left.\begin{array}{rr} & \lambda_1 w_1 \geq \ldots \geq \lambda_n w_n \\ \textit{and} & w_1 \geq \ldots \geq w_n \end{array}\right\} \tag{1}$$

then scheduling the jobs greedily in the order $1, 2, \ldots, n$ *is optimal. (Note that this is a WSEPT policy.)*

Corollary 3.4 *When all weights are equal, SEPT is optimal for scheduling jobs with exponentially distributed processing times.*

3.2 Preliminaries for analyzing the WSEPT heuristic

One might try to use Theorem 3.3 to construct an approximation algorithm that performs better than WSEPT, by using the described optimal policy for cases with equal ratios $\frac{w_i}{\mathbb{E}P_i}$. However, we will now show that such an algorithm will in general not perform better than WSEPT.
Let $z^{OPT}(J)$ denote the expected weighted sum of completion times of an optimal policy for job set J. Let $z^{WSEPT}(J)$ denote the maximum expected objective value over all WSEPT policies on job set J. The following theorem was stated in [6] for deterministic scheduling. We show that with a slightly altered proof it also holds for stochastic scheduling.

Theorem 3.5 *Let α be the ratio $z^{WSEPT}(I)/z^{OPT}(I)$ for a problem instance I in which tasks have the same $w_j/\mathbb{E}P_j$. Then there existst another instance J which satisfies $z^{WSEPT}(J)/z^{OPT}(J) > \alpha - \delta$ for any $\delta > 0$, and in which all tasks have distinct ratios $w_j/\mathbb{E}P_j$. This holds for instances with generally distribute processing times.*

Proof Let a be a constant such that $\forall j \in I: \ w_j = r\ \mathbb{E}P_j$. W.l.o.g. assume that in (one of) the worst WSEPT policies (one corresponding to $z^{WSEPT}(I)$), the jobs are started in the order $1, 2, ..., n$. Now we create a new instance J, with processing times distributed as in I. However, the jobs in J have slightly reduced weights, in such a way that scheduling jobs in the order $1, 2, ..., n$ is the only possible WSEPT policy:
$\forall j \in J \ \ w_j = (1 - \varepsilon_j)\ r\ \mathbb{E}P_j$, where $0 < \varepsilon_1 < \varepsilon_2 < ... < \varepsilon_n$ and $\varepsilon_n < \delta/\alpha$. Note that since all weights are reduced, $z^{OPT}(J) < z^{OPT}(I)$. Then

$$\frac{z^{WSEPT}(J)}{z^{OPT}(J)} > \frac{z^{WSEPT}(J)}{z^{OPT}(I)} \geq \frac{(1-\varepsilon_n) z^{WSEPT}(I)}{z^{OPT}(I)} > \alpha - \delta$$

∎

The following theorem was originally stated for deterministic scheduling. It implies that when looking for an upper bound on the performance of WSEPT, it is sufficient to restrict to instances with equal ratios $\frac{w_j}{\mathbb{E}P_j}$. Since $\mathbb{E}(\sum w_j C_j) = \sum w_j \mathbb{E}(C_j)$ the proof transfers directly to the stochastic case.

Theorem 3.6 [6] *If for all instances J with equal ratios $\frac{w_j}{\mathbb{E}P_j}$ holds*

$$z^{WSEPT}(J)\ /\ z^{OPT}(J) \leq \alpha \qquad \textit{for some constant } \alpha \qquad (2)$$

then (2) holds for any instance J. This holds for instances with generally distributed processing times.

Proof The proof is based on an induction on the number of distinct values of $\frac{w_j}{\mathbb{E}P_j}$. The following notation is used.
$r_j = w_j\ /\ \mathbb{E}P_j$
v: the number of distinct values of ratios r_j
q: the number of jobs whose ratios are equal to r_1 J': a set of n jobs whose processing times are distributed as the ones in J, and whose weights are given by $w'_j = \begin{cases} \varepsilon w_j & for \ \ 1 \leq j \leq q \\ w_j & for \ \ q+1 \leq j \leq n \end{cases}$
where $\varepsilon = r_{q+1}/r_q < 1$
$r'_j = w'_j\ /\ \mathbb{E}P_j$
W: a WSEPT schedule for J
OPT : an optimal schedule for J
W': a schedule for J' in which job j' is put in the position of job j determined by **W**, respectively

OPT': a schedule for J' in which job j' is put in the position of job j determined by OPT, respectively
x: the part of the expected objective value for $\mathbf{W}$ comprised of costs of the first q tasks
y: the part of the expected objective value for $\mathbf{W}$ comprised of costs of the last $n-q$ tasks
x^*: the part of the expected objective value for OPT comprised of costs of the first q tasks
y^*: the part of the expected objective value for OPT comprised of costs of the last $n-q$ tasks

By construction, the number of distinct ratios r'_j is $v-1$. Therefore in what follows, we shall prove that if 2 holds for J', then it also holds for J.

From the definition of x, y, x^* and y^* we obtain

$$z^{\mathbf{W}} = x + y \qquad \text{and} \qquad z^{OPT} = x^* + y^* \tag{3}$$

$$z^{\mathbf{W}'} = \varepsilon x + y \qquad \text{and} \qquad z^{OPT'} = \varepsilon x^* + y^* \tag{4}$$

Note that if $r'_i > r'_j$, then $r_i > r_j$, so job i precedes j in $\mathbf{W}$, and it follows that i' precedes j' in $\mathbf{W}$'. So $\mathbf{W}$' is a WSEPT policy for J'. Therefore applying the inductive hypothesis (2);

$$z^{\mathbf{W}'} \leq \alpha \; z^{OPT'} \tag{5}$$

Furthermore the first q jobs of J are earliest processed in $\mathbf{W}$, and it follows from our assumption that

$$x \leq \alpha \; x^* \tag{6}$$

If $y \leq \alpha \; y^*$ then by (3) and (6) the proof is completed. Otherwise, combining $\varepsilon < 1$ and (6) yields

$$(1-\varepsilon)x^* y > (1-\varepsilon)xy^* \tag{7}$$

Therefore by (3) and (4),

$$\begin{aligned} z^{\mathbf{W}'} \; z^{OPT} &= (\varepsilon x + y)(x^* + y^*) = \\ &\varepsilon x x^* + yy^* + \varepsilon x y^* + x^* y > (\text{ combined with } \varepsilon < 1 \text{ and (7) }) \\ &\varepsilon x x^* + yy^* + xy^* + \varepsilon x^* y = \\ &\qquad (\varepsilon x^* + y^*)(x + y) = z^{OPT'} \; z^{\mathbf{W}} \end{aligned}$$

This, together with (5) yields $z^{\mathbf{W}} \leq \alpha \; z^{OPT}$. Note that this holds for any WSEPT policy $\mathbf{W}$. Therefore $z^{WSEPT}(J) \leq \alpha \; z^{OPT}(J)$. ∎

This implies that when looking for an upper bound on α for WSEPT, the search can be limited to instances with equal ratios $\frac{w_j}{\mathbb{E}P_j}$. The best upper bound for WSEPT known so far is desribed in the following theorem.

