Utrecht University

Graduate School of Natural Sciences

Sampling-based Stochastic Single-machine Scheduling

Master's Thesis

Puck te Rietmole

Mathematical Sciences

Supervisors:

Prof. Dr. Marc Uetz
University of Twente

Prof. Dr. Rob Bisseling
Utrecht University

June 7, 2023

Abstract

The problem to be considered is non-preemptive stochastic single machine scheduling to minimize the total weighted completion time of a set of N jobs. This is a classical problem setting that has been studied in the literature on scheduling, with many applications in operations research. In this thesis, we try to understand algorithms and models where the underlying probability distributions are unknown, and processing times can only be assessed via samples. Specifically, the thesis asks what can be done when there is one single sample only.

Whenever the expected processing times are known exactly, it is known since the 1960's that one must schedule jobs in decreasing order of weight over expected processing time. For the setting with access to only one sample of the processing time, we define `SWOPS`, the algorithm that Schedules by Weights Over Processing Time Samples. This algorithm is the most intuitive candidate for a better-than-random algorithm. We first formalize the concept of comparing algorithm performance. This allows us to formulate our main research question; in which scenarios does `SWOPS` perform at least as well as R, the uniformly random algorithm? We show by counter-example that in general, the performance of `SWOPS` can be arbitrarily worse than R, motivating that our research question is not trivial. We then show that the matter of comparing relative performance of algorithms can be simplified to a case where the number of jobs N equals 2. We also introduce a novel concept called the Relative Optimality Gap (ROG) of an algorithm. This scales the algorithm's cost to always lie in the interval $[0, 1]$, where a ROG of 0 corresponds to an optimal algorithm and a ROG of 1 means the expected performance is the worst possible. It is then proven that finding an upper ROG-bound at most $\frac{1}{2}$ on an algorithm A is equivalent to showing that A performs at least as well as R.

We apply these methods to several scenarios, for different types of processing time distributions, and show that `SWOPS` performs well in these cases. The first scenario is when processing time distributions are symmetric. Second, when all processing time distributions are translated versions of the same underlying distribution. Third, when all processing time distributions are linearly scaled versions of the same underlying distribution. Lastly, we consider several special sub-cases for exponentially distributed processing times.

In the end, we discuss what perspective these findings give us about the performance of `SWOPS`. We also speculate about the broader implications of our results, in terms of what types of critical failure modes might exist when insufficient samples are available.

Contents

1 Acknowledgments

The author would like to thank her supervisors, Marc Uetz and Rob Bisseling, for their much appreciated advice and feedback. Besides proposing the problem, Marc also provided extensive help in weekly meetings, which was essential for getting the project to where it is now.
The author would also like to extend her regards to Tristan van Leeuwen, the second reader, for taking the time to read through the document in detail.
Lastly, it would be unthinkable not to mention the invaluable support of friends and family during a period that might arguably otherwise have been one of the hardest in my life. Thank you very much.

To Jan Fortuin, an awe-inspiring researcher, formidable teacher, and loving grandfather.

2 Introduction: Single Machine Scheduling and Sampling

The research into job scheduling is extremely prolific. In the 1950's, the problem was first introduced to explore optimal ways to organise such things as assembly lines in factories. Since then, many variations have been proposed and researched. Applications have been found in a wide variety of topics, including manufacturing, healthcare, supply chains, computer operating systems, and many more [1].

2.1 Background for fully known processing time distributions

One of the earliest papers in the field, by Smith [2] in 1956, introduces the following problem. Consider N jobs with weights w_j and processing times p_j, $1 \leq j \leq N$. Here the weights can be interpreted as stating something about the urgency of a job. The processing time, as the name suggests, is a measure of how long the execution of a job will take from start to completion. The optimization goal is to find a permutation $s \in S_N$ (also called the schedule) that minimizes the cost function $\text{Cost}(s) = \sum_j w_j C_j(s)$. Here $C_j(s) := \sum_{i,s(i)\leq s(j)} p_i$ is the completion time of job j under schedule s, which is given as the sum of processing times of all jobs before and up to job j in schedule s. S_N denotes the set of all $N!$ possible permutations of the N jobs. This cost function can be thought of as a weighted sum of the waiting times of jobs under the schedule s. An additional assumption we make in the entire setting of this thesis is that jobs are non-preemptive. This means that a job cannot be halted or discontinued until it is completed.
In the Smith [2] paper, it is proven by a simple exchange argument that the optimal solution to the above problem is achieved if and only if the jobs are scheduled in descending order of the value of $\frac{w_i}{p_i}$, i.e. , the job weights divided by their processing times (See [2], Section IV). The higher the weight per processing time for a given job, the better it is to schedule this job early. This is sometimes called the WSPT rule, because it schedules the jobs with the Weighted Shortest Processing Time first.

An extension of the original problem called stochastic single-machine scheduling was later formulated. In the stochastic version of the problem, processing times are given as distributions P_j, and the new cost-function is defined as $\text{Cost}(s) := \mathbb{E}\left(\sum_j w_j C_j(s)\right)$.
The problem of stochastic single-machine scheduling was discussed in 1966 by Rothkopf [3]. It can be shown that the optimal schedule is to put the jobs in non-increasing order of the value of $\frac{w_j}{\mathbb{E}(P_j)}$, i.e. the weights divided by the expected processing times. A recap of the proof of this result in the notation of this thesis will be discussed in Section 3.1.

2.2 Motivation

In practical applications, it is rare to have access to the expected processing times of jobs. Rather, it seems more reasonable that a number of samples is made to sufficiently accurately estimate the expected processing times. Since the result by Rothkopf strictly speaking only applies to the cases where the expected processing time is known precisely,

we can only apply it when the number of samples is high enough, by the law of large numbers. This thesis will give some results and insight into the scenarios where the number of samples is smaller.
It is also worth noting that this research is related to learning-augmented algorithms, which is currently a very active field of study. In learning-augmented algorithms, an algorithm uses a limited number of samples to learn an unknown parameter or set of parameters. See also *Algorithms with Predictions* [4] for a depository of recent papers on this topic.

2.3 Problem description for sampled processing times

We define a new version of the single machine scheduling problem, in which the underlying processing time distributions are unknown, and can only be assessed through sampling. Specifically, we focus on the extreme case where each such distribution may only be sampled exactly once. The goal is to investigate the algorithm that Schedules by Weights Over Processing-time Samples, referred to as `SWOPS`. This algorithm divides the weight w_j of each job j by its sampled processing time p_j, and then schedules the jobs in non-increasing order of these values. In case of ties, it decides on a schedule uniformly at random. Intuitively, one can think of `SWOPS` as doing what can be considered common practice, namely assuming the finite samples actually correspond to the full distribution of processing times. The difference is that this is now applied to the setting where we only have one such sample.
Our main research question is in which scenarios we can show that `SWOPS` performs at least as well as R, the uniformly random algorithm. The statement of this question will be made more rigorous in Section 3.2 of this thesis.

2.4 Outline of the thesis

In Section 3, we first build up a methodology for analyzing the single-sample stochastic single-machine scheduling problem. In Section 3.1 we discuss the result and proof achieved by Rothkopf [3] for the case where processing time distributions are fully known. Next, in Section 3.2, we discuss what challenges arise in the case where distributions are only accessible through sampling. We also give a number of definitions to formalize what it means for an algorithm to perform better than random. Then, in Section 3.3, we discuss a concrete example as a motivation that finding an algorithm that performs better than random is non-trivial. Afterwards, in Section 3.4, we show that under some assumptions on the problem space and the algorithms, the problem can be reduced to problem instances with only two jobs. Lastly, in Section 3.5, we introduce the concept of the Relative Optimality Gap (ROG), which allows us to show stronger results for the performance of algorithms when compared to the uniformly random algorithm R.
Next, in Section 4, we apply the derived methods to study the behavior of `SWOPS` in specific scenarios. First we consider the case of jobs with symmetrically distributed processing times in Section 4.1. Then, in Sections 4.2 and 4.3, we study the cases where processing time distributions are translated or scaled versions of the same underlying distribution, respectively. In Section 4.4 we study the very specific case where jobs have exponentially distributed processing times, and show that under some additional assumptions on the job parameters, we can get a strong result using the Relative Optimality Gap approach.

After that, in Section 4.5, we explore whether SWOPS still performs provably well if we relax some of the constraints from Section 4.4.
Finally, in Section 5.2, we outline future challenges, and potential research directions to be looked into.

3 Methods: Algorithm Analysis for Sampled Processing Times

3.1 Results in the case of fully known processing time distributions

We briefly summarize one of the results from Rothkopf [3].
We will prove that a schedule s^* minimizes $\text{Cost}(s) := \mathbb{E}\left(\sum_j w_j C_j(s)\right)$ if and only if the following implication holds:
If job j occurs before job k in the schedule s^*, then $\frac{w_j}{\mathbb{E}(P_j)} \geq \frac{w_k}{\mathbb{E}(P_k)}$.
In other words, s^* is optimal if and only if it schedules the jobs in order of descending value of the fraction of the job weights over their expected processing times.

Proof. $\Longrightarrow$: From the fact that there exists only a finite number of distinct schedules s, it follows that there must exist at least one schedule that minimizes the cost-function.
Assume s^* to be such an optimal schedule. Suppose to the contrary that there exist two jobs j, k such that j occurs before k in s^*, but $\frac{w_j}{\mathbb{E}(P_j)} < \frac{w_k}{\mathbb{E}(P_k)}$. Then it follows that there must also exist two consecutive jobs l, m in the schedule s^* such that l occurs before m but $\frac{w_l}{\mathbb{E}(P_l)} < \frac{w_m}{\mathbb{E}(P_m)}$.
We can rewrite the expression for $\text{Cost}(s)$ as

$$\text{Cost}(s) = \mathbb{E}\left(\sum_j w_j C_j(s)\right) = \sum_j w_j \mathbb{E}(P_j) + \sum_{\substack{j,k \\ j \text{ occurs before } k \text{ in } s}} w_k \mathbb{E}(P_j),$$

where we have used linearity of expectation. Consider the schedule s' which is the same as s^*, except with jobs l, m swapped. By the above expression for $\text{Cost}(s)$, we see that this swap adds and removes precisely one term from the right-hand sum, and otherwise leaves the cost-function value unchanged. Expressing this in an equation yields

$$\text{Cost}(s^*) - \text{Cost}(s') = w_m \mathbb{E}(P_l) - w_l \mathbb{E}(P_m) > 0,$$

where the last inequality follows from the fact that $\frac{w_l}{\mathbb{E}(P_l)} < \frac{w_m}{\mathbb{E}(P_m)}$.
This contradicts our initial assumption that s^* is optimal, and concludes the proof in this direction.

$\Longleftarrow$:
Assume that s is such that the jobs are in non-increasing order of the value of the fractions $\frac{w_j}{\mathbb{E}(P_j)}$. Then we must show that s minimizes $\text{Cost}(s)$.

Suppose s' is an optimal schedule, and $s' \neq s$.
By the previous argument s' must also schedule the jobs in non-increasing order of the value of the fractions $\frac{w_j}{\mathbb{E}(P_j)}$. Therefore, s and s' can only differ in the order of jobs that have equal values for $\frac{w_j}{\mathbb{E}(P_j)}$. It can easily be shown using a similar argument to the one above that permuting such jobs does not affect the value of the cost-function, because if $\frac{w_j}{\mathbb{E}(P_j)} = \frac{w_k}{\mathbb{E}(P_k)}$, then $w_j\mathbb{E}(P_k) - w_k\mathbb{E}(P_j) = 0$. By repeated swaps of adjacent jobs with equal values of $\frac{w_j}{\mathbb{E}(P_j)}$, we can turn s' into s. Since each swap leaves the cost unchanged, we see that $\text{Cost}(s') = \text{Cost}(s)$. □

3.2 Formalizing the sampling version of the single-machine scheduling problem

In this section we will introduce several new definitions and notations to help formalize the single-sample stochastic single-machine scheduling problem.
There are again N jobs, with weights w_j and processing time distributions P_j, $1 \leq j \leq N$.

3.2.1 Algorithms for sampled processing times

The types of algorithm that will be considered in this thesis are referred to in the literature as "non-preemptive static list policies". Below is a quoted definition from Pinedo [5], page 257. [1]

Definition 3.1 (Non-preemptive static list policy)**.** Under a non-preemptive static list policy the decision maker orders the jobs at time zero according to a priority list. This priority list does not change during the evolution of the process. Every time the machine is freed, the next job on the list is selected for processing.

Here the word static means that the priority list, once made, is not updated. This is in contrast to dynamic algorithms, which can choose to change the priority list order of jobs that have not yet been started, based on the realized processing times of executed jobs. In the setting of the Rothkopf paper, with fully known processing times, we can without loss of generality consider only static policies. This is because the processing time distributions of different jobs are independent from one another. Therefore, knowing the realized processing of one job does not change the optimal order in which to schedule the other jobs. This is formally proven in Pinedo [5], Theorem 10.1.1.
In the case of sampled processing times, we expect for the same reasons that it is sufficient to consider only non-preemptive static list policies.

Formally, we consider non-preemptive static list policy algorithms A to be functions that take as their input the number of jobs, the weights, and a single sample of each processing time distribution. A then returns a discrete distribution over all possible schedules $s \in S_N$. The schedule used is then pulled from this distribution.

[1] The definition has been reworded slightly to apply to single-machine scheduling rather than parallel machine-scheduling.

Algorithm A
Input: $N, (w_j)_{1\leq j\leq N}, (p_j \sim P_j)_{1\leq j\leq N}$
Output: a distribution over the set of schedules, S_N.

The reason A returns a distribution over schedules rather than a schedule, is because we also allow for non-deterministic algorithms. An example is R, the uniformly random algorithm, which assigns each schedule an equal probability of $\frac{1}{N!}$. For ease of reading, in the future we will refer to the resulting schedule as "the schedule picked by A", rather than "the schedule pulled from the distribution that was output by A".

3.2.2 Cost-function for sampled processing times

The changes compared to the case where processing time distributions were fully known add multiple sources of randomness into the problem. Firstly, different processing time samples may lead to different schedules. Secondly, because the algorithms themselves are not necessarily deterministic, even running them multiple times with the same samples in the input may lead to different schedules. In order to make meaningful statements about algorithm performance we want a cost-function that is deterministic, eliminating both these sources of randomness. This motivates the following definition.

Definition 3.2 (cost function)**.** The cost function in the single machine scheduling problem with sampled processing times is defined as

$$\text{Cost}(A) := \sum_{s\in S_N} \Pr\left(A \text{ picks } s\right) \mathbb{E}_{p\sim P}\left(\sum_j w_j C_j(s)\right).$$

Note that this cost-function is now a function of the algorithm A, rather than the schedule s. This makes sense, because in the single-sample setting, the question of what schedule we pick may change based on what we sample for the processing times. The meaningful things to compare are the algorithms, not the schedules themselves.
Another point to note is that the right-most part of the expression, $\mathbb{E}\left(\sum_j w_j C_j(s)\right)$, is a deterministic function of s. It is equivalent to the cost-function used in the setting of the Rothkopf [3] paper. The expectation operator takes the expectation over the processing times. When a schedule s is fixed, this expression simply yields the expected sum of weighted completion times corresponding to that schedule.
The fact that the right-most part of the expression is deterministic when we fix s, means that we can interpret $\text{Cost}(A)$ as a weighted sum over schedules. From this perspective, the role of the algorithm A is to determine the "weights", $\Pr\left(A \text{ picks } s\right)$. It is important to realize that s must not depend on the *realized* processing times, as these are not known to A. We see that $\text{Cost}(A)$ decreases as A returns bad schedules with lower probability, and good schedules with higher probability. In the ideal case, A would return the best schedule, minimizing $\mathbb{E}\left(\sum_j w_j C_j(s)\right)$, with probability 1. However, since the processing time distributions are not fully known, and only sampled once, in general this is not possible.

Also remark that this cost-function can be seen as a generalization of the cost-function employed in the Rothkopf [3] paper. Consider the case where A has the full processing time distributions as its input rather than a single sample (eliminating randomness from sampling), and was required to give a deterministic output rather than a distribution over schedules. Then the cost-function defined above simplifies to the cost-function defined by Rothkopf [3], namely $\mathbb{E}\left(\sum_j w_j C_j(s)\right)$.
Another perspective on this cost-function is that minimizing it is equivalent to minimizing the expected regret achieved by an algorithm A. Here the regret is the difference in cost compared to the offline optimal solution that one could obtain if one had known in advance the realized processing times. Since this idea is not key to any of our further proofs and statements, its explanation can be found in the Appendix, Section A.

3.2.3 Comparing algorithm performance

This new cost-function allows us to compare the performance of algorithms on a single problem instance, as characterized by the weights and processing time distributions. However, we would prefer to be able to meaningfully compare algorithms also in a broader context. This motivates the following definitions.
We formally define a problem instance, denoted I, as $I := (w_j, P_j)_{1 \leq j \leq N}$, where $N \geq 2$, $w_j \in \mathbb{R}_{>0}$, and P_j are distributions over $\mathbb{R}_{\geq 0}$. That is to say, a problem instance consists of a tuple of $N \geq 2$ jobs, which in turn are fully described in terms of their weights and processing time distributions.
The full problem space, denoted $\mathcal{I}_{\text{all}}$, is then defined to be the set of all such problem instances. Or in mathematical notation, $\mathcal{I}_{\text{all}} := \{I = (w_j, P_j) | w_j \in \mathbb{R}_{>0}, P_j \in \{\text{distributions over } \mathbb{R}_{\geq 0}\}\}$.
In our research, we will often only consider the problem in the context of a subset of the full problem space. An example of what such a subset could be is $\mathcal{I}_{exp} := \{I = (w_j, P_j) | w_j \in \mathbb{R}_{>0}, P_j \text{ with PDF } f_j(x) = \lambda_j e^{-\lambda_j x}\}$. This problem space would correspond to the case where we consider arbitrary weights, but only allow exponentially distributed processing times. The ability to distinguish between the problem as considered on different problem spaces allows us to be more nuanced when comparing the performance of algorithms. For example, an algorithm that does not outperform another algorithm in general, might do so when we restrict the problem space to a set of more "reasonable" problems.

Now we have all the tools needed to formally define what it means for an algorithm to outperform another algorithm.
Let $\mathcal{I} \subset \mathcal{I}_{\text{all}}$ be a problem space, and let A, B be algorithms that are well-defined on all of $\mathcal{I}$. We define a partial order "$\leq_{\mathcal{I}}$" on algorithms. Namely, $A \leq_{\mathcal{I}} B$ if and only if $\text{Cost}_I(A) \leq \text{Cost}_I(B)$ for all $I \in \mathcal{I}$. Here the cost-function, $\text{Cost}_I(A)$, has been subscripted with the problem instance I to emphasize that it gives the cost-function value achieved by A for problem instance I. When $A \leq_{\mathcal{I}} B$, we say that "A performs as well as B on the problem space $\mathcal{I}$". Another way to interpret this definition is that in order for A to be as good as B, we require that there does not exist a problem instance on which B achieves a better cost-function value than A.
Note that this order is indeed only partial. There are many easy examples of cases where for two problem instances $I, I' \in \mathcal{I}$, we have that $\text{Cost}_I(A) < \text{Cost}_I(B)$, but

$\text{Cost}_{I'}(A) > \text{Cost}_{I'}(B)$. When this is the case, we cannot compare A and B using $\leq_{\mathcal{I}}$.

3.2.4 Formally defining the SWOPS algorithm

Lastly, let us reiterate the definition of the SWOPS algorithm, already briefly mentioned in Section 2.2. Its name stands for the algorithm that Schedules by Weights Over Processing-time Samples. As the name implies, this algorithm calculates the value of the fraction $\frac{w_j}{p_j}$ for each job j, where w_j is the weight of job j, and p_j is the sampled processing time of job j. The algorithm then schedules the jobs in descending order of the value of these fractions, deciding uniformly at random in the case of ties. Note that the case where $p_j = 0$ does not form a problem, as we can simply decide to always put such a job (or jobs) at the start of the schedule.
The SWOPS algorithm is arguably the best candidate for an algorithm to outperform R. It mirrors the idea behind the proof in Section 3.1. There we saw that for fully known distributions, the optimal schedule is achieved by scheduling jobs in order of the value of the fraction $\frac{w_j}{\mathbb{E}(P_j)}$. It follows trivially that if the samples of the processing times p_j that form the input of the SWOPS algorithm correspond exactly to the expected processing times $\mathbb{E}(P_j)$, then the SWOPS algorithm is optimal as well. In particular, it would outperform R.
In reality, the samples are unlikely to precisely match the expected values of the distributions they are drawn from. Nevertheless, in cases where processing time samples carry a lot of information about expected processing times, it seems reasonable to expect the SWOPS algorithm to perform at least better than R.
The common practice when doing stochastic scheduling with a finite number of samples, is to make the simplifying assumption that enough samples are available that their average is equal to the expected processing time of a job. Another way to look at SWOPS is as the algorithm that applies this practice to the scenario where only one sample is available. Our goal in considering SWOPS is in a sense to see under what circumstances this simplification is harmful.

Having prepared the necessary mathematical tools, we can now state our main research question in the following way. Under what restrictions on $\mathcal{I} \subset \mathcal{I}_{\text{all}}$, if any, does it hold that SWOPS $\leq_{\mathcal{I}} R$?

3.3 A concrete example to show that sampling may not help

At this point, the critical reader might well be wondering if the single-sample single-machine stochastic scheduling problem is, in fact, non-trivial. The algorithm R, which selects among all $N!$ possible schedules $s \in S_N$ uniformly at random, does not seem like a very formidable adversary. We have asked under what restrictions on $\mathcal{I} \subset \mathcal{I}_{\text{all}}$, if any, does it hold that SWOPS $\leq_{\mathcal{I}} R$. In this section we will argue that this question is non-trivial, using a concrete example that shows that following the samples may generally perform worse than random. The example is from [6]. In the problem instance we will construct, the SWOPS algorithm performs worse than R. Not only that, but the cost-function value achieved by SWOPS can be made an arbitrary amount worse than that achieved by R by

simply changing the problem parameters.

Consider the problem instance I with $N = 2$ jobs, and uniform weights $w_1 = w_2 = 1$. The processing time distributions are given as discrete distributions, as follows

$$P_1 = \begin{cases} 0 & \text{with probability } 1 - \frac{1}{M} \\ M^2 & \text{with probability } \frac{1}{M} \end{cases}$$
$$P_2 = \epsilon \text{ with probability 1.}$$

Here $M \in \mathbb{R}_{>0}$ is an arbitrary positive number, taken to be very large. Similarly, $\epsilon \in \mathbb{R}_{>0}$ is also an arbitrary positive number, taken to be very small.
One can easily calculate the expected values of the processing times for P_1 and P_2 to be $\mathbb{E}(P_1) = 0(1 - \frac{1}{M}) + M^2(\frac{1}{M}) = M$, and $\mathbb{E}(P_2) = \epsilon$.
We can see that when the `SWOPS` algorithm has as its input a sample $p_1 = 0$, which happens with probability $1 - \frac{1}{M}$, it schedules job 1 before job 2. And when the sample is $p_1 = M^2$, which happens with probability $\frac{1}{M}$, it schedules job 2 before job 1. The cost-function value achieved by `SWOPS` is then simply

$$\begin{aligned} \text{Cost}_I(\texttt{SWOPS}) &:= \sum_{s \in S_2} \Pr(\texttt{SWOPS} \text{ picks } s)\, \mathbb{E}\left(\sum_j w_j C_j(s)\right) \\ &= w_1\mathbb{E}(P_1) + w_2\mathbb{E}(P_2) + (1 - \frac{1}{M})w_2\mathbb{E}(P_1) + \frac{1}{M}w_1\mathbb{E}(P_2) \end{aligned}$$

The first two terms here are independent of the schedule picked by `SWOPS`. They can be thought of as representing the fact that every job has to at least wait the length of its own processing time before it is completed. The third term means that with probability $(1 - \frac{1}{M})$, `SWOPS` will schedule in the order 12, incurring a cost of $w_2\mathbb{E}(P_1)$. The fourth term signifies that with probability $\frac{1}{M}$, `SWOPS` will schedule in the order 21, incurring a cost of $w_1\mathbb{E}(P_2)$.
Plugging in the weights and expected processing times, we obtain the expression

$$\begin{aligned} \text{Cost}_I(\texttt{SWOPS}) &= M + \epsilon + (1 - \frac{1}{M})M + \frac{1}{M}\epsilon \\ &= 2M - 1 + \epsilon + \frac{\epsilon}{M}. \end{aligned}$$

The algorithm R on the other hand schedules job 1 before job 2 with probability $\frac{1}{2}$, and job 2 before job 1 with probability $\frac{1}{2}$. The cost-function value achieved by R is then

$$\begin{aligned}\mathrm{Cost}_I(R) &:= \sum_{s\in S_2} \Pr(R \text{ picks } s)\,\mathbb{E}\left(\sum_j w_j C_j(s)\right)\\ &= w_1\mathbb{E}(P_1) + w_2\mathbb{E}(P_2) + \frac{1}{2}w_2\mathbb{E}(P_1) + \frac{1}{2}w_1\mathbb{E}(P_2)\\ &= M + \epsilon + \frac{1}{2}M + \frac{1}{2}\epsilon\\ &= \frac{3}{2}M + \frac{3}{2}\epsilon.\end{aligned}$$

Consider the limit where $M \gg 1$ and $\epsilon \ll 1$. Then we have that the difference $\mathrm{Cost}_I(\texttt{SWOPS}) - \mathrm{Cost}_I(R) \approx \frac{1}{2}M \gg 1$. In other words, the difference in cost-function value between the two algorithms can be made arbitrarily large by increasing the parameter M.

With this example we have illustrated that there exist settings in which `SWOPS`, an algorithm that seemingly makes good use of the additional information in the form of processing time samples, nevertheless performs worse than R, which does not make use of the processing time samples. This motivates that our main research question is indeed non-trivial. In the next section, we will give one of our main results. Namely, we will show that under sufficient assumptions on the algorithms and problem space, the question of whether one algorithm outperforms another can be answered by only looking at problem instances of size $N = 2$.

3.4 Reduction to number of jobs $N = 2$

In this section we will show that under specific assumptions on the problem space $\mathcal{I}$, and on the algorithms A and B, the question of whether $A \leq_{\mathcal{I}} B$ can be simplified in terms of the behavior of the algorithms on pairs of jobs only. In order to obtain this result, we first introduce several new definitions and notations.

Definition 3.3 (closed)**.** Let $\mathcal{I} \subset \mathcal{I}_{\text{all}}$ be a problem space. We say a job $j = (w_j, P_j)$ is *legal* in $\mathcal{I}$ if there exists a problem instance I such that $j \in I \in \mathcal{I}$.
We call the problem space $\mathcal{I}$ *closed under deletion and insertion of legal jobs* (henceforth shortened to *closed*) if and only if we have that

$$I \in \mathcal{I}_{\text{all}} \text{ and each } j \in I \text{ is legal in } \mathcal{I} \Longrightarrow I \in \mathcal{I}.$$

In other words, a problem space is closed if and only if any problem instance $I \in \mathcal{I}_{\text{all}}$ consisting of only legal jobs in $\mathcal{I}$ is guaranteed to be in the problem space $\mathcal{I}$. One can also think of this as the guarantee that deleting or inserting legal jobs will not result in the modified problem instance lying outside the problem space.

As an example of a closed problem space, consider the problem space of exponentially distributed jobs, $\mathcal{I}_e = \{I \in \mathcal{I}_{\text{all}} | \forall j = (w_j, P_j) \in I : P_j \sim \exp(\lambda_j)\}$. Here, legal jobs are those with exponentially distributed processing times. Inserting or deleting any such job from a problem instance will not yield a problem instance outside the problem space $\mathcal{I}_e$. Therefore $\mathcal{I}_e$ is closed.

A toy example of a problem space that isn't closed, is $\mathcal{I}_{\text{not closed}} := \{I \in \mathcal{I}_{\text{all}} | \text{at least half of the jobs } j \in I \text{ have a weight of } 1\}$. It is easy to see that deletion of legal jobs with weight 1 can yield a problem instance outside $\mathcal{I}_{\text{not closed}}$. Therefore, $\mathcal{I}_{\text{not closed}}$ is not closed.

For a problem instance $I \in \mathcal{I}$ containing jobs j, k, we denote with $A_{j\to k}(I)$ the event that A picks a schedule in which job j occurs before job k, given a problem instance I. Then we define the following property on A.

Definition 3.4 (Independent of irrelevant alternatives (IIA))**.** Let $\mathcal{I} \subset \mathcal{I}_{\text{all}}$ be a problem space, and A an algorithm that is well-defined on all of $\mathcal{I}$. Let j, k be two jobs $j = (w_j, P_j)$ and $k = (w_k, P_k)$ that are legal in $\mathcal{I}$. The algorithm A is called *independent of irrelevant alternatives (IIA) on* $\mathcal{I}$ if and only if for any two such jobs there exists a unique value $\Pr(A_{j\to k})$ such that for any $I \in \mathcal{I}$ with $j, k \in I$, we have that $\Pr(A_{j\to k}(I)) = \Pr(A_{j\to k})$.[2] In words, A is IIA if and only if the relative order of any jobs j, k depends only on the parameters of those two jobs, and not on the parameters of other jobs.

These definitions give us all the ingredients to prove the following lemma. The lemma allows us to express the cost-function value $\text{Cost}_I(A)$ in terms of the probabilities with which the algorithm A schedules each pair of jobs.

Lemma 3.5. Let $\mathcal{I} \subset \mathcal{I}_{\text{all}}$ be a closed problem space, and A an algorithm that is well-defined on all of $\mathcal{I}$, such that A is IIA on $\mathcal{I}$. Let $I \in \mathcal{I}$ be a problem instance. Denote with $I_{\{j,k\}}$ the problem instance consisting of only jobs j, k. Then we can rewrite the cost-function $\text{Cost}_I(A)$ as

$$\text{Cost}_I(A) = \sum_j w_j \mathbb{E}(P_j) + \sum_{j=1}^{N-1} \sum_{k=j+1}^{N} \Big(\Pr\left(A_{j\to k}(I_{\{j,k\}})\right) w_k \mathbb{E}(P_j) + \Pr\left(A_{k\to j}(I_{\{j,k\}})\right) w_j \mathbb{E}(P_k) \Big).$$

[2] In the context of this thesis, we use the letters j, k to refer to either job indices or the jobs themselves, interchangeably. It is important to note that in the definition of the IIA property, the letters j, k refer to the jobs, and not their indices. This distinction is relevant. If there are two problem instances, one containing the job j, and another containing the job j' with the same weight $w_j = w_{j'}$ and the same processing time $P_j = P_{j'}$, then we consider j to be equivalent to j', even if their indices may be different. If we instead referred to jobs by their indices in the definition of the IIA property, this would lead to issues. For example, consider an algorithm A on a problem space $\mathcal{I}$, and a problem instance $I \in \mathcal{I}$ where j is the job with index 1, and k is the job with index $N > 2$. For the algorithm A to be IIA, $\Pr(A_{j\to k})$ should be independent of parameters of other jobs. So in particular, it should remain the same if we omit all other jobs from the problem instance (denoted I_2). However, since the index of job k is $N > 2$, omitting all other jobs forces us to modify the job k by changing its index. The choice of how to newly index the jobs j, k is quite arbitrary. There is no clear argument in favor of either possible indexation, giving j index 1 and k index 2, or vice versa. This arguably arbitrary decision could in principle influence the value of $\Pr(A_{j\to k}(I_2))$, if the algorithm A depends on the job indices. This in turn means that whether or not A is IIA on $\mathcal{I}$ could depend on arbitrary indexing decisions. Defining jobs j, k by their weights and processing time distributions, rather than also their indices, avoids this issue.

Proof. First, we rewrite the cost-function as follows.

$$\begin{aligned}\mathrm{Cost}_I(A) &:= \sum_{s\in S_N} \Pr\left(A \text{ picks } s \in S_N\right) \mathbb{E}\left(\sum_j w_j C_j(s)\right) \\ &= \sum_{s\in S_N} \Pr\left(A \text{ picks } s \in S_N\right) \\ &\left(\sum_j w_j \mathbb{E}(P_j) + \sum_{j=1}^{N-1}\sum_{k=j+1}^{N}\Big(\delta(j\to k \text{ in } s) w_k \mathbb{E}(P_j) + \delta(k\to j \text{ in } s) w_j \mathbb{E}(P_k)\Big)\right).\end{aligned}$$

Here, the term $\sum_j w_j \mathbb{E}(P_j)$ corresponds to the fact that each job has to wait its own processing time before being completed. The double sum over jobs j, k goes over all pairs of jobs. For each pair, either job j has to wait for job k, or job k has to wait for job j, depending on which comes first in the schedule. If job j is first, then the Kronecker-delta function $\delta(j \to k \text{ in } s) = 1$, $\delta(k \to j \text{ in } s) = 0$, and a cost $w_k \mathbb{E}(P_j)$ is incurred. Vice versa if job k is first. Together, these sums add up to the total incurred cost under a given schedule. Then the part of the expression $\sum_{s\in S_N} \Pr\left(A \text{ picks } s \in S_N\right)$ takes the expected value over schedules for an algorithm A, thus obtaining the $\mathrm{Cost}_I(A)$.
Next, we exchange the order of the sum over the schedule and the sums over the jobs. We use the fact $\sum_{s\in S_N} \Pr\left(A \text{ picks } s \in S_N\right) = 1$ to drop the sum over schedules from the left-hand sum. Also, note that $\sum_{s\in S_N} \Pr\left(A \text{ picks } s \in S_N\right) \delta(j \to k \text{ in } s) = \Pr\left(A_{j\to k}(I)\right)$. Then we obtain that

$$\begin{aligned}\mathrm{Cost}_I(A) &= \sum_j w_j \mathbb{E}(P_j) + \sum_{j=1}^{N-1}\sum_{k=j+1}^{N}\Big(\Pr\left(A_{j\to k}(I)\right) w_k \mathbb{E}(P_j) + \Pr\left(A_{k\to j}(I)\right) w_j \mathbb{E}(P_k)\Big). \\ &= \sum_j w_j \mathbb{E}(P_j) + \sum_{j=1}^{N-1}\sum_{k=j+1}^{N}\Big(\Pr\left(A_{j\to k}(I_{\{j,k\}})\right) w_k \mathbb{E}(P_j) + \Pr\left(A_{k\to j}(I_{\{j,k\}})\right) w_j \mathbb{E}(P_k)\Big),\end{aligned}$$

where $I_{\{j,k\}}$ denotes the problem instance consisting of only jobs j, k. In the last step we have used the IIA property which yields

$$\begin{aligned}\Pr\left(A_{j\to k}(I)\right) &= \Pr\left(A_{j\to k}\right) \\ &= \Pr\left(A_{j\to k}(I_{\{j,k\}})\right).\end{aligned}$$

Note that this hinges on the fact that $\mathcal{I}$ is closed; otherwise the problem instance $I_{\{j,k\}}$, consisting of only jobs j and k, might not be in the problem space $\mathcal{I}$. Since $j, k \in I \in \mathcal{I}$, they are legal jobs. Therefore, we indeed have that $I_{\{j,k\}} \in \mathcal{I}$, which allows us to apply the IIA property. □

Lemma 3.5 allows us to write the cost-function $\mathrm{Cost}_I(A)$ in terms of the probabilities of the order of jobs in problem instances with $N = 2$. The use of this lemma will be demonstrated in the proof of Theorem 3.7.
Before we move on to this theorem, we require one last lemma, which shows that the uniformly random algorithm R is IIA on any problem space $\mathcal{I} \subset \mathcal{I}_{\mathrm{all}}$.

Lemma 3.6. Let $\mathcal{I} \subset \mathcal{I}_{\mathrm{all}}$ be a problem space. Consider the uniformly random algorithm R, which returns the uniform distribution over all schedules $s \in S_N$ for any problem instance $I \in \mathcal{I}$. Then R is IIA on $\mathcal{I}$.

Proof. Let $I, I' \in \mathcal{I}$ such that there exist jobs j, k for which $j, k \in I \cap I'$. That is to say, I and I' are two problem instances that have the same weights and processing time distributions for jobs j and k, but potentially differing parameters for other jobs. Then, since the pairwise order of any two jobs under R trivially has probability $\frac{1}{2}$, we have that $\Pr(R_{j\to k}(I)) = \frac{1}{2} = \Pr(R_{j\to k}(I')) = \Pr(R_{j\to k})$. We conclude that R is IIA on $\mathcal{I}$. □

Having prepared the necessary mathematical ingredients, we can now finally move on to the main theorem of this section.