Theorem 3.7 [9] *When scheduling jobs with exponentially distributed processing times for $P| |\sum w_j C_j$, WSEPT is a $2 - \frac{1}{m}$-approximation.*

Proof We consider the following linear programming relaxation for instance J.

$$\begin{aligned} &\text{minimize} \sum_{j \in J} w_j C_j \\ &\text{subject to} \sum_{j \in I} \mathbb{E}(P_j) C_j \geq f(I), \qquad I \subseteq J, \end{aligned} \tag{8}$$

Where for scheduling exponentially distributed jobs

$$f(I) = \frac{1}{2m}\left((\sum_{j\in I} \mathbb{E}P_j)^2 + \sum_{j\in I} \mathbb{E}(P_j)^2 \right) \qquad \text{for all } I \subseteq J$$

It has been shown in [9] that this is indeed a linear programming relaxation for the stochastic scheduling problem $P||\sum w_jC_j$. This follows from the fact that inequalities (8) are valid for any scheduling policy. Moreover, since f is a supermodular set function, an optimal solution for (8) can be computed using Edmond's greedy algorithm. Under the assumption that $w_1/\mathbb{E}P_1 \geq w_2/\mathbb{E}P_2 \geq ... \geq w_n/\mathbb{E}P_n$, let C_j^{LP} denote an optimal LP solution for this linear programming relaxation. As described in [15], for exponentially distributed jobs the greedy algorithm results in the following optimal LP solution:

$$C_j^{LP} = \frac{1}{m}\sum_{k=1}^{j} \mathbb{E}P_k \tag{9}$$

The j^{th} job starts as soon as the first machine falls idle from processing jobs $,...,j-1$, which is at time $\frac{1}{m}\sum_{k=1}^{j-1} \mathbb{E}P_k$ or earlier. Therefore

$$\begin{aligned} \mathbb{E}C_j &\leq \tfrac{1}{m}\sum_{k=1}^{j} \mathbb{E}P_k + (1-\frac{1}{m})\mathbb{E}P_j && (10)\\ &= C_j^{LP} + (1-\tfrac{1}{m})\mathbb{E}P_j && (11) \end{aligned}$$

Which yields

$$\sum_{j\in I} w_j\mathbb{E}C_j \leq \sum_{j\in I} w_jC_j^{LP} + (1-\frac{1}{m})\sum_{j\in I} w_j\mathbb{E}P_j \tag{12}$$

Since LP (8) is a relaxation for the scheduling problem and since $\sum_{j\in I} w_j\mathbb{E}P_j$ is a lower bound on the expected performance of any scheduling policy, the WSEPT rule for exponentially distributed jobs is a $2-\frac{1}{m}$-approximation. ∎

By theorem 3.2 we know that the objective value of a WSEPT schedule is at most twice the optimal objective value. However, without knowing the optimal objective value, this does not tell us anything about how large the objective value of our policy actually is. It is possible to gain information about the objective value of a list scheduling policy by calculating the objective value for the deterministic counterpart. The following two theorems give bounds on the difference between the two objective values.

Theorem 3.8 *Let $\mathbb{E}C_j^{STO}$ denote the expected completion time of job j in a list scheduling policy for a stochastic scheduling instance, and let C_j^{DET} be the completion time in the same list schedule for the deterministic counterpart of this instance (with $p_j = \mathbb{E}P_j$). Then*

$$\mathbb{E}C_j^{STO} \leq C_j^{DET}.$$

This holds for instances with generally distributed processing times.

Proof For a given policy, let $g^{\Pi} : \mathbb{R}^n \to \mathbb{R}$ be the function that maps the realizations of the processing times to the completion time of job j. That is, $C_j^{STO} = g(\vec{p})$, where $\vec{p}$ denotes the vector of realizations of the processing times. For any list scheduling policy, g must be a concave function, since it consist only of addition and minima. Therefore by Jensen's inequality:

$$\mathbb{E}[g(\vec{P})] \leq g(\mathbb{E}[\vec{P}])$$

Which is equal to

$$\mathbb{E}C_j^{STO} \leq C_j^{DET}$$

▮

Corollary 3.9 *Let z^{STO} denote the expected weighted sum of completion times for scheduling stochastic jobs in a WSEPT order. Notice that this is defined for instances with generally distributed processing times. Let z^{DET} denote the weighted sum of completion times for the same list schedule in the deterministic counterpart. Then*

$$z^{STO} \leq z^{DET}.$$

Theorem 3.10 *Let z^{EXP} denote the expected objective value when scheduling exponentially distributed jobs in a WSEPT order. let z^{DET} denote the objective value for the same list schedule for the deterministic counterpart. Then*

$$z^{EXP} \geq \frac{m}{2m-1}\, z^{DET}$$

for instances with exponentially distributed processing times.