Theorem 3.7. *Let $\mathcal{I} \subset \mathcal{I}_{all}$ be a closed problem space. Let $N \geq 2$, and denote with $\mathcal{I}_N$ the space of problem instances in $\mathcal{I}$ consisting of N jobs, i.e., $\mathcal{I}_N := \{I \in \mathcal{I} \mid |I| = N\}$. Let A, B be algorithms that are well-defined on all of $\mathcal{I}$, and IIA on $\mathcal{I}$. Then we have the equivalence*

$$A \leq_{\mathcal{I}_2} B \iff A \leq_{\mathcal{I}_N} B.$$

Or in words, for A to perform as well as B on $\mathcal{I}_N$, it is sufficient and necessary that A performs as well as B on $\mathcal{I}_2$.

Proof. $\Longrightarrow$: Suppose that $A \leq_{\mathcal{I}_2} B$. Consider a problem instance $I_N \in \mathcal{I}_N$. Then we can invoke Lemma 3.5 to write

$$\mathrm{Cost}_{I_N}(A) = \sum_j w_j \mathbb{E}(P_j) + \sum_{j=1}^{N-1} \sum_{k=j+1}^{N} \Pr\left(A_{j\to k}(I_{\{j,k\}})\right) w_k \mathbb{E}(P_j) + \Pr\left(A_{k\to j}(I_{\{j,k\}})\right) w_j \mathbb{E}(P_k). \tag{3.1}$$

From our assumption that $A \leq_{\mathcal{I}_2} B$, we have that for any problem instance $I_2 \in \mathcal{I}_2$, $\mathrm{Cost}_{I_2}(A) \leq \mathrm{Cost}_{I_2}(B)$. Writing this out, we get the inequality

$$\Pr\left(A_{1\to 2}(I_2)\right) w_2 \mathbb{E}(P_1) + \Pr\left(A_{2\to 1}(I_2)\right) w_1 \mathbb{E}(P_2) \leq \Pr\left(B_{1\to 2}(I_2)\right) w_2 \mathbb{E}(P_1) + \Pr\left(B_{2\to 1}(I_2)\right) w_1 \mathbb{E}(P_2),$$

where the inequality was achieved by subtracting $\sum_j w_j \mathbb{E}(P_j)$ from $\mathrm{Cost}_{I_2}(A)$ and $\mathrm{Cost}_{I_2}(B)$. Since $j, k \in I_N \in \mathcal{I}$ are legal jobs, and $\mathcal{I}$ is closed, we have that the problem instance $I_{\{j,k\}}$ is an element of $\mathcal{I}$. In particular, since it contains only two jobs, we have that $I_{\{j,k\}} \in \mathcal{I}_2$. Thus, we can apply the above inequality to each term in the right-hand sum in Equation 3.1 to get

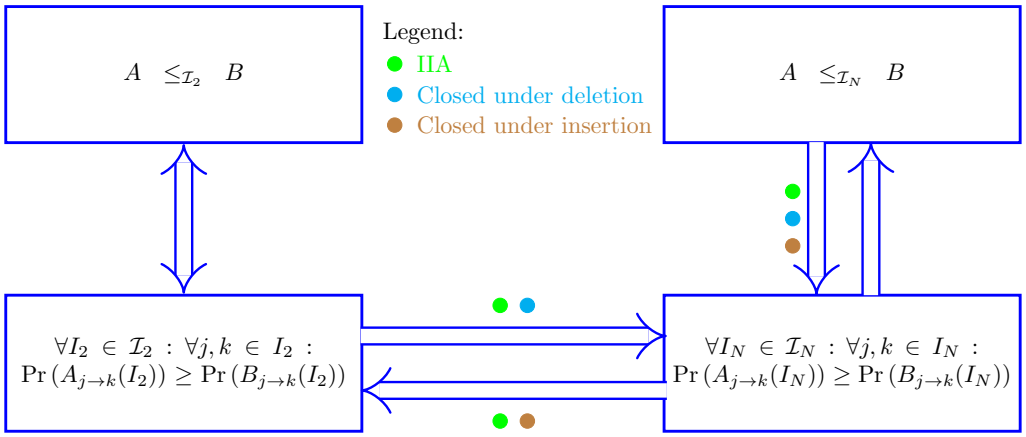

Figure 1: This diagram breaks down the statements in the proof of Theorem 3.7 into their separate steps. Each blue box contains a statement. The arrows between the boxes represent the implications between these statements. The colored dots next to the arrows indicate which assumptions are necessary for the implication to hold. Which color corresponds to which assumption is described in the legend, in the middle of the figure.

$$\begin{aligned}\text{Cost}_{I_N}(A) &\leq \sum_j w_j \mathbb{E}(P_j) + \sum_{j=1}^{N-1} \sum_{k=j+1}^{N} \Pr\left(B_{j\to k}(I_{\{j,k\}})\right) w_k \mathbb{E}(P_j) + \Pr\left(B_{k\to j}(I_{\{j,k\}})\right) w_j \mathbb{E}(P_k) \\ &= \text{Cost}_{I_N}(B).\end{aligned}$$

Here the final equality follows from again applying Lemma 3.5, now on $\text{Cost}_{I_N}(B)$. Since we showed that $\text{Cost}_{I_N}(A) \leq \text{Cost}_{I_N}(B)$ for general $I_N \in \mathcal{I}_N$, we conclude that indeed $A \leq_{\mathcal{I}_N} B$.

$\Longleftarrow$: Suppose that $A \not\leq_{\mathcal{I}_2} B$. Then there exists a problem instance $I_2 \in \mathcal{I}_2$ s.t. $\text{Cost}_{I_2}(A) > \text{Cost}_{I_2}(B)$.
Denote with $1, 2$ the jobs that are in I_2. Define I_N as the problem instance consisting of a single copy of job 1, and $N-1$ copies of job 2. Since $1, 2 \in I_2 \in \mathcal{I}$ are legal jobs, and $\mathcal{I}$ is closed, we have that the problem instance I_N is an element of $\mathcal{I}$. In particular, since it contains N jobs, we have that $I_N \in \mathcal{I}_N$.
We can invoke Lemma 3.5, and use that $\text{Cost}_{I_2}(A) > \text{Cost}_{I_2}(B)$. Then the subsequent steps are analogous to the proof in the other direction, and are therefore omitted. The end result is that $\text{Cost}_{I_N}(A) > \text{Cost}_{I_N}(B)$. From this we conclude that $A \not\leq_{\mathcal{I}_N} B$, which concludes the proof. □

In Figure 1, we see the proof of Theorem 3.7 broken down into smaller steps. The boxes correspond to statements, and the arrows between them correspond to the implications between statements. This diagram illustrates that some implications do not require the full set of assumptions that were made in Theorem 3.7. For example, for the implication

$A \leq_{\mathcal{I}_2} B \implies A \leq_{\mathcal{I}_N} B$ we do not require the problem space $\mathcal{I}$ to be closed under insertion.

3.5 Quantitative algorithm comparison using the Relative Optimality Gap

So far most of methods introduced allow for a statement of the form "under specific circumstances, some algorithm A is better than R". However, what we are lacking is an understanding of *how much* better than R an algorithm performs. This need for a more quantitative comparison is what motivates the content of this section.

Consider a problem space $\mathcal{I}$, and a problem instance $I \in \mathcal{I}$. Denote the lowest achievable cost as $L := \arg\min_s (\text{Cost}_I(s))$, and the highest achievable cost as $H := \arg\max_s (\text{Cost}_I(s))$. For any algorithm A, we then have that $L \leq \text{Cost}_I(A) \leq H$, by definition of L and H.

The first quantitative comparison that might come to mind, is bounding $\text{Cost}_I(A)$ by a constant times the optimal cost. I.e., one might consider trying to prove a result like $\text{Cost}_I(A) \leq 2L$, which would make A a 2-approximation algorithm. However, the problem is that a result of this form could be extremely weak. This is easy to realize by observing the fact that it might be the case that $H < 2L$. In such cases, results of the type just mentioned are even weaker than the tautology $\text{Cost}_I(A) \leq H$.
If we want to make a meaningful statement about the performance of an algorithm, and avoid the issue presented above, we have to consider the cost-function value of an algorithm relative to not only L, but to both L and H.

Definition 3.8. The Relative Optimality Gap (ROG) of an algorithm A on a problem instance $I \in \mathcal{I}$ is defined as

$$\text{ROG}_I(A) := \begin{cases} \frac{\text{Cost}_I(A)-L}{H-L} & \text{when } H \neq L, \\ 0 & \text{when } H = L. \end{cases}$$

Note that this means that we have $0 \leq \text{ROG}_I(A) \leq 1$. The equality $\text{ROG}_I(A) = 0$ holds if and only if A always picks an optimal schedule s with probability 1. The equality $\text{ROG}_I(A) = 1$ holds if and only if A always picks a worst possible schedule s with probability 1, and $H \neq L$.
We also generalize this definition to problem spaces, instead of problem instances. The Relative Optimality Gap of an algorithm A on a problem space $\mathcal{I}$ is defined as

$$\text{ROG}_{\mathcal{I}}(A) := \sup_{I \in \mathcal{I}} (\text{ROG}_I(A)) .$$

The value $\text{ROG}_{\mathcal{I}}(A)$ is essentially a linear re-scaling of $\text{Cost}_I(A)$ to the interval $[0, 1]$. As it turns out, the concept of the Relative Optimality Gap leans itself very naturally to comparing other algorithms to R. This will become clear from the following few results.

Lemma 3.9. Consider a problem space $\mathcal{I}$, and a problem instance $I \in \mathcal{I}$. Then either $H = L$, or $\text{ROG}_I(R) = \frac{1}{2}$.

Proof. Suppose $H \neq L$. Then we have that

$$\mathrm{ROG}_I(R) = \frac{\mathrm{Cost}_I(R) - L}{H - L}.$$

Suppose w.l.o.g. that the jobs in I are indexed such that $j < k \implies \frac{w_j}{\mathbb{E}(P_j)} \geq \frac{w_k}{\mathbb{E}(P_k)}$. That is to say, the optimal cost L is achieved by scheduling in order of the indices, and the worst cost H is achieved by scheduling in the reverse order. (This assumption is only necessary to improve readability of the expressions below.)
Then it follows that

$$\begin{aligned}
\mathrm{Cost}_I(R) - L &= \left(\sum_{j=1}^{N} w_j\mathbb{E}(P_j) - w_j\mathbb{E}(P_j)\right) + \left(\sum_{j=1}^{N-1}\sum_{k=j+1}^{N} \frac{1}{2}w_k\mathbb{E}(P_j) + \frac{1}{2}w_j\mathbb{E}(P_k) - w_k\mathbb{E}(P_j)\right) \\
&= \sum_{j=1}^{N-1}\sum_{k=j+1}^{N} \frac{1}{2}w_j\mathbb{E}(P_k) - \frac{1}{2}w_k\mathbb{E}(P_j) \\
&= \frac{1}{2}\sum_{j=1}^{N-1}\sum_{k=j+1}^{N} w_j\mathbb{E}(P_k) - w_k\mathbb{E}(P_j) \\
&= \frac{1}{2}\left(H - L\right).
\end{aligned}$$

Plugging this back into the formula for $\mathrm{ROG}_I(R)$, we reconfirm that $\mathrm{ROG}_I(R) = \frac{1}{2}$. □

The above Lemma shows that by re-scaling cost to lie in the interval $[0, 1]$, the algorithm R yields an expected cost halfway between L and H, as one would expect. $\mathrm{ROG}_I(R) = 0$ if and only if all schedules achieve the same cost, e.g. when all jobs are identical copies of one another. These observations naturally lead to the following result.

Theorem 3.10. *Let $\mathcal{I}$ be a problem space, and consider an algorithm A. Then it follows that*

$$ROG_{\mathcal{I}}(A) \leq \frac{1}{2} \iff A \leq_{\mathcal{I}} R.$$

Or in words, showing that an algorithm has a Relative Optimality Gap of $\frac{1}{2}$ or less is equivalent to showing it performs at least as well as R.

Proof. $\implies$: Suppose $\mathrm{ROG}_{\mathcal{I}}(A) \leq \frac{1}{2}$. Let $I \in \mathcal{I}$ be a problem instance. Then there are two possible cases.
Case 1: $H = L$**.** Since $L \leq \mathrm{Cost}_I(A) \leq H$, and $L \leq \mathrm{Cost}_I(R) \leq H$, it follows that $H = L = \mathrm{Cost}_I(A) = \mathrm{Cost}_I(R)$.

Case 2: $H \neq L$**.** Applying Lemma 3.9, we see that $\mathrm{ROG}_I(A) \leq \mathrm{ROG}_{\mathcal{I}}(A) \leq \frac{1}{2} = \mathrm{ROG}_I(R)$. By filling in the formula for ROG, this means that also $\mathrm{Cost}_I(A) \leq \mathrm{Cost}_I(R)$.

From these cases together, we conclude that $\mathrm{Cost}_I(A) \leq \mathrm{Cost}_I(R)$ for any problem instance $I \in \mathcal{I}$, and thus that $A \leq_{\mathcal{I}} R$.

$\Longleftarrow$: Suppose that $A \leq_{\mathcal{I}} R$. Let $I \in \mathcal{I}$ be a problem instance. Then $\mathrm{Cost}_I(A) \leq \mathrm{Cost}_I(R)$. There are two cases to consider.
Case 1: $H = L$**.** Then $\mathrm{ROG}_I(A) = 0 \leq \frac{1}{2}$.
Case 2: $H \neq L$**.** Then $\mathrm{ROG}_I(A) = \frac{\mathrm{Cost}_I(A) - L}{H - L} \leq \frac{\mathrm{Cost}_I(R) - L}{H - L} = \frac{1}{2}$, where the last equality follows from Lemma 3.9.

□

N.B.: An important point to note here is that the above result is true for R, but *not* for general algorithms A, B. In other words, it is not true that $\mathrm{ROG}_{\mathcal{I}}(A) \leq \mathrm{ROG}_{\mathcal{I}}(B) \Longleftrightarrow A \leq_{\mathcal{I}} B$. Consider the following counter-example. Suppose we have a problem space $\mathcal{I}$, and algorithms A, B s.t. $\mathrm{ROG}_{\mathcal{I}}(A) = \frac{1}{4}$ and $\mathrm{ROG}_{\mathcal{I}}(B) = \frac{1}{3}$, so $\mathrm{ROG}_{\mathcal{I}}(A) < \mathrm{ROG}_{\mathcal{I}}(B)$. However, the ROG on the problem space $\mathcal{I}$ is defined as the *supremum* over the ROG values on the problem instances $I \in \mathcal{I}$. Therefore, there might well exist a problem instance $I \in \mathcal{I}$ with $\mathrm{ROG}_I(A) = \frac{1}{4} > \frac{1}{5} = \mathrm{ROG}_I(B)$. Then $\mathrm{Cost}_I(A) > \mathrm{Cost}_I(B)$ on that problem instance, and thus $A \not\leq_{\mathcal{I}} B$.

Besides providing us with an alternative method of proving $A \leq_{\mathcal{I}} R$, Theorem 3.10 also gives us a strong intuition about what the Relative Optimality Gap means. It can be looked at as comparing an algorithm A to an algorithm R_γ. Here, R_γ is an algorithm where each pair has a probability γ of being scheduled in the correct order[3]. For $\gamma = \frac{1}{2}$, R_γ is equivalent to R, the uniformly random algorithm. But as γ increases, $\mathrm{ROG}_I(R_\gamma)$ decreases, and the algorithm performs better. The statement $\mathrm{ROG}_I(A) \leq \frac{1}{3}$ boils down to the statement "the algorithm A performs at least as well as the algorithm that picks each pairwise order correctly with a probability $\frac{2}{3}$ for each pair". This perspective on the Relative Optimality Gap will be formalized by Theorem 3.12. In order to simplify the proof of this theorem, we will first introduce a Lemma 3.11, which allows us to express the formula for the Relative Optimality Gap in a simpler way.

Lemma 3.11. Consider the ROG of an algorithm A on a problem instance $I \in \mathcal{I}$. Assume w.l.o.g. that the jobs are indexed such that $j < k$ implies $\frac{w_j}{\mathbb{E}(P_j)} \geq \frac{w_k}{\mathbb{E}(P_k)}$. Define the extra contribution to the cost of a pair of jobs $j < k \in I$, when scheduled in the incorrect order, as

[3]This property alone is not enough to define R_γ uniquely. For example, the algorithm that picks an optimal schedule with probability γ, and the reverse schedule all other times, also has this property. If we want to properly define R_γ as a generalization of R, we would do something along the following lines. For each schedule s, up to normalization, define the probability of R picking s as $\Pr(R_\gamma \text{ picks } s) \sim \gamma^{\#\text{correct pairs}(s)}(1-\gamma)^{\#\text{incorrect pairs}(s)}$. That is to say, the probability of a given schedule should be proportional to γ to the power of the number of correctly ordered pairs, times $(1-\gamma)$ to the power of the number of incorrectly ordered pairs in that schedule. To make this definition work, one would need to fix the normalization, and make sure the definition holds up when some jobs have equal priority $\frac{w_j}{\mathbb{E}(P_j)} = \frac{w_k}{\mathbb{E}(P_k)}$. This is tedious, and slightly besides the point here, so we will not work it out further.

$$D_{jk} := w_j \mathbb{E}(P_k) - w_k \mathbb{E}(P_j).$$

Then we can write the ROG of A on I as

$$\mathrm{ROG}_I(A) = \begin{cases} \sum_{j<k} \Pr\left(A_{k\to j}(I)\right) \frac{D_{jk}}{\sum_{j'<k'} D_{j'k'}} & \text{when } H \neq L, \\ 0 & \text{when } H = L. \end{cases}$$

In words, the contribution of each pair to the ROG is given by the probability that they are scheduled in the wrong order, multiplied by the normalized extra cost incurred by scheduling in the pairwise incorrect order.