Proof In particular, (8) is also is a relaxation for the deterministic scheduling problem. Therefore, by the same arguments as (12):

$$z^{DET} \leq z^{LP} + (1 - \frac{1}{m}) \sum w_j p_j \tag{13}$$

Multiplying both sides by $\frac{m}{2m-1}$ yields:

$$\frac{m}{2m-1}\, z^{DET} \leq \frac{m}{2m-1}\, z^{LP} + (\frac{m-1}{2m-1}) \sum w_j p_j \leq z^{EXP} \tag{14}$$

Where the second inequality holds because $\frac{m}{2m-1} + \frac{m-1}{2m-1} = 1$ and both z^{LP} and $\sum w_j p_j$ are lower bounds for z^{EXP}. ▮

3.3 A tight bound for WSPT

The main result of this thesis, which is found in Section 3.4, was inspired by a result from Kawaguchi and Kyan [6]. They prove that for deterministic scheduling WSPT is a $\frac{1}{2}(1+\sqrt{2})$-approximation. Their paper also considers a worst-case instance where this bound is achieved. We briefly review this instance as the instance we will consider is a stochastic variant thereof. Let n be the number of jobs and m the number of machines. Denote the processing time of job j by p_j and its weight by w_j. The (deterministic) instance is then given by:

$$\begin{aligned}
&m = h + \lfloor (1+\sqrt{2})h \rfloor \\
&n = mk + h \\
&p_j = w_j = 1/k && \text{for} \quad 1 \leq j \leq mk \\
&p_j = w_j = 1 + \sqrt{2} && \text{for} \quad mk+1 \leq j \leq mk+h\,.
\end{aligned}$$

Here, h denotes an integer, and k is an integer that can be divided by $\lfloor (1+\sqrt{2})h \rfloor$. Notice that $\frac{w_j}{p_j} = 1$ for all j. This means that any list schedule is in fact a WSPT schedule. Let us refer to the first mk jobs as short jobs, and the remaining h jobs as long jobs.

Let S_L be the total weighted completion time of a schedule in which all short jobs are processed first, and S^* be the total weighted completion time of a schedule where the long jobs are processed first. Figure 3 depicts these two schedules. The schedule on the left of Figure 3 has value S^*. Here the last jobs of length $1/k$ finish at time $t \approx 1.4$ (for large values of m and k). The schedule on the right of Figure 3 has value S_L, it finishes the last jobs of length $1/k$ exactly at time 1. In Figure 3 we used $h = 5$ and $k = 32$. It can be verified (see [6]) that $S_L = (1+\sqrt{2})(2+\sqrt{2})h + (m/2)(1+1/k)$ and $S^* = (1+\sqrt{2})^2 h + (m/2)(m/\lfloor (1+\sqrt{2})h \rfloor + 1/k)$. The ratio S_L/S^* then tends to $(1+\sqrt{2})/2$ as $h \to \infty$ and $k \to \infty$.

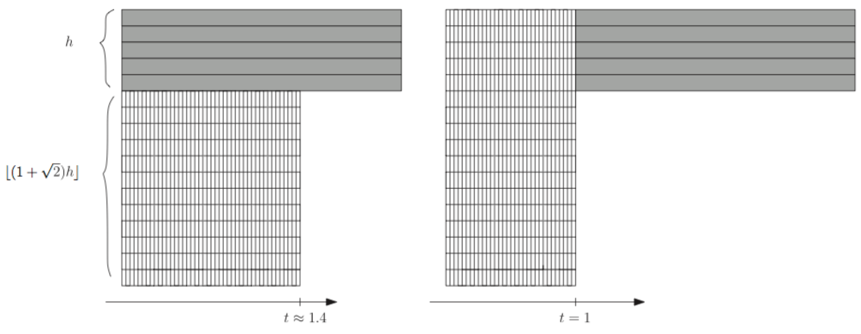

Figure 3: Two different WSPT schedules with values S^* and S_L respectively.

3.4 A lower bound for WSEPT

Theorem 3.11 *When scheduling jobs with exponentially distributed processing times on parallel, identical machines in order to minimize $\mathbb{E}[\sum w_j C_j]$, the performance guarantee of WSEPT is not better than α with $\alpha > 1.243$.*

To obtain our result, we carefully adapt and analyze the worst-case instance described in Section 3.3. Note that the originality of this result lies in the fact that $1.243 > \frac{1}{2}(1+\sqrt{2}) \approx 1.207$. Hence, stochastic scheduling with exponentially distributed processing times has worse worst-case instances than deterministic scheduling.

This result may seem counterintuitive: For exponentially distributed processing times and weights that are agreeable in the sense that there exists an ordering such that $w_1 \geq \cdots \geq w_n$ and $w_1\lambda_1 \geq \cdots \geq w_n\lambda_n$, scheduling the jobs in this WSEPT order is optimal (see Theorem 3.3), while the corresponding deterministic problem is already NP-hard, and in particular, WSPT is not optimal. That is to say, there are examples where the stochastic version with exponentially distributed processing times is computationally easier than the deterministic version of the same problem. Our result shows that with arbitrary weights, the situation is again fundamentally different.

The crucial insight when stochastifying the instance by Kawaguchi and Kyan is the following. The schedule with value S_L is essentially identical to the expected situation in stochastic scheduling. However, the schedule with value S^* has a significantly different realization with exponentially distributed processing times. In the next section we will analyze the behaviour of such jobs in order to understand the situation better.

3.4.1 Expected behaviour of exponentially distributed jobs

The key concepts that we need to understand in order to calculate the value S^* are expressed in the following lemmas, where λ and x are arbitrary positive parameters. In the following we denote by

$$H_n := \sum_{i=1}^{n} \frac{1}{i}$$

the nth harmonic number, with $H_0 := 0$.
The first lemma gives an estimate on expected job completion times for parallel jobs with $P_j \sim \exp(\lambda)$.

Lemma 3.12 *When scheduling in parallel m jobs with i.i.d. exponential processing times $P_j \sim \exp(\lambda)$, the expected number $m(t)$ of machines that are idle at a given time t is bounded as follows,*

$$m(t) \geq \lfloor m\left(1 - e^{-\lambda t}\right) \rfloor .$$

Proof The first completion time is distributed as the minimum of m independent $\exp(\lambda)$ distributions. This is an $\exp(m\lambda)$ distribution, hence it is expected at time $t_1 = \frac{1}{m\lambda}$. After the first job completion, we have $m-1$ jobs remaining. Since the exponential distribution is memoryless, the next completion is expected a time $\frac{1}{(m-1)\lambda}$ later, so $t_2 = \frac{1}{m\lambda} + \frac{1}{(m-1)\lambda}$. By continuing this argument we find that the kth job completion is expected at time

$$t_k = \sum_{j=1}^{k} \frac{1}{(m-j+1)\lambda} = \frac{1}{\lambda} \sum_{l=m-k+1}^{m} \frac{1}{l} = \frac{1}{\lambda} \left(H_m - H_{m-k} \right). \tag{15}$$