Proof. We have from the definition of the ROG that

$$\mathrm{ROG}_I(A) := \begin{cases} \frac{\mathrm{Cost}_I(A)-L}{H-L} & \text{when } H \neq L, \\ 0 & \text{when } H = L. \end{cases}$$

When $H = L$, the proof is trivial. Therefore, assume $H \neq L$. Then the quantities in the formula for the ROG can be expressed as

$$\begin{aligned} L &= \sum_{j=1}^{N} w_j \mathbb{E}(P_j) + \sum_{j=1}^{N-1} \sum_{k=j+1}^{N} w_k \mathbb{E}(P_j) \\ H &= \sum_{j=1}^{N} w_j \mathbb{E}(P_j) + \sum_{j=1}^{N-1} \sum_{k=j+1}^{N} w_j \mathbb{E}(P_k) \\ \mathrm{Cost}_I(A) &= \sum_{j=1}^{N} w_j \mathbb{E}(P_j) + \sum_{j=1}^{N-1} \sum_{k=j+1}^{N} \Pr\left(A_{j\to k}(I)\right) w_k \mathbb{E}(P_j) + \Pr\left(A_{k\to j}(I)\right) w_j \mathbb{E}(P_k). \end{aligned}$$

Therefore, we can write the numerator in the expression for the ROG as

$$\begin{aligned} \mathrm{Cost}_I(A) - L &= \sum_{j=1}^{N-1} \sum_{k=j+1}^{N} \Pr\left(A_{j\to k}(I)\right) w_k \mathbb{E}(P_j) + \Pr\left(A_{k\to j}(I)\right) w_j \mathbb{E}(P_k) - w_k \mathbb{E}(P_j) \\ &= \sum_{j=1}^{N-1} \sum_{k=j+1}^{N} \left(\Pr\left(A_{j\to k}(I)\right) - 1\right) w_k \mathbb{E}(P_j) + \Pr\left(A_{k\to j}(I)\right) w_j \mathbb{E}(P_k) \\ &= \sum_{j=1}^{N-1} \sum_{k=j+1}^{N} \Pr\left(A_{k\to j}(I)\right) \left(-w_k \mathbb{E}(P_j) + w_j \mathbb{E}(P_k)\right) \\ &= \sum_{j=1}^{N-1} \sum_{k=j+1}^{N} \Pr\left(A_{k\to j}(I)\right) D_{jk}, \end{aligned}$$

where in the third line we used that $\Pr(A_{j\to k}(I)) + \Pr(A_{k\to j}(I)) = 1$.
For the denominator in the expression for the ROG, we can write

$$\begin{aligned} H - L &= \sum_{j=1}^{N-1} \sum_{k=j+1}^{N} w_j \mathbb{E}(P_k) - w_k \mathbb{E}(P_j) \\ &= \sum_{j=1}^{N-1} \sum_{k=j+1}^{N} D_{jk}. \end{aligned}$$

Combining the formula for the numerator and the denominator, we reconfirm the desired expression,

$$\mathrm{ROG}_I(A) = \sum_{j<k} \Pr(A_{k\to j}(I)) \frac{D_{jk}}{\sum_{j'<k'} D_{j'k'}}.$$

□

Theorem 3.12. *Let $\mathcal{I}$ be a problem space, and $I \in \mathcal{I}$ a problem instance. Consider an algorithm A such that for each pair of jobs $j, k \in I$, we have that*

$$\frac{w_j}{\mathbb{E}(P_j)} > \frac{w_k}{\mathbb{E}(P_k)} \implies Pr(A_{j\to k}(I)) \geq \kappa,$$

for some real number $0 \leq \kappa \leq 1$. In other words, the probability that A schedules a pair in the optimal order is at least κ for each pair of jobs. Then it follows that

$$ROG_I(A) \leq 1 - \kappa.$$

Proof. Assume w.l.o.g. that the jobs in I are indexed such that $j < k \implies \frac{w_j}{\mathbb{E}(P_j)} \geq \frac{w_k}{\mathbb{E}(P_k)}$.

If $H = L$, we have that $ROG_I(A) = 0 \leq 1 - \kappa$, trivially proving the desired inequality.
For the rest of the proof, we assume that $H \neq L$.
Using Lemma 3.11, we can then express the ROG as

$$\mathrm{ROG}_I(A) = \sum_{j<k} \Pr(A_{k\to j}(I)) \frac{D_{jk}}{\sum_{j'<k'} D_{j'k'}}.$$

Our assumption that A schedules in the pairwise correct order with at least probability κ is equivalent to assuming the probability of scheduling in the pairwise incorrect order is at most $1 - \kappa$. That is to say,

$$\frac{w_j}{\mathbb{E}(P_j)} > \frac{w_k}{\mathbb{E}(P_k)} \implies \Pr\left(A_{k\to j}\right) \leq 1-\kappa.$$

This means that on each term in the expression for the ROG, we have the inequality

$$\Pr\left(A_{k\to j}(I)\right)\frac{D_{jk}}{\sum_{j'<k'} D_{j'k'}} \leq (1-\kappa)\frac{D_{jk}}{\sum_{j'<k'} D_{j'k'}}.$$

Note that this inequality still holds for pairs with $\frac{w_j}{P_j} = \frac{w_k}{P_k}$. In such cases, $D_{jk} = 0$, so the order of the pair is irrelevant and does not affect the cost.
Plugging this inequality into the formula for $\mathrm{ROG}_I(A)$, we conclude that

$$\begin{aligned}\mathrm{ROG}_I(A) &\leq \sum_{j<k}(1-\kappa)\frac{D_{jk}}{\sum_{j'<k'} D_{j'k'}} \\ &= (1-\kappa)\sum_{j<k}\frac{D_{jk}}{\sum_{j'<k'} D_{j'k'}} \\ &= 1-\kappa.\end{aligned}$$

□

Corollary 3.13. *Theorem 3.12 can be seen as an extension of our previous results, though only in one direction. Previously, we proved Theorem 3.7, which states that under certain assumption, $A \leq_{\mathcal{I}_2} B \iff A \leq_{\mathcal{I}_N} B$. We can combine this with Theorem 3.10, which states that $ROG_{\mathcal{I}}(A) \leq \frac{1}{2} \iff A \leq_{\mathcal{I}} R$. Together, we get that under the proper assumptions,*

$$ROG_{\mathcal{I}_2}(A) \leq \frac{1}{2} \iff ROG_{\mathcal{I}_N}(A) \leq \frac{1}{2} \tag{3.2}$$

The implication from left to right is analogous to what Theorem 3.12 is stating if we fill in a value of $\kappa = \frac{1}{2}$.

Note that Theorem 3.12 does not require that A is IIA. This is because the assumption made in Theorem 3.12, that $\Pr\left(A_{j\to k}(I)|\frac{w_j}{\mathbb{E}(P_j)} > \frac{w_k}{\mathbb{E}(P_k)}\right) \geq \kappa$, concerns the probability of the pairwise order of jobs j,k on the problem instance I, and the problem instance I is of size N. If instead we started with the assumption that $\Pr\left(A_{j\to k}(I_{jk})|\frac{w_j}{\mathbb{E}(P_j)} > \frac{w_k}{\mathbb{E}(P_k)}\right) \geq \kappa$, where I_{jk} is the problem instance consisting of only jobs j,k, then we *would* need the IIA property. Namely, then we could use IIA to argue that $\Pr\left(A_{j\to k}(I_{jk})\right) = \Pr\left(A_{j\to k}\right) = \Pr\left(A_{j\to k}(I)\right)$, and then continue the proof from there.

4 Application: Analysis of Well-behaved Families of Distributions

In Section 3.4 and Section 3.5, we proved several theorems that allow us to compare the performance of algorithms on the single-sample single-machine stochastic job scheduling problem. Our next goal is to apply these theorems in answering our main research question. Namely, we want to explore what choices of the problem space $\mathcal{I}$ result in $\texttt{SWOPS} \leq_{\mathcal{I}} R$. That is to say, for which problem spaces does SWOPS perform at least as well as R? The following Lemma will help us answer this question.

Lemma 4.1. Consider the SWOPS algorithm, which schedules jobs j in descending order of value of the fraction $\frac{w_j}{p_j}$, where w_j is the weight and p_j is the sampled processing time. In case of ties, it decides the ties uniformly at random. Then SWOPS is IIA on any problem space $\mathcal{I} \subset \mathcal{I}_{\text{all}}$.

Proof. Let $\mathcal{I} \subset \mathcal{I}_{\text{all}}$. The probability of the pairwise order of two jobs under SWOPS is only dependent on their relative values of $\frac{w_j}{p_j}$. Changing the parameters of other jobs does not affect these values. Therefore, SWOPS is IIA on $\mathcal{I}$. □

By Lemma 3.6 and Lemma 4.1, we know that SWOPS and R are IIA on any problem space $\mathcal{I} \subset \mathcal{I}_{\text{all}}$. Assume for the time being that $\mathcal{I}$ is a closed problem space. Then it follows by Theorem 3.7 that the statement $\texttt{SWOPS} \leq_{\mathcal{I}} R$ is equivalent to the statement $\texttt{SWOPS} \leq_{\mathcal{I}_2} R$. Here $\mathcal{I}_2$ is defined as the problem space $\mathcal{I}_2 := \{I \in \mathcal{I} \mid |I| = 2\}$, i.e. all problem instances in $\mathcal{I}$ containing only two jobs.
In the following three sections, Section 4.1, Section 4.2, and Section 4.3, we will use this result. By making several assumptions about $\mathcal{I}$, we will give sufficient conditions on $\mathcal{I}$ such that $\texttt{SWOPS} \leq_{\mathcal{I}_2} R$ holds, and by extension that $\texttt{SWOPS} \leq_{\mathcal{I}} R$.
In order to show results about the probability of the pairwise order of jobs, we will need the following lemma.

Lemma 4.2. Let $I_2 \in \mathcal{I}_{\text{all}}$ be a problem instance with $N = 2$ jobs $1, 2$. Assume their processing time distributions have probability density functions (PDFs) $f_1(x)$ and $f_2(x)$ respectively. Then the probability that SWOPS schedules job 1 before job 2 is given by the expression

$$\Pr\left(\texttt{SWOPS}_{1\to 2}(I_2)\right) = \int_0^\infty dy\ f_2(y) \int_0^{\frac{w_1}{w_2} y} dx\ f_1(x).$$

Proof. $f_1(x)$ and $f_2(x)$ are two independent PDFs. It follows that the probability of sampling P_1 to be in the interval $[a, b]$ and P_2 in the interval $[c, d]$ is given by

$$\Pr\left(a \leq p_1 \leq b, c \leq p_2 \leq d\right) = \int_c^d dy\ f_2(y) \int_a^b dx\ f_1(x).$$

By definition, SWOPS will schedule job 1 before job 2 if and only if the samples $p_1 \sim P_1$ and $p_2 \sim P_2$ satisfy the inequality

$$\begin{aligned} &\frac{w_1}{p_1} \geq \frac{w_2}{p_2} \\ \iff &p_1 \leq \frac{w_1}{w_2} p_2. \end{aligned}$$

This corresponds with having the samples in the intervals $0 \leq p_2 \leq \infty$ and $0 \leq p_1 \leq \frac{w_1}{w_2} p_2$. Plugging these into our integral bounds gives the desired expression,

$$\Pr\left(\texttt{SWOPS}_{1\to 2}(I_2)\right) = \int_0^\infty dy\ f_2(y) \int_0^{\frac{w_1}{w_2} y} dx\ f_1(x).$$

□

Note that whenever we apply this lemma, we are always implicitly assuming that each distribution we consider has a corresponding probability density function (PDF). This means that our proofs are technically not valid for distributions without PDFs, such as finitely discrete distributions. This is a shortcoming that can be fixed, and will also be discussed in Section 5.2.

4.1 Symmetric processing time distributions

In this section, we will first motivate why one might expect the SWOPS algorithm to perform well when all processing time distributions of jobs are symmetric. Then we will formally define what we mean with symmetric distributions, and prove that SWOPS performs at least as well as R in such cases.

We already know that when all samples are exactly equal to the expected processing times, SWOPS achieves an optimal cost, as SWOPS then just boils down to the same algorithm applied in the Rothkopf [3] paper, which schedules jobs in non-decreasing order of $\frac{w_j}{\mathbb{E}(P_j)}$. However, in Section 3.3, we saw that there are distributions where with high probability, we sample a processing time that is very different from the expected processing time. We could leverage this fact to construct an example where SWOPS performed significantly worse than R.
This is where the assumption of symmetry comes in. If the distributions are symmetric, for every time we sample significantly lower than the expected processing time, there is equal probability of sampling an equal amount higher than the expected processing time. In expectation, these effects balance out. Suppose the sampled processing times p_j, p_k for jobs j, k follow the inequality $p_j < p_k$. In fact, when the distributions are symmetric, this knowledge still makes it more probable that for the expected processing times, we will also have that $\mathbb{E}(P_j) < \mathbb{E}(P_k)$, even if p_j, p_k are very different from $\mathbb{E}(P_j)$ and $\mathbb{E}(P_k)$.

Definition 4.3. Let P be a distribution over $\mathbb{R}_{\geq 0}$ with expected value $E := \mathbb{E}(P)$ and probability density function $f(x)$. We say that P is a symmetric distribution if and only if the following two conditions hold:

1. $\forall 0 \leq x \leq E$, we have that $f(E+x) = f(E-x)$, and
2. $\forall x > E$, we have that $f(E+x) = 0$.

An equivalent way of thinking about this is by extending the probability density function $f(x)$ to all of $\mathbb{R}$, by defining it to be zero for all $x < 0$ and all $x > 2E$. Then the symmetry of P is the same as requiring that $f(x)$ is symmetric around E on all of $\mathbb{R}$.

Theorem 4.4. *Define $\mathcal{I}_s \subset \mathcal{I}_{all}$ as*

$$\mathcal{I}_s := \{I \in \mathcal{I}_{all} | \forall j \in I, P_j \text{ is symmetric}\}.$$

Then it holds that

$$\textit{SWOPS} \leq_{\mathcal{I}_s} R.$$

Proof. Consider a problem instance $I_2 \in \mathcal{I}_{s,2}$ consisting of jobs $1, 2$ with arbitrary weights $w_1, w_2 \in \mathbb{R}_{>0}$, and symmetric processing time distributions P_1, P_2. We denote the expected processing times with the shorthand $E_1 := \mathbb{E}(P_1)$ and $E_2 := \mathbb{E}(P_2)$. Assume without loss of generality that $\frac{w_1}{E_1} \geq \frac{w_2}{E_2}$, or equivalently $w_1 E_2 \geq w_2 E_1$. Our strategy will be to show that on such a problem instance, $\Pr(\texttt{SWOPS}_{1\to 2}(I_2)) \geq \Pr(\texttt{SWOPS}_{2\to 1}(I_2))$. That is to say, the probability of scheduling in the correct order is at least as large as the probability of scheduling in the incorrect order. From there, the desired result follows by Theorem 3.7.

Using Lemma 4.2, we can express $\Pr(\texttt{SWOPS}_{1\to 2}(I_2))$ in integral form as

$$\Pr(\texttt{SWOPS}_{1\to 2}(I_2)) = \int_0^\infty dy\ f_2(y) \int_0^{\frac{w_1}{w_2} y} dx\ f_1(x).$$

Next, we make the substitution $x' = w_2(x - E_1)$. Noting that this means that $dx = \frac{1}{w_2} dx'$ and $x = E_1 + \frac{x'}{w_2}$, this yields the expression

$$\Pr(\texttt{SWOPS}_{1\to 2}(I_2)) = \frac{1}{w_2} \int_0^\infty dy\ f_2(y) \int_{-w_2 E_1}^{w_1 y - w_2 E_1} dx'\ f_1(E_1 + \frac{x'}{w_2}).$$

Analogously, we make the substition $y' = w_1(y - E_2)$, which yields

$$\Pr(\texttt{SWOPS}_{1\to 2}(I_2)) = \frac{1}{w_1 w_2} \int_{-w_1 E_2}^\infty dy'\ f_2(E_2 + \frac{y'}{w_1}) \int_{-w_2 E_1}^{y' + w_1 E_2 - w_2 E_1} dx'\ f_1(E_1 + \frac{x'}{w_2}).$$

From the symmetry of P_2, it follows that $f_2(E_2 + \frac{y'}{w_1}) = 0$ for $y' > w_1 E_2$. Thus, we can change the upper boundary of the left-hand integral to $w_1 E_2$, since this only excludes a

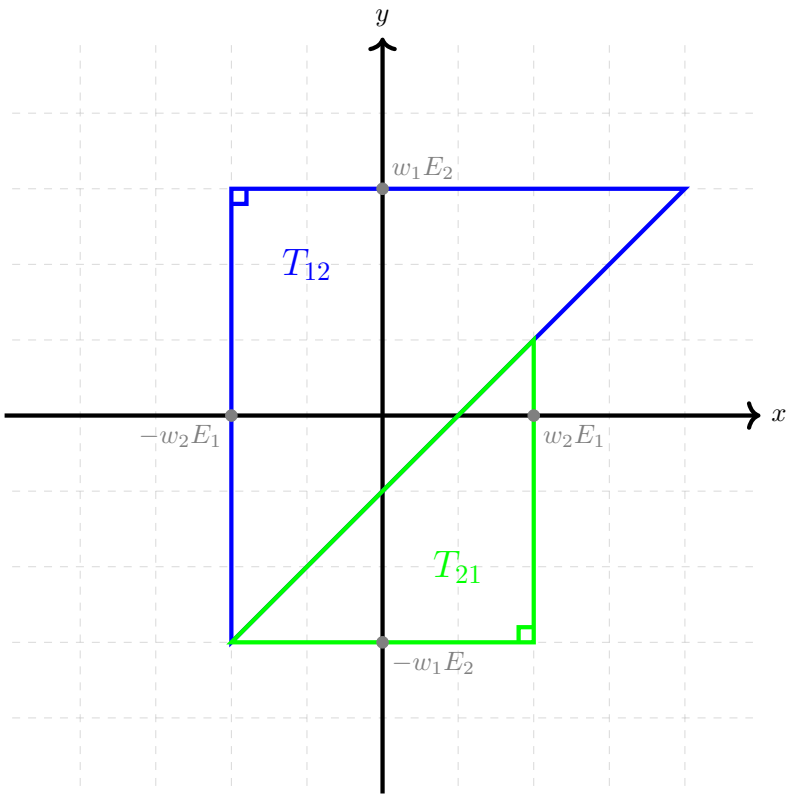

Figure 2: This figure shows two triangles representing the integral domains in the expression for the probabilities of SWOPS scheduling in each pairwise order. Triangle T_{12}, in blue, corresponds to the integral domain in the expression for $\Pr(\texttt{SWOPS}_{1\to 2}(I_2))$. Triangle T_{21}, in green, corresponds to the integral domain in the expression for $\Pr(\texttt{SWOPS}_{2\to 1}(I_2))$.

part of the domain where the integrand is zero. Relabeling x' as x and y' as y for ease of notation, we are left with the formula

$$\Pr(\texttt{SWOPS}_{1\to 2}(I_2)) = \frac{1}{w_1 w_2}\int_{-w_1E_2}^{w_1E_2} dy\; f_2(E_2+\frac{y}{w_1})\int_{-w_2E_1}^{y+w_1E_2-w_2E_1} dx\; f_1(E_1+\frac{x}{w_2}). \quad (4.1)$$

Analogously, we can derive a similar formula for $\Pr(\texttt{SWOPS}_{2\to 1}(I_2))$,

$$\Pr(\texttt{SWOPS}_{2\to 1}(I_2)) = \frac{1}{w_1 w_2}\int_{-w_2E_1}^{w_2E_1} dx\; f_1(E_1+\frac{x}{w_2})\int_{-w_1E_2}^{x+w_2E_1-w_1E_2} dy\; f_2(E_2+\frac{y}{w_1}). \quad (4.2)$$

These final two expressions can be interpreted as surface integrals in $\mathbb{R}^2$, with Cartesian coordinates x and y. The integration domain in both cases is an isosceles right-angled triangle.