Note that $m(t_k) = k$, for $k = 1, \dots, m$. We now use that $H_i \geq \ln(i) + \gamma$ for all i, where

$$\gamma := \lim_{i \to \infty} (H_i - \ln i) \approx 0.57721$$

denotes the Euler-Mascheroni constant. Furthermore, $H_i - \ln(i)$ is monotonically decreasing in i. Hence we may conclude that

$$t_k \leq \frac{1}{\lambda} \left(\ln(m) + \gamma - \ln(m-k) - \gamma \right) = \frac{1}{\lambda} \ln\left(\frac{m}{m-k} \right). \tag{16}$$

Here, k is the expected number of finished jobs at time t_k. Hence, (16) yields

$$m(t_k) = k \geq m(1 - e^{-\lambda t_k}) \tag{17}$$

for $k = 1, 2 \dots, m$. Together with the fact that $m(t)$ is integer valued, (17) yields

$$m(t) \geq \lfloor m \left(1 - e^{-t\lambda} \right) \rfloor .$$

for all $t \geq 0$. ∎

Note that the last job is expected to finish at time $\frac{1}{\lambda}\Theta(\log m)$. Nevertheless, the average expected completion time of the jobs is $1/\lambda$; see also Figure 4 for an illustration.

Lemma 3.13 *Consider kT jobs with i.i.d. processing times $P_j \sim \exp(k)$ and weights $w_j = 1/k$, scheduled on a single machine. Then for all $\varepsilon > 0$ there exists k large enough so that*

$$\mathbb{E}\left[\sum_j w_j C_j \right] \leq \int_0^T t \, dt + \varepsilon .$$

Proof As there is an expected job completion each $1/k$ time steps, one easily calculates that $\mathbb{E}\left[\sum_j w_j C_j \right] = \frac{1}{2}T^2 + \frac{1}{2k}T$, so for $k \geq \frac{T}{2\varepsilon}$ the claim is true. ∎

Lemma 3.14 *Let $m(t) \geq 0$ denote the number of available machines at time t, and assume $m(t)$ is non-decreasing. When greedily scheduling jobs with i.i.d. processing times $P_j \sim \exp(k)$ and weights $w_j = 1/k$ from time 0 until T on the available machines, for all $\varepsilon > 0$ there exists k large enough so that*

$$\mathbb{E}\left[\sum_j w_j C_j \right] \leq \int_0^T m(t)\, t \, dt + \varepsilon .$$

Proof Let T_i $(i = 0, 1, 2, ..)$ be the times that a new machine becomes available, with $T_0 := 0$. For k large enough, we expect $m(T_i)k(T_{i+1} - T_i)$ jobs to be scheduled between times T_i and T_{i+1}. It is straightforward to extend Lemma 3.13 to this case, which yields for the jobs scheduled between times T_i and T_{i+1},

$$\mathbb{E}\left[\sum_j w_j C_j \right] \leq m(T_i) \int_{T_i}^{T_{i+1}} t \, dt + \varepsilon_i .$$

Therefore we get for all jobs,

$$\mathbb{E}\left[\sum_j w_j C_j \right] \leq \sum_i m(T_i) \int_{T_i}^{T_{i+1}} t \, dt + \varepsilon_i = \int_0^T m(t)\, t \, dt + \sum_i \varepsilon_i .$$

So for $\varepsilon = \sum_i \varepsilon_i$ and k accordingly large, the claim is true. ∎

The next lemma is concerned with the total weighted completion time of short jobs that succeed a set of long jobs.

Lemma 3.15 *Suppose we first schedule m i.i.d. long jobs with processing times $P_j \sim \exp(\lambda)$, followed by $x\,m\,k$ i.i.d. short jobs, with processing times $P_j \sim \exp(k)$ and weights $w_j = 1/k$, where k is large. Let S_{short} be the expected weighted sum of completion times of the short jobs. Then for any T such that $\frac{1}{\lambda}(e^{-\lambda T}-1) + \frac{(m-1)\,T}{m} \geq x$, and k large enough we have that*

$$S_{short} \leq \int_0^T \big(m(1-e^{-\lambda t}) - 1\big)\ t\ dt\,. \tag{18}$$

Proof Denote by $m(t)$ the number of machines at time t that are available for processing short jobs, and by T^* the earliest point in time such that we can expect all short jobs to be finished by time T^*. Notice that the total expected processing of short jobs equals $x\,m$. Therefore, for k large enough, T^* can be approximated arbitrarily well by the solution of the equation

$$x\,m = \int_0^{T^*} m(t)\,dt\,. \tag{19}$$

With T^* as in (19), Lemma 3.14 yields that $S_{short} \leq \int_0^{T^*} m(t)t\,dt + \varepsilon$. Now recall that $m(t)$ can be bounded as in Lemma 3.12. Define

$$f(t)\ =\ m(1-e^{-\lambda t}) - 1\,. \tag{20}$$

Then $m(t) \geq f(t)$ for all $t \geq 0$. If we require for T that

$$\begin{aligned} &\int_0^T f(t)\,dt\ \geq\ x\,m \\ \Leftrightarrow\ &\frac{1}{\lambda}\,(e^{-\lambda T}-1)\ +\ \frac{(m-1)\,T}{m}\ \geq\ x\,, \end{aligned}$$

then

$$x\,m = \int_0^{T^*} m(t)\,dt \leq \int_0^T f(t)\,dt$$

We therefore conclude that

$$\int_0^{T^*} m(t)\,t\,dt < \int_0^T f(t)\,t\,dt\,, \tag{21}$$

because $m(t) \geq f(t)$, and $m(t)$ is a step function while $f(t)$ is continuous. Intuitively, the expression $\int_0^T f(t)\,t\ dt$ equals the total weighted sum of completion times for infinitesimally small jobs (i.e., when $k \to \infty$), with total expected processing at least $x\,m$, scheduled on a set of “machines” that become available no earlier than $m(t)$. We finally conclude from (21) that

$$S_{short} \leq \int_0^{T^*} m(t)\,t\,dt + \varepsilon \leq \int_0^T f(t)\,t\,dt\,,$$

because ε can be chosen arbitrarily small for k large enough. ∎

A variation of this lemma will be used later in the analysis. Notice that the technical condition on T as stated in Lemma 3.15 only makes sure that all short jobs can be processed by time T when the machine availability is governed by $f(t)$ rather than $m(t)$. Finally, we make a statement about scheduling a block of (short) jobs.

Lemma 3.16 *Suppose we schedule $x\,m\,k$ i.i.d. short jobs with processing times $P_j \sim \exp(k)$ greedily on m machines. Then the average expected machine completion time equals x, and for any $\delta > 0$ there exists k large enough such that the earliest expected machine completion time is at time $t \geq x - \delta$.*

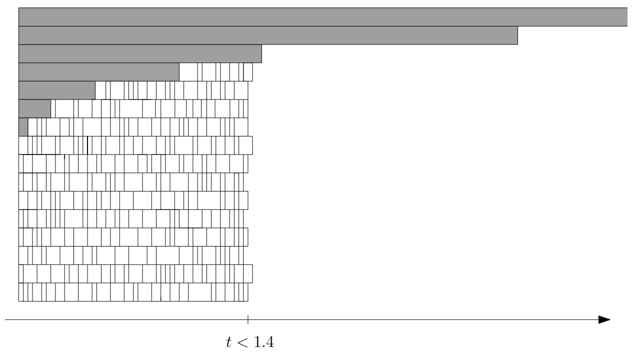

Figure 4: Schedule with value S^*: long jobs starting at time 0.

Proof The claim about the average expected machine completion time is clear, because the total expected processing is $x\,m$. For the second claim, consider the first time, say t, that a machine runs out of jobs. Then there are $m-1$ jobs still in process. We know from Lemma 3.12 that the last machine that runs out of jobs is expected to be at time $t+\sum_{i=1}^{m-1}\frac{1}{i\,k}$. For m large enough, we have $\sum_{i=1}^{m-1}\frac{1}{i\,k} \leq \frac{1}{k}\,[\ln(m)+\gamma]$. So for $k \geq (m-1)/(\delta(\ln(m)+\gamma))$, the last machine completion time is expected no later than $t+\delta/(m-1)$. Now the claim follows, as the average machine completion time is x. ▌

Even though other instances may lead to comparable results, we find it instructive to consider the stochastic analogue of the instance presented by Kawaguchi and Kyan [6]. Indeed, it turns out that the analyses for such instances use identical arguments, the core of which is represented by the lemmas given in Section 3.4.1.

We keep all parameters the same as in Section 3.3, except that the processing times of long jobs will be $P_j \sim \exp(1/(1+\sqrt{2}))$, and the processing times of short jobs will be $P_j \sim \exp(k)$. So the expected processing times of long and short jobs are identical to the deterministic processing times in [6].

Intuition about the schedules Suppose we start all long jobs first and then fill up the remaining machines with short jobs. By Lemma 3.12 we expect the i^{th} long job to finish at time:

$$t_i = \sum_{j=1}^{i} \frac{1+\sqrt{2}}{h-j+1} \tag{22}$$

Therefore, we expect the last short job to be completed significantly earlier than in the deterministic case. For a finite number of machines, the schedule will look like depicted in Figure 4. The crucial point is that the average expected time that machines finish processing short jobs will be smaller than in the deterministic case. This happens because many long jobs finish much earlier, and the late finishing of few long jobs doesn't matter for the short jobs. Hence, the contribution of the short jobs will decrease when compared to the deterministic case.

Suppose on the other hand that we first start all the short jobs. The set of short jobs is not likely to produce the ideal rectangle as it did in the deterministic case. However, as suggested by Lemma 3.16 the gap between the time the first machine runs out of short jobs and the time the last machine runs out of short jobs can be made arbitrarily small, by letting k, the inverse of the expected processing time of short jobs, be large. The crucial point is that, in this situation, the expected cost of the schedule is almost the same as the cost in the deterministic case.

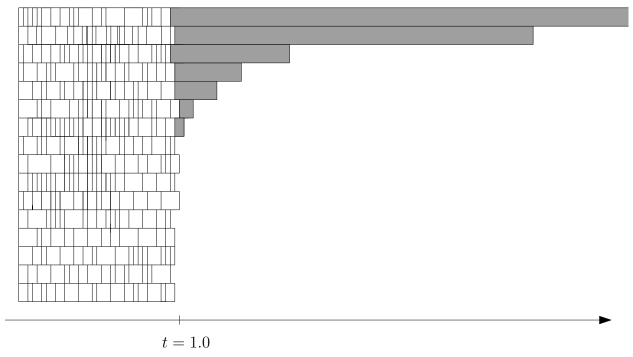

Figure 5: Schedule with value S_L: long jobs starting only after short jobs.

3.4.2 A lower bound on the approximation ratio

Let S^* denote the objective value $\mathbb{E}\left[\sum_j w_j\, C_j\right] = \sum_j w_j\, \mathbb{E}\left[C_j\right]$ for the case when we first schedule all long jobs. Similarly, let S_L denote the objective value for the schedule that starts long jobs only when there is no short job left to be scheduled. S^* is in fact optimal, whereas S_L is the worst case, but this is inessential. Both are in fact WSEPT, he
We choose h sufficiently large, and k, a multiple of $\lfloor(1+\sqrt{2})h\rfloor$, we may choose arbitrarily large in comparison to h (i.e., $k >> h$). In fact, we can make the choice of these two parameters in such a way that all our technical lemmas from Section 3.4.1 do apply.

The optimal case, S^*. We split S^* up into the contribution of the long jobs S^*_{long} and the contribution of the short jobs S^*_{short}. So

$$S^* = S^*_{long} + S^*_{short} \tag{23}$$

The value S^*_{long}. We start all h long jobs at time 0. Their expected completion time is $1+\sqrt{2}$ each. Hence the contribution of the long jobs is simply given by

$$S^*_{long} \;=\; h(1+\sqrt{2})^2\,, \tag{24}$$

which is actually the same as in the deterministic case.

The value S^*_{short}: This is a bit more complicated to calculate. We expect the short jobs to be located in the light and dark gray areas, as depicted in Figure 6.

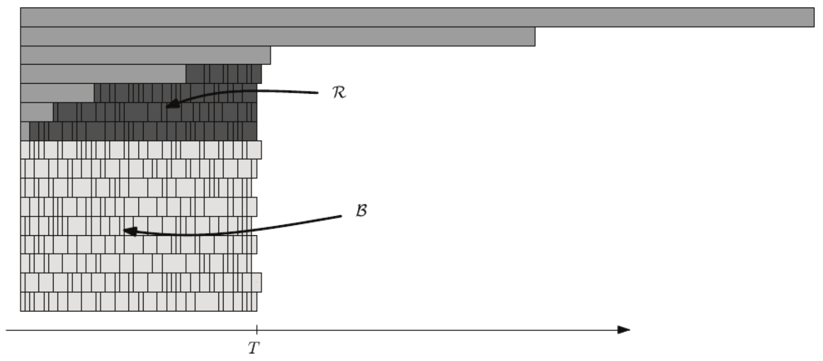

Figure 6: For T large enough, all short jobs fit in the light and dark gray areas $\mathcal{R}$ and $\mathcal{B}$.