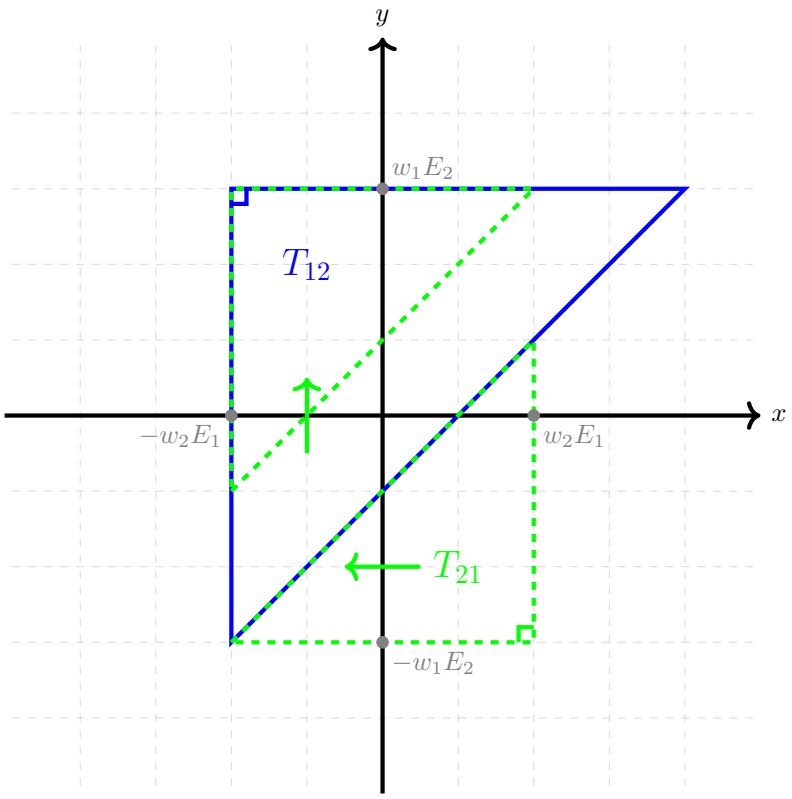

Figure 3: This figure shows two triangles representing the integral domains in the expression for the probabilities of SWOPS scheduling in each pairwise order. Mirroring either triangle through either axis does not change the values of the integrals. By mirroring T_{21} once in each axis, it aligns with T_{12}. We see that the diagonal strip where the triangles do not overlap corresponds to the difference in the integral domains, proving that $\Pr(\texttt{SWOPS}_{1\to2}(I_2)) \geq \Pr(\texttt{SWOPS}_{2\to1}(I_2))$.

These triangles are shown in Figure 2. We denote the triangle corresponding to $\Pr(\texttt{SWOPS}_{1\to2}(I_2))$ with T_{12}, and analogously for T_{21}. Starting at the bottom left and going counterclockwise, the triangle T_{12} has corners with coordinates $(-w_2E_1, -w_1E_2), (2w_1E_2 - w_2E_1, w_1E_2), (-w_2E_1, w_1E_2)$ respectively. Similarly, the triangle T_{21} has corners with coordinates $(-w_2E_1, -w_1E_2), (w_2E_1, -w_1E_2), (w_2E_1, 2w_2E_1 - w_1E_2)$.
The symmetry properties of P_1 and P_2 can be interpreted as meaning that the integrals over the triangles remain the same after reflection through either of the coordinate axes. Mirroring the triangle T_{21} corresponding to $\Pr(\texttt{SWOPS}_{2\to1}(I_2))$ twice, once in the x-axis, and once in the y-axis, perfectly aligns its right-angle corner with the right-angle corner of the triangle T_{12} corresponding to $\Pr(\texttt{SWOPS}_{1\to2}(I_2))$. This is illustrated in Figure 3. The assumption that $w_1E_2 \geq w_2E_1$ corresponds to the fact that in the figure, triangle T_{21} has shorter side-lengths than triangle T_{12}. This means the domains of the two integrals differ precisely by the diagonal strip seen in the figure, and since the integrand is non-negative we see that $\Pr(\texttt{SWOPS}_{1\to2}(I_2))$ must be the greater of the two. A more formal version of

this mirroring process is done in the Appendix, Section B. □

4.2 Translated processing time distributions

In this section, we will first motivate why one might expect the SWOPS algorithm to perform well when all processing time distributions are translated versions of the same underlying distribution. Then we will formally define what we mean with these translated distributions, and prove that SWOPS performs at least as well as R in these cases, assuming all weights are identical as well. Finally, we show that the assumption of equal weights is necessary by providing a counter-example in the case of unequal weights.

As also mentioned at the start of Section 4.1, the problem we face when using SWOPS is that potentially, with high probability the sampled processing times will be very different from the expected processing times. Then our inaccurate information about the processing times might lead SWOPS to make bad scheduling decisions.
This is remedied by the assumption that all jobs have the same processing time distributions around potentially different means. Suppose that we sample job j, k to have processing times $p_j < p_k$. If we know the distributions are the same up to a constant shift, this knowledge means it is still more probable that $\mathbb{E}(P_j) < \mathbb{E}(P_k)$, rather than the other way around. Therefore, if the weights don't interfere with this, we can still expect SWOPS to perform at least as well as R.
With this motivation, we will next make this result more precise and rigorous.

Definition 4.5 ($\mathcal{I}_{g-t}$)**.** Let g be the PDF of a processing time distribution over $\mathbb{R}_{\geq 0}$. Extend its definition to negative numbers by stating that $g(x) := 0\ \forall x < 0$. Define $\mathcal{I}_{g-t} \subset \mathcal{I}_{\text{all}}$ as

$$\mathcal{I}_{g-t} := \{I \in \mathcal{I}_{\text{all}} | \forall j \in I, P_j \text{ has a PDF } f_{a_j}(x) := g(x - a_j), \text{ where } a_j \in \mathbb{R}_{\geq 0}\}.$$

This is the set of problem instances where all processing time distributions are translated versions of the same underlying distribution given by $g(x)$. The translation distance for job j is given by a_j. For a problem instance in this space, one can think of it as the case where all jobs have the same underlying processing time distribution, but centered around different means. Note that the expected processing time is given by $\mathbb{E}(P_j) = \mathbb{E}(g(x)) + a_j$.

Theorem 4.6. *Let g be the PDF of a processing time distribution over $\mathbb{R}_{\geq 0}$. Let $\mathcal{I} \subset \mathcal{I}_{all}$ be the problem space $\mathcal{I} := \{I \in \mathcal{I}_{g-t} \mid \forall j, k \in I, w_j = w_k\}$. That is, we consider the problem space where all jobs have equal weights, and their processing time distributions are identical up to having different means.*
Then it holds that

$$\textit{SWOPS} \leq_{\mathcal{I}} R.$$

Proof. Our strategy will be to show that SWOPS $\leq_{\mathcal{I}_2} R$. It can easily be verified that $\mathcal{I}$ is closed under deletion of legal jobs. From Lemma 3.6 and Lemma 4.1 we know that

SWOPS and R are IIA. By applying Theorem 3.7, we then have that for any $N > 2$, $\texttt{SWOPS} \leq_{\mathcal{I}_2} R \implies \texttt{SWOPS} \leq_{\mathcal{I}_N} R$, and we are done. Note that $\mathcal{I}$ is *not* closed under insertion, but since we only require the implication in one direction for this proof, this is not a problem. See also Figure 1 for why closedness under insertion is not required.

Consider a two-job problem instance $I_2 \in \mathcal{I}_2$. Without loss of generality, it consists of jobs $1, 2$ with $w_1 = w_2$, and processing time distributions given by the PDFs $f_1 = g(x)$ and $f_2 = g(x - a)$ respectively. Here $a \geq 0$ is some constant extra processing time that job 2 incurs compared to job 1. Note that $\mathbb{E}(P_2) = a + \mathbb{E}(P_1)$.
Then we can write $\Pr(\texttt{SWOPS}_{1\to 2}(I_2))$ and $\Pr(\texttt{SWOPS}_{2\to 1}(I_2))$ as integral expressions.

$$\begin{aligned}
\Pr(\texttt{SWOPS}_{1\to 2}(I_2)) &= \int_0^\infty dy\ g(y-a) \int_0^{\frac{w_1}{w_2}y} dx\ g(x) \\
&= \int_{-a}^\infty dy'\ g(y') \int_0^{y'+a} dx\ g(x) \\
&= \int_0^\infty dy'\ g(y') \int_0^{y'+a} dx\ g(x),
\end{aligned}$$

where first we have made the substitution $y' = y - a$, and then we changed the left-hand integral boundary by using $g(y') := 0\ \forall y' < 0$. Going through the same steps for $\Pr(\texttt{SWOPS}_{2\to 1}(I_2))$ yields the expression

$$\begin{aligned}
\Pr(\texttt{SWOPS}_{2\to 1}(I_2)) &= \int_0^\infty dy\ g(y) \int_0^{\frac{w_1}{w_2}y} dx\ g(x-a) \\
&= \int_0^\infty dy\ g(y) \int_{-a}^{y-a} dx'\ g(x') \\
&= \int_0^\infty dy\ g(y) \int_0^{y-a} dx'\ g(x').
\end{aligned}$$

Consider the difference between these two expressions, $\Pr(\texttt{SWOPS}_{1\to 2}(I_2)) - \Pr(\texttt{SWOPS}_{2\to 1}(I_2))$.
Then we have that

$$\begin{aligned}
\Pr(\texttt{SWOPS}_{1\to 2}(I_2)) - \Pr(\texttt{SWOPS}_{2\to 1}(I_2)) &= \left(\int_0^\infty dy\ g(y) \int_0^{y+a} dx\ g(x)\right) - \left(\int_0^\infty dy\ g(y) \int_0^{y-a} dx\ g(x)\right) \\
&= \int_0^\infty dy\ g(y) \left(\int_0^{y+a} dx\ g(x) - \int_0^{y-a} dx\ g(x)\right) \\
&= \int_0^\infty dy\ g(y) \int_{y-a}^{y+a} dx\ g(x) \\
&\geq 0.
\end{aligned}$$

We see that SWOPS schedules job 1 before job 2 with at least as high probability as scheduling job 2 before job 1. Job 2 is identical to job 1, except it incurs an extra constant processing time of $a \geq 0$. Therefore, in terms of the cost-function, it is better or

equally good to schedule job 1 before job 2. We conclude that $\text{Cost}_{I_2}(\texttt{SWOPS}) \leq \text{Cost}_{I_2}(R)$. Since this holds for general $I_2 \in \mathcal{I}_2$, we conclude that indeed $\texttt{SWOPS} \leq_{\mathcal{I}_2} R$.

□

Note that the assumption of all weights being equal is necessary here. This can be seen by considering the following counterexample.
Consider the problem instance $I \in \mathcal{I}_{g-t}$ defined by $N = 2$, $w_1 = 1$, and $w_2 = 2$. The processing time distributions are given by

$$P_1 = \begin{cases} a, & \text{with probability } 1 - \frac{1}{M} \\ M^2, & \text{with probability } \frac{1}{M} \end{cases}$$
$$P_2 = \begin{cases} 2a + \epsilon, & \text{with probability } 1 - \frac{1}{M} \\ M^2 + a + \epsilon, & \text{with probability } \frac{1}{M}. \end{cases}$$

One can easily verify that the PDF of P_2 is obtained by translating the PDF of P_1 to the right by $a + \epsilon$.
We will now calculate the relevant terms in the cost-function to determine the optimal schedule. We have that $w_2\mathbb{E}(P_1) = 2M + 2a - \frac{2a}{M}$, and $w_1\mathbb{E}(P_2) = M + 2a - \frac{a}{M} + \epsilon$. By assuming $M \gg a$, $M \gg 1$ and $M \gg \epsilon$, we can estimate these as $w_2\mathbb{E}(P_1) = 2M + o(M)$ and $w_1\mathbb{E}(P_2) = M + o(M)$. The terms differ by roughly a factor 2. From this we conclude that the ideal schedule puts job 2 before job 1, as this incurs the smaller of the two cost terms, $w_1\mathbb{E}(P_2) \approx M$ instead of $w_2\mathbb{E}(P_1) \approx 2M$.
However, with probability $(1 - \frac{1}{M})^2$, the SWOPS algorithm will sample $p_1 = a$, and $p_2 = 2a + \epsilon$. Then we have that $\frac{w_2}{p_2} = \frac{2}{2a+\epsilon} < \frac{1}{a} = \frac{w_1}{p_1}$. So with probability close to 1, SWOPS will schedule job 1 before job 2. This means it will achieve a cost-function value significantly higher than that achieved by R.

4.3 Scaled processing time distributions

In Section 4.2, we saw that for processing time distributions that are identical up to a constant shift SWOPS performs at least as well as R, assuming identical weights. This begs the question whether we can also show something similar for the case where all distributions are identical up to a linear scaling factor.
In this section we consider this question. First we will make a more precise definition of this scaling property. Then we prove that when all processing time distributions are scaled versions of the same underlying distribution, it follows that SWOPS performs at least as well as R.

Definition 4.7. Let $g(x)$ be a PDF over $\mathbb{R}_{\geq 0}$ with an expected value of 1. Define $\mathcal{I}_{g-s} \subset \mathcal{I}_{\text{all}}$ as follows.

$$\mathcal{I}_{g-s} := \{I \in \mathcal{I}_{\text{all}} \mid \forall j \in I, P_j \text{ has a PDF } f_{\lambda_j}(x) := \lambda_j g(\lambda_j x), \text{ for some } \lambda_j \in \mathbb{R}_{>0}\}.$$

That is to say, $\mathcal{I}_{g-s}$ is the set of problem instances where all processing time distributions are linearly scaled versions of the same underlying distribution given by $g(x)$. Note that this includes as a special case exponentially distributed processing times with different rates λ_j.

Lemma 4.8. Let $I_g \in \mathcal{I}_{g-s}$ be a problem instance. Consider job $j \in I$. Then the expected processing time $\mathbb{E}(P_j) = \frac{1}{\lambda_j}$.

Proof.

$$\begin{aligned}
\mathbb{E}(P_j) &= \int_0^\infty dx\ x f_{\lambda_j}(x) \\
&= \lambda_j \int_0^\infty dx\ x g(\lambda_j x) \\
&= \lambda_j \int_0^\infty \frac{dx'}{\lambda_j}\ \frac{x'}{\lambda_j} g(x') \\
&= \frac{1}{\lambda_j} \int_0^\infty dx'\ x' g(x') \\
&= \frac{1}{\lambda_j}.
\end{aligned}$$

In the last step of the proof we used that $g(x)$ has expectation value 1.

□

Theorem 4.9. *Let $g(x)$ be a PDF over $\mathbb{R}_{\geq 0}$ with an expected value of 1. Consider $\mathcal{I}_{g-s}$ as defined above. Then the* ***SWOPS*** *algorithm performs at least as well as R on $\mathcal{I}_{g-s}$. That is to say,* ***SWOPS*** $\leq_{\mathcal{I}_{g-s}} R$.

Proof. By Lemma 3.6, R is IIA on $\mathcal{I}_{g-s}$. Similarly, by Lemma 4.1, the SWOPS algorithm is IIA on $\mathcal{I}_{g-s}$.

Denote $\mathcal{I}_{g\to s,N} := \{I \in \mathcal{I}_{g-s} |\ |I| = N\}$, i.e., the space of problem instances in $\mathcal{I}_{g-s}$ consisting of N jobs.
Suppose we were able to show that SWOPS $\leq_{\mathcal{I}_{g\to s,2}} R$, i.e., that SWOPS performs as well as R for problem instances with only two jobs. Then by the application of Theorem 3.7, it follows that for any $N \geq 2$, SWOPS $\leq_{\mathcal{I}_{g\to s,N}} R$. Note that we can write $\mathcal{I}_{g-s} = \bigcup_{N\geq 2} \mathcal{I}_{g\to s,N}$. Thus we then also have that SWOPS $\leq_{\mathcal{I}_{g-s}} R$, the desired result. We conclude that it is sufficient to prove that SWOPS $\leq_{\mathcal{I}_{g\to s,2}} R$.

Let $I_2 \in \mathcal{I}_{g\to s,2}$ be a problem instance consisting of jobs 1 and 2. Assume without loss of generality that $\frac{w_1}{\mathbb{E}(P_1)} \geq \frac{w_2}{\mathbb{E}(P_2)}$. Applying Lemma 4.8, which states that $\mathbb{E}(P_j) = \frac{1}{\lambda_j}$, we see that this is equivalent to $w_1\lambda_1 \geq w_2\lambda_2$.