The expected total processing of short jobs $\mathcal{B}$ that fit in the light grey area is given by

$$\mathcal{B} = \int_0^T (m-h)\,dt$$

According to (20) in the proof of Lemma 3.15, the number of finished long jobs at time $t \geq 0$ is at least:

$$f(t) \;=\; h(1 - e^{-t/(1+\sqrt{2})}) - 1\,.$$

Therefore, the expected total processing of short jobs $\mathcal{R}$ that fit in the dark grey area is bounded by

$$\mathcal{R} \geq \int_0^T f(t)\,dt$$

We want to find a value for T such that all short jobs are expected to be finished by T, i.e. $\mathcal{B} + \mathcal{R} \geq m$. We have not attempted to solve this equation analytically, but one can easily check that

$$T = 1.2933 \tag{25}$$

suffices.

Then S^*_{short}, the expected weighted sum of completion times for all mk short jobs, can be bounded similarly as in Lemma 3.15. We now find, for h and k sufficiently large,

$$S^*_{short} \leq \int_0^T (m-h)\,t\,dt \;+\; \int_0^T f(t)t\,dt\,. \tag{26}$$

With (25) and (26) we can calculate

$$S^*_{short} \leq\; 2.266h - 0.836\,. \tag{27}$$

Combining (24) and (27) gives

$$S^* \;=\; S^*_{long} \;+\; S^*_{short} \leq (1+\sqrt{2})^2 h + 2.266h - 0.836\,. \tag{28}$$

The worst case, S_L. Now we switch to the case where we first schedule all the short jobs. Again split the objective value into the two parts contributed by the short and long jobs, respectively.

$$S_L = S_L^{short} + S_L^{long}\,.$$

The value S_L^{short}: We have m machines working on mk jobs with processing times $P_j \sim \exp(k)$. According to Lemma 3.16, on average a machine is expected to finish with these jobs at time 1, and for any $\delta > 0$, we can find k large enough so that we expect all machines to be filled with short jobs at least until time $1-\delta$. Hence, we conclude that the average expected completion time of a short job is arbitrarily close to $1/2$. Therefore, for any $\varepsilon > 0$, there is k large enough so that

$$S_L^{short} \;\geq\; \frac{m}{2} \;-\; \varepsilon/2\,. \tag{29}$$

The value S_L^{long}: Remember that the schedule is expected to look like depicted in Figure 5. Using Lemma 3.16 again, we know that long jobs are expected to start no earlier than $1-\delta$. So by assuming they all start at this time, we get a lower bound for their completion times (and also for S_L^{long}). If all long jobs start at $1-\delta$, the average expected completion time is $2-\delta+\sqrt{2}$. Multiplying this by the weight and summing over all h jobs, we may conclude that for any $\varepsilon > 0$ there is k large enough so that

$$S_L^{long} \geq (2+\sqrt{2})\,(1+\sqrt{2})h \;-\; \varepsilon/2\,. \tag{30}$$

With (29) and (30) we now have

$$S_L = S_L^{short} + S_L^{long} \geq \frac{m}{2} \;+\; (2+\sqrt{2})\,(1+\sqrt{2})h \;-\; \varepsilon\,. \tag{31}$$

The ratio. Finally, let α be the approximation ratio of Smith's rule for exponentially distributed processing times. Then

$$\alpha \geq \frac{S_L}{S^*}.$$

Remember that $m = h + \lfloor (1+\sqrt{2})h \rfloor$. Now for carefully chosen $k >> h$, and taking $h \to \infty$, equations (28) and (31) give

$$\frac{S_L}{S^*} \geq \frac{m/2 + (2+\sqrt{2})\,(1+\sqrt{2})h - \varepsilon}{(1+\sqrt{2})^2 h + 2.266h - 0.836} > 1.229.$$

So we conclude that $\alpha > 1.229$. Note that this is strictly larger than the approximation ratio for WSPT in the the deterministic case, which is 1.207.

Optimizing the parameters. What remains to be done is to optimize over the parameters of the instance to improve the obtained lower bound. To that end, recall that the considered instance has h long jobs and $m = h + \lfloor (1+\sqrt{2})h \rfloor \approx 3.4h$ machines, and long jobs have processing times $P_j \sim \exp(\frac{1}{1+\sqrt{2}}) \approx \exp(0.41)$. However, these parameters are optimized for the deterministic instance. Taking slightly more longer jobs, namely by letting $m = 2.3h$, with somewhat shorter processing times, namely $P_j \sim \exp(0.56)$, we obtain a ratio of at least 1.2436, which proves Theorem 3.11.

4 Complexity considerations

A common problem in stochastic scheduling is that we are not able to calculate values such as an objective value or approximation ratio, because the situation is too complex. It is not easy to oversee what happens for all realizations, and how this affects the value we want to calculate. As before, we restrict the instances to jobs with exponentially distributed processing. This narrows down the situations that can occur in a schedule, and therefore we hope to be able to prove more statements. We ask ourselves, how difficult can it be to calculate the objective value for a certain policy, and how difficult is it in general to find an optimal policy.

4.1 Calculating the objective value given a policy

A policy can base the next decision on all realizations in the past, and calculating the corresponding objective value can therefore become computationally intractable. But even for a static list scheduling policy, it is not always clear whether the expected starting time of a job can be determined in polynomial time.

But let us start with a simple setting in which the objective value can be calculated in polynomial time.

Observation 4.1 [15] *If* $\forall i \;\; P_i \sim exp(\lambda)$ *the expected comletion time of the* j^{th} *job in a static list schedule is given by*

$$\mathbb{E}C_j = \begin{cases} 1/\lambda & \text{for } j = 1,..,m \\ j/(m\lambda) & \text{for } j = m+1,...,n \end{cases}$$

By using this observation the objective value can clearly be calculated in polynomial (linear) time.