We can write the probability of SWOPS scheduling in the order 1 before 2 as

$$\begin{aligned}\Pr\left(\texttt{SWOPS}_{1\to 2}(I_2)\right) &= \int_0^\infty dy\ f_{\lambda_2}(y) \int_0^{\frac{w_1}{w_2}y} dx\ f_{\lambda_1}(x) \\ &= \lambda_1\lambda_2 \int_0^\infty dy\ g(\lambda_2 y) \int_0^{\frac{w_1}{w_2}y} dx\ g(\lambda_1 x) \\ &= \int_0^\infty dy'\ g(y') \int_0^{\frac{w_1\lambda_1}{w_2\lambda_2}y'} dx'\ g(x'),\end{aligned}$$

where the last step was done by making substitutions $y' = \lambda_2 y$ and $x' = \lambda_1 x$. Analogously, we can write the probability for scheduling in the reverse order as

$$\Pr\left(\texttt{SWOPS}_{2\to 1}(I_2)\right) = \int_0^\infty dy'\ g(y') \int_0^{\frac{w_2\lambda_2}{w_1\lambda_1}y'} dx'\ g(x').$$

We will rewrite $x' = x$ and $y' = y$ for ease of notation. Consider the difference $\Pr\left(\texttt{SWOPS}_{1\to 2}(I_2)\right) - \Pr\left(\texttt{SWOPS}_{2\to 1}(I_2)\right)$. Then we have that

$$\begin{aligned}\Pr\left(\texttt{SWOPS}_{1\to 2}(I_2)\right) - \Pr\left(\texttt{SWOPS}_{2\to 1}(I_2)\right) &= \int_0^\infty dy\ g(y) \left[\int_0^{\frac{w_1\lambda_1}{w_2\lambda_2}y} dx\ g(x) - \int_0^{\frac{w_2\lambda_2}{w_1\lambda_1}y} dx\ g(x)\right] \\ &= \int_0^\infty dy\ g(y) \int_{\frac{w_2\lambda_2}{w_1\lambda_1}y}^{\frac{w_1\lambda_1}{w_2\lambda_2}y} dx\ g(x).\end{aligned}$$

The integrand is non-negative everywhere. Also, the lower integral boundaries are less than the upper integral boundaries, since we have that $w_1\lambda_1 \geq w_2\lambda_2$. Therefore we conclude that $\Pr\left(\texttt{SWOPS}_{1\to 2}(I_2)\right) - \Pr\left(\texttt{SWOPS}_{2\to 1}(I_2)\right) \geq 0$, i.e. $\Pr\left(\texttt{SWOPS}_{1\to 2}(I_2)\right) \geq \Pr\left(\texttt{SWOPS}_{2\to 1}(I_2)\right)$. In other words, SWOPS will schedule job 1 before job 2 more likely than the other way around. Because SWOPS schedules these two jobs in the correct order more likely than in the wrong order, we have that $\text{Cost}_{I_2}(\texttt{SWOPS}) \leq \text{Cost}_{I_2}(R)$. Specifically, when $\frac{w_1}{\mathbb{E}(P_{\lambda_1})} \neq \frac{w_2}{\mathbb{E}(P_{\lambda_2})}$ we even have that $\text{Cost}_{I_2}(\texttt{SWOPS}) < \text{Cost}_{I_2}(R)$. Since this holds for any $I_2 \in \mathcal{I}_{g\to s,2}$, we conclude that $\texttt{SWOPS} \leq_{\mathcal{I}_{g\to s,2}} R$, concluding the proof.

□

4.4 Exponential processing time distributions with α-far priorities

So far, we've only been able to show that for some subclasses $\mathcal{I}$ of $\mathcal{I}_{\text{all}}$, we have that $\texttt{SWOPS} \leq_{\mathcal{I}} R$. Or equivalently, as shown in Theorem 3.10, that $\text{ROG}_{\mathcal{I}}(\texttt{SWOPS}) \leq \frac{1}{2}$. In this section we will try to explore a set of assumptions that allows us to prove a stronger result, using the Relative Optimality Gap outlined in Section 3.5. The goal is to prove a result of the form $\text{ROG}_{\mathcal{I}}(\texttt{SWOPS}) \leq C(\mathcal{I})$, where $0 \leq C(\mathcal{I}) < \frac{1}{2}$ is some constant depending only on the choice of $\mathcal{I}$.

Define $\mathcal{I}_e := \{I \in \mathcal{I}_{\text{all}} \mid \forall j \in I, P_j \text{ has a PDF } f_j(x) := \lambda_j e^{-\lambda_j x} \text{ for some } \lambda_j > 0\}$. That is to say, $\mathcal{I}_e$ is the space of problem instances where all jobs have exponentially distributed processing times. Note that we have that $\mathcal{I}_e = \mathcal{I}_{g-s}$ for the choice $g(x) = e^{-x}$, so $\mathcal{I}_e$ is just a special case. It then already follows from Section 4.3 that $\texttt{SWOPS} \leq_{\mathcal{I}_e} R$, or equivalently that $\text{ROG}_{\mathcal{I}_e}(\texttt{SWOPS}) \leq \frac{1}{2}$. On $\mathcal{I}_e$ however, we can derive an explicit formula for the probability of pairwise order under a schedule picked by SWOPS, which also leads to an improved result. This explicit formula is derived in the following lemma.

Lemma 4.10. Let $I \in \mathcal{I}_e$ be a problem instance with $N \geq 2$ jobs. Consider jobs $j, k \in I$. Then it follows that

$$\Pr\left(\texttt{SWOPS}_{j \to k}(I)\right) = \frac{\pi_j}{\pi_j + \pi_k},$$

where $\pi_j = w_j \lambda_j = \frac{w_j}{\mathbb{E}(P_j)}$ is the priority of job j.

Proof. Since $I \in \mathcal{I}_e$, there exists a value $\lambda_j > 0$ such that P_j has a PDF $f_j(x) = \lambda_j e^{-\lambda_j x}$, and idem for job k. From Lemma 4.1 we know that SWOPS is IIA on $\mathcal{I}_e$. This means that instead of using the full problem instance I, we can consider the problem instance I_{jk} consisting of only jobs j, k. Then we can write the probability for the pairwise order as

$$\begin{aligned}
\Pr\left(\texttt{SWOPS}_{j \to k}(I)\right) &= \Pr\left(\texttt{SWOPS}_{j \to k}(I_{jk})\right) \\
&= \int_0^\infty dy\ f_k(y) \int_0^{\frac{w_j}{w_k} y} dx\ f_j(x) \\
&= \int_0^\infty dy\ \lambda_k e^{-\lambda_k y} \int_0^{\frac{w_j}{w_k} y} dx\ \lambda_j e^{-\lambda_j x} \\
&= \int_0^\infty dy'\ e^{-y'} \int_0^{\frac{w_j}{\lambda_k w_k} y'} dx\ \lambda_j e^{-\lambda_j x} \\
&= \int_0^\infty dy'\ e^{-y'} \int_0^{\frac{\lambda_j w_j}{\lambda_k w_k} y'} dx'\ e^{-x'},
\end{aligned}$$

where in the last two steps, we used the substitutions $y' = \lambda_k y$ and $x' = \lambda_j x$. Explicitly evaluating this integral yields the formula

$$\begin{aligned}
\Pr\left(\texttt{SWOPS}_{j\to k}(I_{jk})\right) &= \int_0^\infty dy'\ e^{-y'}\left[-e^{-x'}\right]_{x'=0}^{x'=\frac{\lambda_j w_j}{\lambda_k w_k}y'} \\
&= \int_0^\infty dy'\ e^{-y'}\left[-e^{-\frac{\lambda_j w_j}{\lambda_k w_k}y'}+1\right] \\
&= \left[\left(\frac{\lambda_j w_j}{\lambda_k w_k}+1\right)^{-1} e^{-\left(\frac{\lambda_j w_j}{\lambda_k w_k}+1\right)y'} - e^{-y'}\right]_{y'=0}^{y'=\infty} \\
&= \left[0-\left(\left(\frac{\lambda_j w_j}{\lambda_k w_k}+1\right)^{-1}-1\right)\right] \\
&= 1-\left(\frac{\lambda_j w_j+\lambda_k w_k}{\lambda_k w_k}\right)^{-1} \\
&= \frac{\lambda_j w_j}{\lambda_j w_j+\lambda_k w_k} \\
&= \frac{\pi_j}{\pi_j+\pi_k}.
\end{aligned}$$

□

One of the things we see from this lemma is that SWOPS schedules jobs in the pairwise correct order with a high probability when their priorities are far apart. That is to say, when $\pi_j \gg \pi_k$, we have that $\Pr\left(\texttt{SWOPS}_{j\to k}(I)\right) \approx 1$. This motivates the idea of considering a problem space where the priorities of jobs are far apart, which leads to an improved ROG-bound.

Definition 4.11 (α-far, α-close)**.** Let $\alpha > 1$, and let $I \in \mathcal{I}_{\text{all}}$ be some problem instance. Consider two distinct jobs $j, k \in I$. We say that j, k are *α-far* if and only if at least one of the following holds

$$\max\{\frac{\pi_j}{\pi_k}, \frac{\pi_k}{\pi_j}\} = \max\{\frac{w_j\mathbb{E}(P_k)}{w_k\mathbb{E}(P_j)}, \frac{w_k\mathbb{E}(P_j)}{w_j\mathbb{E}(P_k)}\} \geq \alpha \text{ if } \mathbb{E}(P_j) \neq 0 \text{ and } \mathbb{E}(P_k) \neq 0,$$

OR

$$\mathbb{E}(P_j) = 0 \text{ and } \mathbb{E}(P_k) \neq 0,$$

OR

$$\mathbb{E}(P_j) \neq 0 \text{ and } \mathbb{E}(P_k) = 0,$$

where $\pi_j := \frac{w_j}{\mathbb{E}(P_j)}$ denotes the priority of a job j. In words, two jobs j, k are α-far if their priorities are at least a factor α apart. The second and third case in the case distinction are necessary to avoid problems with dividing by an expected processing time of zero. If j, k are not α-far, we say that j, k are *α-close*.

Given some $\alpha > 1$, define the problem space $\mathcal{I}_{\alpha\text{-far}} := \{I \in \mathcal{I}_{\text{all}} | \forall j \neq k \in I, j, k \text{ are } \alpha\text{-far}\}$. That is to say, $\mathcal{I}_{\alpha\text{-far}}$ is the space of problem instances where the priorities of any two jobs lie at least a factor α apart.

Consider the problem space $\mathcal{I}_{e,\alpha-far} = \mathcal{I}_e \cap \mathcal{I}_{\alpha\text{-far}}$. This is the problem space for which we will prove a quantitative performance result using the Relative Optimality Gap. Consider the following theorem.

Theorem 4.12. *Let $\alpha > 1$, and consider the problem space $\mathcal{I}_{e,\alpha\text{-}far}$ as defined above. Then it holds that*

$$ROG_{\mathcal{I}_{e,\alpha\text{-}far}}(\textit{SWOPS}) \leq \frac{1}{\alpha+1}.$$

Proof. Let $I \in \mathcal{I}_{e,\alpha\text{-far}}$, and $j,k \in I$. Assume w.l.o.g. that $\frac{w_j}{\mathbb{E}(P_j)} \geq \frac{w_k}{\mathbb{E}(P_k)}$. Our goal is to prove that $\Pr\left(\texttt{SWOPS}_{j\to k}(I) \middle| \frac{w_j}{\mathbb{E}(P_j)} > \frac{w_k}{\mathbb{E}(P_k)}\right) \geq \frac{\alpha}{\alpha+1}$, i.e. to show the probability of scheduling correctly is relatively large. Then the desired result will follow directly from Theorem 3.12. Using Lemma 4.10, we have an explicit formula for $\Pr(\texttt{SWOPS}_{j\to k}(I))$. Namely,

$$\Pr(\texttt{SWOPS}_{j\to k}(I)) = \frac{\pi_j}{\pi_j + \pi_k}.$$

Since jobs $j,k \in \mathcal{I}_{e,\alpha\text{-far}}$ are α-far, we know that $\pi_j \geq \alpha\pi_k$. Therefore it follows that

$$\begin{aligned}\Pr(\texttt{SWOPS}_{j\to k}(I_{jk})) &\geq \frac{\alpha\pi_k}{\alpha\pi_k + \pi_k} \\ &= \frac{\alpha}{\alpha+1}.\end{aligned}$$

Note that π_k can never be equal to zero, since $w_k > 0$, so the above formula doesn't break down.
Since we picked j,k as arbitrary jobs, it follows that $\Pr(\texttt{SWOPS}_{j\to k}(I)) \geq \frac{\alpha}{\alpha+1}$ for any pair of jobs in I. Thus we can apply Theorem 3.12 with $\kappa = \frac{\alpha}{\alpha+1}$ to state that

$$\begin{aligned}\mathrm{ROG}_I(\texttt{SWOPS}) &\leq 1-\kappa \\ &= 1 - \frac{\alpha}{\alpha+1} \\ &= \frac{1}{\alpha+1}.\end{aligned}$$

Since we showed that this holds for arbitrary $I \in \mathcal{I}_{e,\alpha\text{-far}}$, we conclude that $\mathrm{ROG}_{\mathcal{I}_{e,\alpha\text{-far}}}(\texttt{SWOPS}) = \sup_{I\in\mathcal{I}_{e,\alpha\text{-far}}} (\mathrm{ROG}_I(\texttt{SWOPS})) \leq \frac{1}{\alpha+1}$. □

4.5 Exponential processing time distributions with any priorities

In the previous section, we showed that for jobs with exponentially distributed processing times, the Relative Optimality Gap of SWOPS can be upper-bounded by $\frac{1}{\alpha+1}$, assuming all jobs have priorities that are at least a factor α apart. This last assumption, that the jobs have very different priorities, is very restrictive. In this section we will explore whether we can relax this assumption. To do this, we will walk through the intuition and motivation of what might happen if the priorities of some jobs are less than a factor α apart. Afterwards, in Theorem 4.14 we will make a more precise statement in this direction.

Suppose we consider some problem instance $I \in \mathcal{I}_e$ and a value $\alpha > 1$ such that some pairs of jobs in I are α-far, and some are α-close. Then intuitively, from Section 4.4 we know that SWOPS is relatively likely to schedule the α-far pairs in the pairwise correct order. This means that the α-far pairs contribute to SWOPS performing better than R.
On the other hand, for the α-close pairs, we know that $\frac{w_j}{\mathbb{E}P_j}$ and $\frac{w_k}{\mathbb{E}P_k}$ are relatively close together. This means that the value of the extra incurred cost for scheduling in the incorrect pairwise order, $w_j\mathbb{E}(P_k) - w_k\mathbb{E}(P_j)$, will be relatively small.
Combining these two arguments, it seems that even when some jobs are α-close, we should still be able to show that SWOPS is quantitatively better than R. This statement is made more precise in Theorem 4.14 below. Before we start the theorem, we first need a final piece of notation.

Definition 4.13 $(D, D_{\alpha-far}, D_{\alpha-close})$**.** Let $\alpha > 1$, and let $I \in \mathcal{I}_{\text{all}}$ be some problem instance. For ease of notation, assume w.l.o.g. that the jobs are indexed in order of high to low priority, that is to say, $j < k$ implies $\frac{w_j}{\mathbb{E}(P_j)} \geq \frac{w_k}{\mathbb{E}(P_k)}$. On this problem instance, consider the highest possible cost H and the lowest possible cost L. Then D is defined as $D := H - L$, the highest cost minus the lowest cost. Writing this out, we have that

$$D := \sum_{j=1}^{N-1} \sum_{k=j+1}^{N} w_j\mathbb{E}(P_k) - w_k\mathbb{E}(P_j).$$

Furthermore, we define $D_{\alpha-far}$ to be the difference between the highest cost and the lowest cost due to the contribution from all α-far pairs of jobs. Concretely, we have that

$$D_{\alpha-far} := \sum_{j=1}^{N-1} \sum_{k=j+1}^{N} \delta(j,k \text{ are } \alpha\text{-far}) \left(w_j\mathbb{E}(P_k) - w_k\mathbb{E}(P_j)\right).$$

Here, δ denotes the Kronecker-delta function which is 1 when j, k are α-far, and 0 when they are not. Analogously, we define $D_{\alpha-close}$ as

$$D_{\alpha-close} := \sum_{j=1}^{N-1} \sum_{k=j+1}^{N} \delta(j,k \text{ are } \alpha\text{-close}) \left(w_j\mathbb{E}(P_k) - w_k\mathbb{E}(P_j)\right).$$

Note that we have that $H - L = D = D_{\alpha-far} + D_{\alpha-close}$, by definition.

Theorem 4.14. *Let $\alpha > 1$, and let $I \in \mathcal{I}_e$ be a problem instance. Then it follows that*

$$ROG_I(\textit{SWOPS}) \leq \frac{1}{\alpha+1}\frac{D_{\alpha-far}}{D_{\alpha-far}+D_{\alpha-close}} + \frac{1}{2}\frac{D_{\alpha-close}}{D_{\alpha-far}+D_{\alpha-close}}.$$

Proof. For ease of notation, assume w.l.o.g. that the jobs are indexed in order of high to low priority, that is to say, $j < k$ implies $\frac{w_j}{\mathbb{E}(P_j)} \geq \frac{w_k}{\mathbb{E}(P_k)}$. If $H = L$ then $\mathrm{ROG}_I(\texttt{SWOPS}) = 0$, and there is nothing to prove. Therefore, from now on we assume that $H \neq L$. Then we have that $\mathrm{ROG}_I(\texttt{SWOPS}) := \frac{\mathrm{Cost}_I(\texttt{SWOPS})-L}{H-L}$. We will write out the numerator of this expression to achieve the desired inequality.