We continue with the case with two types of jobs; $P_j \sim \begin{cases} exp(\lambda_1) & \text{for } j = 1,..,k \\ exp(\lambda_2) & \text{for } j = k+1,..,n \end{cases}$

Even when we ignore the weights, difficulties arise. We consider a policy where the jobs are scheduled greedily such that the jobs of one type (w.l.o.g. say type 1,) are all started before the jobs of the other type. One might hope there is an easy way to calculate the objective value for this policy. If we had some way of knowing that every machine was assigned at least one job of type 2, it would in fact be easy. We can calculate the completion times of the jobs of type 1 by observation 4.1, and use that *on average* a machine is expected to start on the jobs of type 2 at time $\frac{1}{m}\frac{k}{\lambda_1}$. However, note that regardless of the values of λ_1 and λ_2, there are always realizations for which not all machines have a job of type 2 assigned to it. For those cases we then need to obtain rather complex conditional expectations. So even for this relatively simple case, the situation still seems too complicated for us to easily calculate the objective value.

However, we are able to make a statement about the space complexity of calculating the objective value. Note that the crucial difference is that space can be reused, whereas time cannot. What we can show is the following.

Theorem 4.2 *Calculating the objective value of a given set strategy is $\in$ PSPACE*

Proof We can represent the given policy by a tree. The tree is structured as follows. A state consists of the set of completed jobs and the set of jobs in process at that time. Each node is a state the schedule can be in. The arcs are alternatingly either the starting or ending of a job, or a decision "continue by immediately scheduling another job" (which is of course only possible if there are a machine and a job available) or "wait for the next job to finish". If the decision was to wait for the next job to finish, the successor has at most m outgoing arcs (each corresponding to one of the currently running jobs being the next to finish). Otherwise, the successor has only one outgoing arc. An example might make this description more clear, see figure 7.

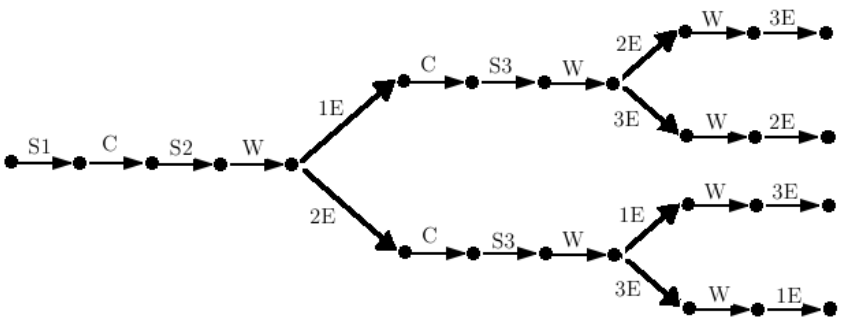

Figure 7: A tree respresenting a policy that greedily schedules jobs in the order 1, 2, 3 on two machines. The notation used in the tree is as follows. Sx means "start job x", C means "continue by immediately scheduling another job", W means "wait for the next job to finish" and xE means "job x ends".

Since eventually there are no more jobs available, this construction is finite. The result is a tree with at most $n!$ nodes, each corresponding to a different order of completion times. We go through the tree by depth first search. For every leaf, the conditional expectation of the objective value given the order or completion times corresponding to the leaf, can be calculated in polynomial time (and therefore also polynomial space). In order to calculate the objective value, we introduce a variable v, which we initially set to be 0. For every leaf, we compute the conditional expectation of the objective value multiplied with the probability of the jobs completing in the order corresponding to this leaf. We add this value to v and store only v. When successively going through all nodes, we can reuse space as we only need to store one number. ∎

The calculation desribed in the proof above takes exponential time. There may exists a faster, better way to calculate the objective value, but we believe that in general it is still hard. We are not able to prove the time complexity, but state the following conjecture.

Conjecture 4.3 *In general, calculating the expected weigthed sum of completion times given a policy is #P-hard.*

To get an intuition why #P would be the right class, think of a discrete probability distribution for every job length. There does not seem to be a way of calculating the objective value without considering every possible realization. There exists a stochastic scheduling problem that is proven to be #P-hard [3]. In this problem, the joblenghts are independent random variables with discrete, finite ranges. There are precedence constraints among the jobs, and the objective is to minimize the completion time of the last job (also known as the makespan). The network reliability problem, which is #P-complete [11], can be reduced to the described scheduling problem. However, the problem we are considering is much less structured due to the lack of precedence constraints. Therefore a hardness will probably not be similar to the one in [3].

4.2 Finding an optimal policy

Independent of the question if the objective value of a *given* policy can be computed efficiently, an even greater challenge is to find an optimal policy. We conjecture the problem is difficult, yet haven't found a proof.

A policy makes its decisions based on realizations in the past, and the distributions and weights of the unfinished jobs. Other than restricting to set strategies, there is no way known to exclude certain types of decisions from optimal policies in general. Comparing the perfomance of two policies by explicitly calculating their objective values, also seems to be difficult (see Conjecture 4.3). Therefore we suspect it is hard to find an optimal policy

Finding an optimal policy when we request that the jobs are distributed over the machines at time 0, is NP-hard. This is very similar to the deterministic variant, see Theorem 1.1. However, we know that assigning jobs to certain machines in advance, is suboptimal whenever $n > m$. And anyhow, this observation is not sufficient to prove that finding an optimal policy is NP-hard. Still, giving some thought to optimal policies in stochastic scheduling, quickly raises the suspicion that they are hard to find. The large amount of possible policies, combined with the fact that it is unclear how to compare the objective values without explicitly calculating them, leads to the following conjecture.

Conjecture 4.4 *When scheduling exponentially distributed jobs in order to minimize the weigthed sum of completion times, finding an optimal policy is most probably NP-hard, and probably even PSPACE-hard.*

A typical PSPACE-hard problem is the quantified Boolean formula problem:

$$\exists x_1 \forall x_2 \exists x_3 ... \forall x_n : \varphi(x_1, ..., x_n) \tag{32}$$

This can describe generalized versions of games such as Go and chess, where one plays against an optimizing opponent. We can consider scheduling to be a game where our goal is to find a policy with objective value $\leq k$ for a given number k, and we play against a randomizing "opponent" who decides on the realizations of the processing times. By slightly altering problem (32) we obtain a variant SSAT (*stochastic satisfiability*) that contains the stochastic nature of our scheduling problem.

$$\exists x_1 \mathbf{R} x_2 \exists x_3 ... \mathbf{R} x_n : P[\varphi(x_1, ..., x_n) = true] > \frac{1}{2}$$

With a new kind of quantifier $\mathbf{R}x$, for random x. Playing against a randomizing opponent turns out to be computationally as hard as playing against an optimizing opponent. Note that in our scheduling problem we accept if the expected objective value is less than k, while SSAT accepts if the probability of acceptance is more than half. In order to make SSAT a suitable description for our problem, there needs to be a correction. [10] contains a proof of PSPACE hardness for a version of stochastic scheduling that has a lot more structure than our problem.

However, obtaining a hardness result for our problem is probably difficult, since the problem has much less structure compared to the one adressed in [10]. This potentially makes it hard to define an appropriate transformation.

5 Conclusion

Let us summarize the results we have obtained in this thesis.

For stochastic scheduling on parallel machines in order to minimize the weigted sum of completion times, an optimal policy is idle time free when $m = 2$. Also, for any time t holds; if at the next completion time there are enough machines available to process all unscheduled jobs, idleness at time t is suboptimal. When $m = 3$ and $n = 5$, we proved that an optimal policy is idle time free. For larger instances, it remains an open question whether optimal policies ever require idleness when scheduling exponentially distributed jobs.

On one machine, WSEPT is optimal for minimizing $\mathbb{E}\sum w_j C_j$. When we consider multiple machines, WSPT for deterministic scheduling is an α-approximation, where $\alpha = \frac{1+\sqrt{2}}{2}$. This bound is tight. For exponentially distributed jobs, WSEPT is an α-approximation, with $1.243 < \alpha < 2$. When looking for an upper bound on α, it is sufficient to consider only instances with equal ratios $\frac{w_j}{\mathbb{E}P_j}$.

In general, attempting to solve stochastic scheduling problems by considering the jobs as if they were deterministic with length $\mathbb{E}P_j$ results in an overestimation of the objective value $\mathbb{E}\sum w_j C_j$. We show that for a WSEPT scheduling policy, the objective value for an instance with exponentially distributed processing times is at least half the objective value of the same (WSPT) list scheduling policy for the deterministic counterpart.

Calculating the expected weigthed sum of completion times for a given set strategy for exponentially distributed jobs is $\in$ PSPACE. It is probably #P-hard. Finding an optimal policy is most probably NP-hard, and probably even PSPACE-hard.

6 Discussion

Note that all numerical calculations have been performed using Mathematica.

We deliberately did not state a conjecture for the question of idleness in optimal policies. The fact that no one came up with an example where it is necessary, makes it appealing to think that optimal policies are idle time free. However, situations can become so unclear that we hesitate to make any statement at all.

For the performance of WSEPT, small improvements in the ratio 1.243 might be possible. We stated some theorems that might be useful for further research regarding the upper bound. However, the value $(2 - 1/m)$ seems out of reach for any instance. This leaves the question to improve the upper bound on the performance guarantee for WSEPT; in that respect, it is interesting to note that the analysis of [9] does not explicitly exploit the exponential distribution; it is valid in more generality.

For the complexity analysis, the lack of structure in our problem seems to be a recurring issue. For a problem with more structure, such as precedence constraints, it might be easier to find a hardness transformation.

Acknowledgements

First of all, I would like to thank Marc Uetz for all the time he spent supervising me. He suggested the topic of the thesis, and we often had productive meetings. He did a lot of work on the paper [4], and helped me with the preparations for the talk at WAOA 2010. Without him this thesis would have looked very different.

I would like to show my gratitude to Roberto Fernandez for agreeing to supervise me even though the topic was not exactly his branch of mathematics.

I also wish to express my thanks to Uwe Schwiegelshohn, for his contributions to our paper [4], part of which is contained in this thesis.

Finally, I would like to thank Bodo Manthey for sharing his expertise on complexity theory.

References

[1] J. L. Bruno, E. G. Coffman, and R. Sethi. Scheduling independent tasks to reduce mean finishing time. *Communications of the Association for Computing Machinery*, 17:382–387, 1974.

[2] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[3] J. N. Hagstrom. Computational complexity of PERT problems. *Networks*, 18:139–147, 1988.

[4] C. Jagtenberg, U. Schwiegelshohn, and M. Uetz. Lower bounds for smith's rule in stochastic machine scheduling. In K. Jansen and R. Solis-Oba, editors, *Approximation and Online Algorithms*, volume 6534 of *Lecture Notes in Computer Science*, pages 142–153. Springer, Heidelberg, 2011.

[5] T. Kämpke. *Optimalitätsaussagen für spezielle stochastische Schedulingprobleme*. PhD thesis, Rheinisch-Westfälische Technische Hochschule Aachen, Aachen, Germany, 1985.

[6] T. Kawaguchi and S. Kyan. Worst case bound on an LRF schedule for the mean weighted flow-time problem. *SIAM Journal on Computing*, 15:1119–1129, 1986.

[7] J. K. Lenstra. Unpublished manuscript.

[8] R. H. Möhring, F. J. Radermacher, and G. Weiss. Stochastic scheduling problems II: Set strategies. *ZOR - Zeitschrift für Operations Research*, 29:65–104, 1985.

[9] R. H. Möhring, A. S. Schulz, and M. Uetz. Approximation in stochastic scheduling: The power of LP-based priority policies. *Journal of the Association for Computing Machinery*, 46:924–942, 1999.

[10] C. H. Papadimitriou. Games against nature. *Journal of Computer and System Sciences*, 31:288–301, 1985.

[11] J. S. Provan and M. O. Ball. The complexity of counting cuts and of the probability that a graph is connected. *SIAM Journal on Computing*, 12:777–788, 1983.

[12] M. H. Rothkopf. Scheduling with random service times. *Management Science*, 12:703–713, 1966.

[13] W. E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3:59–66, 1956.

[14] M. Uetz. Stochastisches Scheduling — Algorithmen und Polyedrische Methoden. Diploma Thesis (in German), Institut für Mathematik, Technische Universität Berlin, Berlin, Germany, 1996.

[15] M. Uetz. *Algorithms for Deterministic and Stochastic Scheduling.* PhD thesis, Institut für Mathematik, Technische Universität Berlin, Berlin, Germany, 2001. Published: Cuvillier Verlag, Göttingen, Germany.

[16] M. Uetz. When greediness fails: Examples from stochastic scheduling. *Operations Research Letters*, 31:413–419, 2003.