$$\begin{aligned}
\mathrm{Cost}_I(\texttt{SWOPS}) - L &= \sum_{j=1}^{N-1}\sum_{k=j+1}^{N} \Pr\left(\texttt{SWOPS}_{j\to k}(I)\right) w_k\mathbb{E}(P_j) + \Pr\left(\texttt{SWOPS}_{k\to j}(I)\right) w_j\mathbb{E}(P_k) - w_k\mathbb{E}(P_j) \\
&= \sum_{j=1}^{N-1}\sum_{k=j+1}^{N} \left(\Pr\left(\texttt{SWOPS}_{j\to k}(I)\right) - 1\right) w_k\mathbb{E}(P_j) + \Pr\left(\texttt{SWOPS}_{k\to j}(I)\right) w_j\mathbb{E}(P_k) \\
&= \sum_{j=1}^{N-1}\sum_{k=j+1}^{N} \Pr\left(\texttt{SWOPS}_{k\to j}(I)\right) \left(-w_k\mathbb{E}(P_j) + w_j\mathbb{E}(P_k)\right),
\end{aligned}$$

where in the last step we used that $\Pr\left(\texttt{SWOPS}_{k\to j}(I)\right) + \Pr\left(\texttt{SWOPS}_{j\to k}(I)\right) = 1$. From Lemma 4.10 we know that for α-far pairs of jobs with exponentially distributed processing times, the probability of scheduling in the correct order $\Pr\left(\texttt{SWOPS}_{j\to k}(I)\right) \geq \frac{\alpha}{\alpha+1}$. It follows that for such pairs of jobs the probability of scheduling in the incorrect order $\Pr\left(\texttt{SWOPS}_{k\to j}(I)\right) \leq \frac{1}{\alpha+1}$. Furthermore, for α-close jobs with exponentially distributed processing times, we know that $\Pr\left(\texttt{SWOPS}_{k\to j}(I)\right) \leq \frac{1}{2}$. This can be seen by using Lemma 4.10 and filling in $\pi_1 = \pi_2$, or alternatively by using the results from Sections 4.3 and Theorem 3.10.
Applying these inequalities to the expression for $\mathrm{Cost}_I(\texttt{SWOPS}) - L$, we obtain that

$$\begin{aligned}
\mathrm{Cost}_I(\texttt{SWOPS}) - L &= \sum_{j=1}^{N-1}\sum_{k=j+1}^{N} \Pr\left(\texttt{SWOPS}_{k\to j}(I)\right)\left(-w_k\mathbb{E}(P_j) + w_j\mathbb{E}(P_k)\right) \\
&= \sum_{j=1}^{N-1}\sum_{k=j+1}^{N} \delta(j,k \text{ are } \alpha\text{-far})\Pr\left(\texttt{SWOPS}_{k\to j}(I)\right)\left(-w_k\mathbb{E}(P_j) + w_j\mathbb{E}(P_k)\right) + \\
&\sum_{j=1}^{N-1}\sum_{k=j+1}^{N} \delta(j,k \text{ are } \alpha\text{-close})\Pr\left(\texttt{SWOPS}_{k\to j}(I)\right)\left(-w_k\mathbb{E}(P_j) + w_j\mathbb{E}(P_k)\right) \\
&\leq \sum_{j=1}^{N-1}\sum_{k=j+1}^{N} \delta(j,k \text{ are } \alpha\text{-far})\frac{1}{\alpha+1}\left(-w_k\mathbb{E}(P_j) + w_j\mathbb{E}(P_k)\right) + \\
&\sum_{j=1}^{N-1}\sum_{k=j+1}^{N} \delta(j,k \text{ are } \alpha\text{-close})\frac{1}{2}\left(-w_k\mathbb{E}(P_j) + w_j\mathbb{E}(P_k)\right) \\
&= \frac{1}{\alpha+1}D_{\alpha-far} + \frac{1}{2}D_{\alpha-close}.
\end{aligned}$$

Plugging this into the formula for $\mathrm{ROG}_I(\texttt{SWOPS})$, we conclude that

$$\begin{aligned}
\mathrm{ROG}_I(\texttt{SWOPS}) &\leq \left(\frac{1}{\alpha+1}D_{\alpha-far} + \frac{1}{2}D_{\alpha-close}\right)/(H-L) \\
&= \frac{1}{\alpha+1}\frac{D_{\alpha-far}}{D_{\alpha-far}+D_{\alpha-close}} + \frac{1}{2}\frac{D_{\alpha-close}}{D_{\alpha-far}+D_{\alpha-close}}.
\end{aligned}$$

□

This result can be seen as an interpolation between the result of Theorem 4.12, which shows a ROG-bound of $\frac{1}{\alpha+1}$ when all jobs are α-far, and the result of Theorem 4.9, which is equivalent to a ROG-bound of $\frac{1}{2}$.
On any particular problem instance I, this bound holds for any choice of α. By choosing the right value of α, the hope is that we can obtain a bound as tight as possible. There are several conflicting considerations to take into account when choosing α. On the one hand, if we pick α very large, potentially there will be very few jobs that are still α-far. Then it might be the case that $D_{\alpha-far} \ll D_{\alpha-close}$. If that happens, the right-hand term in the ROG bound will dominate, and we end up with a ROG-bound close to $\frac{1}{2}$. On the other hand, if we pick α very small, close to 1, then $\frac{1}{\alpha+1}$ will be close to $\frac{1}{2}$. Then we also end up with a bound not much better than $\frac{1}{2}$.
What the best choice for α is depends on the problem instance. It depends on the jobs, their priorities, and how far these lie apart. Since the priorities are given by the weights and the expected processing times of jobs, and the processing time distributions are only accessible through a single sample, the above bound is hard to apply in practice.

To give more intuition, next we will show two examples. One where the above bound yields a strong result, and one where it does not.

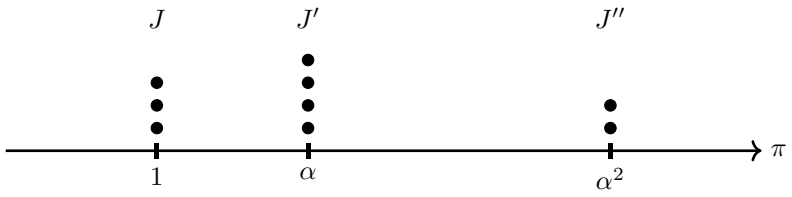

Figure 4: This figure shows a number line for the values of the priorities $\pi_j = \frac{w_j}{\mathbb{E}(P_j)}$ of jobs. The jobs are indicated with dots at their corresponding priority value. We see that the assumptions in Corollary 4.15 are such that the jobs are contained in "piles". All jobs within a pile have the same priority, and are α-far from the jobs in other piles.

Corollary 4.15. *Consider a problem instance $I \in \mathcal{I}_e$. Suppose we collect jobs j into sets J, such that all jobs within a set J have equal priorities $\pi_j = \frac{w_j}{\mathbb{E}(P_j)}$. Furthermore suppose that there are at least two such distinct sets, and that the priorities for different sets are at least a factor α apart. That is to say, if $j \in J$ and $j' \in J' \neq J$, then j and j' are α-far. Then*

$$ROG_I(\texttt{SWOPS}) \leq \frac{1}{\alpha + 1}.$$

Proof. Consider Figure 4. It shows a number line, with dots representing each job. The dots are placed along the number line at the value corresponding to their priority. We see that our assumptions boil down to the fact that the jobs are arranged in "piles", and that each pair of jobs from different piles is at least a factor α apart.
Consider $D_{\alpha-close}$, the contribution to the cost from the pairs of jobs that are α-close to one another. This is given by the expression

$$\begin{aligned} D_{\alpha-close} &:= \sum_{j=1}^{N-1} \sum_{k=j+1}^{N} \delta(j,k \text{ are } \alpha\text{-close}) \left(w_j \mathbb{E}(P_k) - w_k \mathbb{E}(P_j)\right) \\ &= 0. \end{aligned}$$

The equality to zero follows from the fact that whenever j, k are α-close, then $\pi_j = \pi_k$, which yields $w_j \mathbb{E}(P_k) - w_k \mathbb{E}(P_j) = 0$.
In other words, the order of the α-close jobs doesn't contribute to the cost-function at all. Plugging this into the result of Theorem 4.14, we obtain the desired result,

$$\mathrm{ROG}_I(\texttt{SWOPS}) \leq \frac{1}{\alpha + 1}.$$

□

Based on this previous example, we see that in some cases, the knowledge that only some of the jobs are α-far can still be leveraged to prove a good bound on the ROG of `SWOPS`. However, this doesn't work well in general. In the following corollary, we provide an extreme counter-example.

Corollary 4.16. *Consider a problem instance $I \in \mathcal{I}_e$. Suppose that we know that for each pair of jobs j, k except one, j, k are α-far for some $\alpha > 1$. Then the ROG-bound obtained in Theorem 4.14, and the ROG itself, can still be arbitrarily close to $\frac{1}{2}$.*

Proof. Our strategy will be to sketch a concrete problem instance that constitutes a counter-example. The extension to a more general counter-example is left up to the reader.

To start with, note that the magnitude of the weight and expected processing time of the jobs within a given pair influences the extent to which they contribute to the cost-function. If both jobs j, k have large weights and expected processing times, $w_j\mathbb{E}(P_k) - w_k\mathbb{E}(P_j)$ may be large, even if π_j and π_k are close together.
This motivates the idea of a "heavy pair" counter-example, where the single α-close pair of jobs has extremely large weights and expected processing times. Let $\alpha > 1$, e.g. $\alpha = 2$. Consider a problem instance with $N > 2$, where all pairs of jobs are α-far, except for the pair of jobs $1, 2$. Here we will consider $N = 100$, but other values work just as well. Consider the case where jobs $1, 2$ have extremely large values for the weight and expected processing time. Here we will take $w_1 = 0.99 \cdot 10^9, \lambda_1 = 0.99 \cdot 10^{-9}$ and $w_2 = 10^9, \lambda_2 = 10^{-9}$. Suppose furthermore that each job beyond 2 has weight and expected processing time of at most a single order of magnitude. For convenience, here we will use $w_j = 1$ and $\mathbb{E}(P_j)^{-1} = \lambda_j = 2^{j-1}$ for $2 < j \leq N$.
In this case, almost all of the contribution to the cost comes from the pair of jobs $1, 2$. Since the priorities of jobs $1, 2$ are close together, `SWOPS` will schedule this pair in the wrong order with probability almost $\frac{1}{2}$. Though it will schedule each other pair of jobs in the correct order with a relatively high probability of $\frac{1}{\alpha+1}$, this will have little impact on the overall cost. Therefore the overall $\mathrm{ROG}_I(\texttt{SWOPS})$ will be close to $\frac{1}{2}$.
This is also reflected in the bound derived in Theorem 4.14. For the specific values mentioned above, we can verify that the values of $D_{\alpha-close}$ and $D_{\alpha-far}$ are given by

$$\begin{aligned} D_{\alpha-close} &\approx 2.01 \cdot 10^{16} \\ D_{\alpha-far} &\approx 1.95 \cdot 10^{11}. \end{aligned}$$

Plugging these values into the formula from Theorem 4.14, we obtain the bound

$$\begin{aligned} \mathrm{ROG}_I(\texttt{SWOPS}) &\leq \frac{1}{\alpha+1}\frac{D_{\alpha-far}}{D_{\alpha-far}+D_{\alpha-close}} + \frac{1}{2}\frac{D_{\alpha-close}}{D_{\alpha-far}+D_{\alpha-close}} \\ &\approx 0.49999838318. \end{aligned}$$

As one can see, we need to a large number of digits to even see that the bound is not equal to $\frac{1}{2}$. The actual ROG for this scenario is given by

$$\mathrm{ROG}_I(\texttt{SWOPS}) \approx 0.49497025226.$$

The code used to calculate these values can be found in Appendix Section C. One should note that this example can be trivially modified to give an even worse ROG and ROG-bound, simply by increasing the weights and expected processing times of jobs $1, 2$ by an identical constant factor. □

One should be careful on how to interpret the above example. The lesson to be learned from it is that the information that almost all jobs are α-far is not sufficient to guarantee a good ROG-bound. This isn't necessarily caused by the formula of our bound not being tight enough; in the above example we see the actual ROG is also close to $\frac{1}{2}$. Rather, the actual value of the ROG can become arbitrarily close to $\frac{1}{2}$, even when there is only a single pair of jobs that is α-close.
In the last example, we saw that we could make this happen by introducing a large discrepancy in the order of magnitude of the parameters for different jobs. One could imagine that if we were to bound such discrepancies, perhaps we can recover a better bound on $\mathrm{ROG}_I(\texttt{SWOPS})$. An attempt at this is made in Appendix Section D.

We have considered the case where our assumptions are that jobs have exponentially distributed processing times, and almost all jobs are α-far. Overall, our conclusion must be that with only these assumptions, we cannot guarantee SWOPS performs significantly better than R.

5 Conclusions

In this section we will first summarize and discuss the main results from Sections 3 and 4. Afterwards, we will list some potential improvements, and ideas for topics to study in the future.

5.1 Discussion

First, we made the stochastic single-machine scheduling problem with sampled processing times more rigorous by defining a cost-function on algorithms,
$\mathrm{Cost}_I(A) := \sum_{s \in S_N} \Pr(A \text{ picks } s)\, \mathbb{E}_{p\sim P}\left(\sum_j w_j C_j(s)\right)$. We introduced the partial order $\leq_{\mathcal{I}}$, so we could make meaningful statements about whether an algorithm performs at least as well as another.
Then we simplified the problem by noting that the cost of an algorithm only depends on the pairwise ordering of each pair of jobs. This idea yielded Theorem 3.7, where we showed that under several simple assumptions, the question of whether an algorithm A outperforms another algorithm B can be answered by considering instances with only two jobs.

Another contribution is the introduction of the concept of the Relative Optimality Gap. We showed that bounding the ROG of an algorithm A generalizes the concept of comparing A to R. This also allowed us to compare algorithms in a more quantitative sense, by showing ROG bounds strictly lower than $\frac{1}{2}$. In particular, when A schedules in the pairwise correct order with probability at least κ, we showed that we can guarantee a ROG-bound of $1 - \kappa$.

With these tools, we managed to show for several cases that `SWOPS` performs at least as well as the algorithm R that schedules jobs uniformly at random. Specifically we considered the cases where the processing time distributions are either symmetric, or translated or scaled copies of the same underlying distribution. We managed to explain intuitively why `SWOPS` performs well in these cases.
Additionally, we considered the well-behaved case of exponential processing times. We showed that when all the jobs have priorities that are a large factor apart, the `SWOPS` algorithm performs considerably better than R. We also explored the possibility of proving a similar ROG-bound when only some subset of pairs of jobs have α-far priorities. However, we ran into the problem that the derived bounds depend on job parameters to such an extent that improved results require additional strong assumptions on the input data, which contradicts the overall scope of the research question.

An interesting perspective on these results is as an indication of the failure modes of `SWOPS`, which is arguably the only reasonable algorithm in the setting that we study. When jobs have very asymmetric processing time distributions, `SWOPS` can perform poorly, even compared to R. Similarly when the processing time distributions of different jobs have a very different shape, `SWOPS` might not perform well. These are also properties we saw in the counter-example outlined in Section 3.3. One could take this to mean that under such circumstances, the practical approach of pretending we can apply the law of large numbers to estimate the expected processing times might fail dramatically when the number of samples is insufficient.

Overall, we managed to explore a new problem, develop some novel methods, and obtain original results. We built a lot of intuition about what aspects of the problem are important. However, there is certainly still a lot to study. We touch on this in the next section.

5.2 Future research directions

There are several ideas for directions in which to take this research. Below we discuss a number of these, in no particular order.

We have shown in Section 4 that we can find restrictions on the space of job scheduling problems such that `SWOPS` performs at least as well as R. For example, when processing time distributions are symmetric or when they are scaled or translated versions of the same underlying distribution. Are these the only such restrictions, or do there exist more? Can we find some more general, easy, and intuitive rule to tell whether `SWOPS` performs at least as well as R given a problem space $\mathcal{I}$?
A logical next question to ask is whether we can still prove partial results when these

conditions only hold in approximation. That is to say, if all processing time distributions are nearly symmetric, can we still prove a performance bound on SWOPS? What about if they are nearly but not quite copies of the same underlying distribution, up to scaling or translation?

Another thing to note is that the proofs from Section 4 make use of Lemma 4.2 to write down the probability of the pairwise order of jobs in a closed form. Technically, this lemma only works for distributions that have a well-defined probability density function. When working with finitely discrete distributions and delta-peaks, the proofs would have to be adapted to sums instead of integrals. It is conceivable that the proofs would look approximately the same.

In Section 4.4 we show that SWOPS has a ROG-bound strictly less than $\frac{1}{2}$ on the space of problem instances with exponentially distributed processing times and α-far jobs. Are there more choices of problem spaces with such a result? Perhaps one with more natural assumptions?
In Section 4.5 we attempt to prove a similar ROG-bound in the case where not all jobs are α-far. Eventually, we run into the problem that the bound we derive depends on parameters that are in principle unknown to us. Is there some alternative approach here that does yield useful results? What additional, minimal assumptions could we make about the problem space in order to still obtain a usable bound?

Lastly, so far we have restricted ourselves to considering the setting where the processing time distributions are only sampled once. If we reach a good understanding of this limited scenario, it would be compelling to also consider the more general case, where multiple samples are allowed. The natural extension would be to first consider the case with exactly two samples per distribution.

A Interpreting the cost-function as expected regret

In Section 3.2, we define the cost-function as

$$\text{Cost}(A) := \sum_{s \in S_N} \Pr\left(A \text{ picks } s\right) \mathbb{E}\left(\sum_j w_j C_j(s)\right).$$

One way of interpreting the setting where we minimize this cost-function is as being equivalent to minimizing the "expected regret". Here the regret is the difference in cost compared to the offline optimal solution that one could obtain if one had known in advance the realized processing times. In this appendix we will first explain what we mean with this statement. Then we will briefly discuss why this perspective is useful, and what insights it can give.

Given a problem instance $I \in \mathcal{I}$, a schedule s, and realizations of the processing times p_j, we define the regret as

$$\text{Regret}_I(s, p = (p_1, \ldots, p_N)) = \left(\sum_j w_j C_j(s, p)\right) - \min_{s^* \in S_N} \left(\sum_j w_j C_j(s^*, p)\right).$$

In other words, the regret is the difference between the obtained Smith [2] cost, minus the cost we could have obtained optimally if we knew all the parameter realizations beforehand. Note that $C_j(s, p)$ simply denotes the completion time of job j under schedule s and realizations of the processing times $p = (p_1, \ldots, p_N)$.
If we apply the expectation operator with regards to the processing time distributions P_j to this expression, the right-hand term becomes a constant with respect to the choice of s.
When trying to find the argument that minimizes a function, we can always drop any constant terms. From this, we see that

$$\begin{aligned}\underset{s \in S_N}{\text{argmin}} \left(\mathop{\mathbb{E}}_{p \sim P} \left(\text{Regret}_I(s, \{p_j\})\right)\right) &= \underset{s \in S_N}{\text{argmin}} \left(\mathop{\mathbb{E}}_{p \sim P} \left(\sum_j w_j C_j(s, \{p_j\})\right)\right) \\ &= \underset{s \in S_N}{\text{argmin}} \left(\sum_j w_j \mathop{\mathbb{E}}_{p \sim P} (C_j(s, \{p_j\}))\right).\end{aligned}$$

Or in words, the schedule that minimizes the expected regret is the same as the schedule that minimizes the expected cost as defined in Rothkopf [3].
Additionally taking the expected value with regards to the schedule picked by an algorithm A, we can argue that

$$\begin{aligned}\operatorname*{argmin}_{s\in S_N}\left(\mathrm{Cost}_I(A)\right) &= \operatorname*{argmin}_{A} \mathop{\mathbb{E}}_{s\sim A}\left(\sum_j w_j \mathop{\mathbb{E}}_{p\sim P}(C_j(s,\{p_j\}))\right)\\ &= \operatorname*{argmin}_{A} \mathop{\mathbb{E}}_{s\sim A}\left(\mathop{\mathbb{E}}_{p\sim P}\left(\mathrm{Regret}_I(s,\{p_j\})\right)\right).\end{aligned}$$

That is to say, minimizing the cost of an algorithm A is the same as minimizing the expected regret incurred by that algorithm.

This perspective can help us understand what is happening when an algorithm has a high cost. Consider the example outlined in Section 3.3, where $w_1 = w_2 = 1$ and the processing time distributions are given as

$$P_1 = \begin{cases} 0 & \text{with probability } 1 - \frac{1}{M} \\ M^2 & \text{with probability } \frac{1}{M} \end{cases}$$
$$P_2 = \epsilon \text{ with probability 1.}$$

In Section 3.3, we showed that for this problem instance, `SWOPS` performs very poorly. We can now explain why this happens, using the concept of regret.
With overwhelmingly high probability, `SWOPS` will schedule job 1 before job 2. In most cases, namely with probability $(1 - \frac{1}{M})$, the processing times will be realized as $p_1 = 0$ and $p_2 = \epsilon$, meaning the regret will be 0. However, with probability $\frac{1}{M}$, the processing time of job 1 will be realized as $p_1 = M^2$. The regret in this case will be $w_2 p_1 - w_1 p_2 = M^2 - \epsilon \approx M^2 \gg 1$. This regret is extremely high. Even though it only occurs with low probability, it still makes the expected regret when scheduling job 1 before job 2 significantly higher than when scheduling in the reverse order.

Another useful insight from looking at the cost-function in this way is that it gives us an intuition for some other choices for a cost-function. The cost-function is arguably a design-choice, and is dependent on what our goal is. Instead of minimizing expected regret, we could for example also try to minimize the probability of having non-zero regret. A fun exercise is to convince oneself that under such a cost-function, the `SWOPS` algorithm actually performs extremely well on the problem instance described above.

B A formal proof of SWOPS $\leq_{\mathcal{I}_{symmetric}} R$

This proof continues from Equations 4.1 and 4.2 in Theorem 4.4. These two equations are copied below. Here we will do a formal proof, rather than a visual one.

$$\Pr\left(\texttt{SWOPS}_{1\to 2}(I_2)\right) = \frac{1}{w_1 w_2}\int_{-w_1E_2}^{w_1E_2} dy\ f_2(E_2+\frac{y}{w_1})\int_{-w_2E_1}^{y+w_1E_2-w_2E_1} dx\ f_1(E_1+\frac{x}{w_2}). \quad \text{(B.1)}$$

Analogously, for $\Pr\left(\texttt{SWOPS}_{2\to 1}(I_2)\right)$, we have the expression

$$\Pr\left(\texttt{SWOPS}_{2\to 1}(I_2)\right) = \frac{1}{w_1 w_2}\int_{-w_2E_1}^{w_2E_1} dx\ f_1(E_1+\frac{x}{w_2})\int_{-w_1E_2}^{x+w_2E_1-w_1E_2} dy\ f_2(E_2+\frac{y}{w_1}). \quad \text{(B.2)}$$

Since the integrand in this last expression is a product of probability density functions, it is certainly absolutely integrable. This means we can apply Fubini's theorem to exchange the order of integration, and express $\Pr\left(\texttt{SWOPS}_{2\to 1}(I_2)\right)$ as

$$\Pr\left(\texttt{SWOPS}_{2\to 1}(I_2)\right) = \frac{1}{w_1 w_2}\int_{-w_1E_2}^{-w_1E_2+2w_2E_1} dy\ f_2(E_2+\frac{y}{w_1})\int_{y+w_1E_2-w_2E_1}^{w_2E_1} dx\ f_1(E_1+\frac{x}{w_2}).$$

Next, we make the substitution $x' = -x$. By using the fact that f_1 is symmetric, i.e. $f_1(E_1 - \frac{x'}{w_2}) = f_1(E_1 + \frac{x'}{w_2})$, we get the expression

$$\Pr\left(\texttt{SWOPS}_{2\to 1}(I_2)\right) = \frac{1}{w_1 w_2}\int_{-w_1E_2}^{-w_1E_2+2w_2E_1} dy\ f_2(E_2+\frac{y}{w_1})\int_{-w_2E_1}^{-y-w_1E_2+w_2E_1} dx'\ f_1(E_1+\frac{x'}{w_2}).$$

Here we got rid of a minus sign by swapping the lower and upper bound of the right-hand integral. Similarly substituting $y' = -y$ gives us that

$$\Pr\left(\texttt{SWOPS}_{2\to 1}(I_2)\right) = \frac{1}{w_1 w_2}\int_{w_1E_2-2w_2E_1}^{w_1E_2} dy'\ f_2(E_2+\frac{y'}{w_1})\int_{-w_2E_1}^{y'-w_1E_2+w_2E_1} dx'\ f_1(E_1+\frac{x'}{w_2}).$$

We now compare this to the expression for $\Pr\left(\texttt{SWOPS}_{1\to 2}(I_2)\right)$ in 4.1. We see that the two integral expressions differ only in their integration boundaries. Next, we try to match up the integration boundaries in the expression for $\Pr\left(\texttt{SWOPS}_{2\to 1}(I_2)\right)$ with those for $\Pr\left(\texttt{SWOPS}_{1\to 2}(I_2)\right)$. Starting with the boundaries on the right-hand integral, we have that

$$\begin{aligned}
\Pr\left(\texttt{SWOPS}_{2\to1}(I_2)\right) &= \frac{1}{w_1 w_2}\int_{w_1E_2-2w_2E_1}^{w_1E_2} dy'\ f_2(E_2+\frac{y'}{w_1}) \\
&\quad \left[\int_{-w_2E_1}^{y'+w_1E_2-w_2E_1} dx'\ f_1(E_1+\frac{x'}{w_2}) - \int_{y'-w_1E_2+w_2E_1}^{y'+w_1E_2-w_2E_1} dx'\ f_1(E_1+\frac{x'}{w_2})\right] \\
&= -\text{int1} + \left[\frac{1}{w_1 w_2}\int_{w_1E_2-2w_2E_1}^{w_1E_2} dy'\ f_2(E_2+\frac{y'}{w_1}) \int_{-w_2E_1}^{y'+w_1E_2-w_2E_1} dx'\ f_1(E_1+\frac{x'}{w_2})\right], \\
\text{where int1} &:= \frac{1}{w_1 w_2}\int_{w_1E_2-2w_2E_1}^{w_1E_2} dy'\ f_2(E_2+\frac{y'}{w_1}) \int_{y'-w_1E_2+w_2E_1}^{y'+w_1E_2-w_2E_1} dx'\ f_1(E_1+\frac{x'}{w_2}).
\end{aligned}$$

Analogously, splitting the left-hand integral, we have that

$$\begin{aligned}
\Pr\left(\texttt{SWOPS}_{2\to1}(I_2)\right) &= -\text{int1} + \frac{1}{w_1 w_2}\left[\int_{w_1E_2-2w_2E_1}^{w_1E_2} dy'\ f_2(E_2+\frac{y'}{w_1}) \int_{-w_2E_1}^{y'+w_1E_2-w_2E_1} dx'\ f_1(E_1+\frac{x'}{w_2})\right] \\
&= -\text{int1} + \frac{1}{w_1 w_2}\left[\int_{-w_1E_2}^{w_1E_2} dy'\ f_2(E_2+\frac{y'}{w_1}) - \int_{-w_1E_2}^{w_1E_2-2w_2E_1} dy'\ f_2(E_2+\frac{y'}{w_1})\right] \\
&\quad \int_{-w_2E_1}^{y'+w_1E_2-w_2E_1} dx'\ f_1(E_1+\frac{x'}{w_2}) \\
&= -\text{int1} - \text{int2} + \left[\frac{1}{w_1 w_2}\int_{-w_1E_2}^{w_1E_2} dy'\ f_2(E_2+\frac{y'}{w_1}) \int_{-w_2E_1}^{y'+w_1E_2-w_2E_1} dx'\ f_1(E_1+\frac{x'}{w_2})\right], \\
\text{where int2} &:= \int_{-w_1E_2}^{w_1E_2-2w_2E_1} dy'\ f_2(E_2+\frac{y'}{w_1}) \int_{-w_2E_1}^{y'+w_1E_2-w_2E_1} dx'\ f_1(E_1+\frac{x'}{w_2}).
\end{aligned}$$

The right-hand integral in the above expression is identical to the formula for $\Pr\left(\texttt{SWOPS}_{1\to2}(I_2)\right)$ in Equation 4.1. Subtracting $\Pr\left(\texttt{SWOPS}_{2\to1}(I_2)\right)$ from $\Pr\left(\texttt{SWOPS}_{1\to2}(I_2)\right)$, we obtain that

$$\begin{aligned}
\Pr\left(\texttt{SWOPS}_{1\to2}(I_2)\right) - \Pr\left(\texttt{SWOPS}_{2\to1}(I_2)\right) &= \text{int1} + \text{int2} \\
&= \frac{1}{w_1 w_2}\int_{w_1E_2-2w_2E_1}^{w_1E_2} dy'\ f_2(E_2+\frac{y'}{w_1}) \int_{y'-w_1E_2+w_2E_1}^{y'+w_1E_2-w_2E_1} dx'\ f_1(E_1+\frac{x'}{w_2}) \\
&\quad + \int_{-w_1E_2}^{w_1E_2-2w_2E_1} dy'\ f_2(E_2+\frac{y'}{w_1}) \int_{-w_2E_1}^{y'+w_1E_2-w_2E_1} dx'\ f_1(E_1+\frac{x'}{w_2}) \\
&\geq 0,
\end{aligned}$$

where the last inequality follows from applying the assumption $w_1E_2 \geq w_2E_1$ to the integral boundaries, and also using that the integrands are strictly non-negative everywhere. We conclude that indeed $\Pr\left(\texttt{SWOPS}_{1\to2}(I_2)\right) \geq \Pr\left(\texttt{SWOPS}_{2\to1}(I_2)\right)$, which finishes the proof.

C Code for calculating ROG$_I$(SWOPS) for exponential processing times

```
import numpy as np

N = 100
alpha = 2
# weights and lambdas for one heavy pair

wl = np.zeros((N,2))
#w0 is
wl[0,0] = 0.99e9
#lambda0 is
wl[0,1] = 0.99e-9
#w1 is
wl[1,0] = 1e9
#Lambda1 is
wl[1,1] = 1e-9
#rest is
idx = 2
while idx < N:
        wl[idx,0] = 1
        wl[idx,1] = 2*wl[idx-1,1]
        if idx==2:
                wl[idx,1] = 2
        idx += 1


# Calculate D_close, D_far
D_close = 0
D_far = 0
ROG = 0
H_minus_L = 0
for j in range(0,N-1):
        for k in range(j,N):
                # wj lj / wk lk
                priority_factor = (wl[j,0]*wl[j,1]) / (wl[k,0]*wl[k,1])
                # if jobs j,k alpha-far
                if priority_factor >= alpha or priority_factor <= alpha**-1:
                        # add contribution |(wj / lk) - (wk / lj)| to D_far
                        D_far += abs(wl[j,0]/wl[k,1] - wl[k,0]/wl[j,1])
                        # if jobs j,k alpha-close
                else:
                        # add contribution |(wj / lk) - (wk / lj)| to D_close
                        D_close += abs(wl[j, 0] / wl[k, 1] - wl[k, 0] / wl[j, 1])
```

```
            # add contribution |(wj / lk) - (wk / lj)| to D=H-L
            H_minus_L += abs(wl[j, 0] / wl[k, 1] - wl[k, 0] / wl[j, 1])

            # calculation real ROG
            pi_j = wl[j,0]*wl[j,1]
            pi_k = wl[k,0]*wl[k,1]
            # add contribution |(wj / lk) - (wk / lj)|
            # times probability of scheduling in incorrect order
            # pi_j/(p_j + pi_k)
            ROG += (pi_j/(pi_j + pi_k)) *\
                    abs(wl[j, 0]/wl[k, 1] - wl[k, 0]/wl[j, 1])

# Normalize ROG, so it lies in [0,1]
ROG = ROG/H_minus_L

# Calculate value of ROG-bound
ROG_bound = (1/(alpha+1))*(D_far/(D_close+D_far)) + (1/2)*(D_close/(D_close+D_far))

print("D_close is :", '{:.10E}'.format(D_close))
print("D_far is :", '{:.10E}'.format(D_far))
print("ROG_bound is :", '{:.10E}'.format(ROG_bound))
print("ROG is :", '{:.10E}'.format(ROG))
```

D Exponential processing times with any priorities, and bounded weights and rates

Theorem D.1. *Consider the setting of Theorem 4.14. Additionally assume that we know a lower and upper bound on the weights, $w_{min} \leq w_j \leq w_{max} \forall j \in I$. Similarly, assume that we know a lower and upper bound on the expected processing times, $P_{min} \leq \mathbb{E}(P_j) \leq P_{max} \forall j \in I$. Then it follows that*

$$ROG_I(\texttt{SWOPS}) < \frac{1}{\alpha+1}\frac{\#far}{\#far + \#close\left(\frac{w_{max}}{w_{min}}\frac{P_{max}}{P_{min}}\right)} + \frac{1}{2}\frac{\#close}{\#far\left(\frac{w_{min}}{w_{max}}\frac{P_{min}}{P_{max}}\right) + \#close}.$$

Proof. For α-close pairs of jobs j, k, we have that

$$\begin{aligned} \frac{w_j}{\mathbb{E}(P_j)} &< \alpha\frac{w_k}{\mathbb{E}(P_k)} \\ \iff w_j\mathbb{E}(P_k) &< \alpha w_k\mathbb{E}(P_j) \\ \iff w_j\mathbb{E}(P_k) - w_k\mathbb{E}(P_j) &< (\alpha-1)w_k\mathbb{E}(P_j) \leq (\alpha-1)w_{max}P_{max}. \end{aligned}$$

Analogously, we can show that for α-far pairs of jobs, $w_j\mathbb{E}(P_k) - w_k\mathbb{E}(P_j) \geq (\alpha - 1)w_{min}P_{min}$.
Applying this inequality to each of the terms in the double sum expression for $D_{\alpha-far}$ and $D_{\alpha-close}$, we obtain that

$$\begin{aligned} D_{\alpha-close} &< \#close(\alpha-1)w_{max}P_{max} \\ D_{\alpha-far} &\geq \#far(\alpha-1)w_{min}P_{min}, \end{aligned}$$

where $\#close$ and $\#far$ stand for the number of α-close and the number of α-far pairs of jobs respectively. In the result of Theorem 4.14 we had a term $\frac{D_{\alpha-close}}{D_{\alpha-far}+D_{\alpha-close}}$. Assume for a moment that $D_{\alpha-close} \neq 0$. Then applying the above inequalities to this term yields the expression

$$\begin{aligned} \frac{D_{\alpha-close}}{D_{\alpha-far}+D_{\alpha-close}} &= \frac{1}{\frac{D_{\alpha-far}}{D_{\alpha-close}}+1} \\ &< \frac{1}{\frac{\#far(\alpha-1)w_{min}P_{min}}{\#close(\alpha-1)w_{max}P_{max}}+1} \\ &= \frac{\#close}{\#far\left(\frac{w_{min}}{w_{max}}\frac{P_{min}}{P_{max}}\right)+\#close}. \end{aligned}$$

If we *did* have $D_{\alpha-close} = 0$, this inequality would still hold trivially. Thus, it holds in general. Consider the result of Theorem 4.14

$$\mathrm{ROG}_I(\texttt{SWOPS}) \leq \frac{1}{\alpha+1}\frac{D_{\alpha-far}}{D_{\alpha-far}+D_{\alpha-close}} + \frac{1}{2}\frac{D_{\alpha-close}}{D_{\alpha-far}+D_{\alpha-close}}$$

Since $\frac{D_{\alpha-far}}{D_{\alpha-far}+D_{\alpha-close}} + \frac{D_{\alpha-close}}{D_{\alpha-far}+D_{\alpha-close}} = 1$, and $\frac{1}{\alpha+1} < \frac{1}{2}$, we see that for fixed α the upper bound on $\mathrm{ROG}_I(\texttt{SWOPS})$ is maximal when $\frac{D_{\alpha-close}}{D_{\alpha-far}+D_{\alpha-close}}$ is maximal. Applying the upper bound we derived on $\frac{D_{\alpha-close}}{D_{\alpha-far}+D_{\alpha-close}}$, we obtain that

$$\mathrm{ROG}_I(\texttt{SWOPS}) < \frac{1}{\alpha+1}\frac{\#far}{\#far+\#close\left(\frac{w_{max}}{w_{min}}\frac{P_{max}}{P_{min}}\right)} + \frac{1}{2}\frac{\#close}{\#far\left(\frac{w_{min}}{w_{max}}\frac{P_{min}}{P_{max}}\right)+\#close}.$$

□

Suppose we know that a number of pairs of jobs $\#far$ is α-far for some $\alpha > 1$, and we somehow have access to a bound $\frac{P_{max}}{P_{min}} = \frac{\max\limits_j \lambda_j}{\min\limits_k \lambda_k} \leq C$ for some constant C. Then Theorem

D.1 gives us a ROG-bound on SWOPS strictly better than $\frac{1}{2}$. However, this estimated bound is very rough, and not useful in practice.
For example, consider the case where at least half the jobs are 2-far, and we know that $\frac{w_{max}}{w_{min}} = \frac{P_{max}}{P_{min}} = 10$. These are already strong assumptions. Normally there is no reason we would have any knowledge about $\frac{P_{max}}{P_{min}}$. If we knew the expected processing times, we would use the Rothkopf [3] algorithm after all, not the SWOPS algorithm. Moreover, all expected processing times lying within a single order of magnitude of one another is a bold assumption to make; in practice this might not hold.
Even under these strong assumptions, the bound derived above evaluates to

$$\begin{aligned}\mathrm{ROG}_I(\texttt{SWOPS}) &< \frac{1}{3}\frac{N/2}{N/2 + N/2\,(10 * 10)} + \frac{1}{2}\frac{N/2}{N/2\,(1/10 * 1/10) + N/2} \\ &= \frac{1}{3}\frac{1}{101} + \frac{1}{2}\frac{100}{101} = \frac{151}{303} \approx 0.4983,\end{aligned}$$

which is barely better than the bound of $\frac{1}{2}$ we already had.

References

[1] Hegen Xiong et al. “A survey of job shop scheduling problem: The types and models”. In: *Computers & Operations Research* (2022), p. 105731.

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

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

[4] *Algorithms with Predictions*. `https://algorithms-with-predictions.github.io/`. Accessed: 2023-05-17.

[5] Michael L Pinedo. *Scheduling*. Vol. 29. Springer, 2012.

[6] Wouter Fokkema. Personal Communication. 2022.