Scheduling with explorable uncertainty for minimising the total weighted completion time

Xiao-Ou Zhang
Utrecht University, Department of Information and Computing Sciences
xzhang1404@gmail.com

Abstract

This thesis addresses the problem of scheduling with explorable uncertainty on a single machine. We consider the scenario where n jobs have a uniform upper limit on the processing time $\bar{u}$, a uniform testing time of 1 and a true processing time p_j ($0 \leq p_j \leq \bar{u}$), which is only revealed after testing. Moreover, each job j has a weight $w_j \in \mathbb{N}$. The machine can either test a job or execute a job. The duration of executing any job j is either $\bar{u}$ or p_j, depending on whether the job was tested or not. Since decisions, regarding testing and execution, are irrevocable and the true processing time is initially hidden, this problem can be viewed as an online problem. The objective is to schedule all jobs to minimise the total weighted completion time. We present two deterministic algorithms, Delay-All (DA) and L-Delay-All (L-DA), and analyse their competitive ratios. For the special case with two weights (1 and α), DA achieves a competitive ratio of $3 + \frac{1}{2} \times \alpha$, while L-DA achieves a ratio of $3 + \frac{1}{2} \times \bar{u}$. For the more general case with multiple weights, DA and L-DA achieve a competitive ratios of $1 + 2 \times w_{max}$ and $3 + \frac{5}{3} \times \bar{u}$, respectively. We also introduce an algorithm called Unified-Delay-All (U-DA) which combines DA and L-DA. For the case with two weights U-DA is 3-competitive when $\alpha \geq \bar{u}$. Moreover, we present a modified version of L-DA, called Postpone-L-Delay-All (PL-DA), which achieves a 3-competitive ratio for the case with two weights when the number of α-weighted jobs is at least $\frac{n}{\bar{u}}$. Furthermore, we show that for DA and L-DA there exists an instance where the competitive ratio is at least 2.4-competitive and 3.4-competitive, respectively.

Keywords and phrases Scheduling with testing, Online, Explorable uncertainty, Minimising the total weighted completion time

Contents

1 Introduction

Real-world problems often involve incomplete information, such as uncertain job durations or unknown costs. For instance, a programmer tasked with testing different functions of a computer program receives every day a batch of code files where each code file represents a function. The testing duration of each file depends on its size, but the programmer can potentially reduce the file size by optimising the code using ChatGPT. However, ChatGPT cannot guarantee a reduction in file size, as the code may already be fully optimized. The challenge is to efficiently test all functions while balancing the cost and benefit of optimising each file. The strategy may depend on previous actions and findings. In other words, the programmer must decide whether to invest time in optimising a file and what the order of operations should be.

Whenever there is the possibility of acquiring more explicit information by performing some preliminary action, we can model this problem as scheduling with explorable uncertainty. Mathematically, we consider m machines, where n jobs have to be scheduled on a machine. For each job j the true processing time p_j $(0 \leq p_j)$ is initially hidden. However, the upper limit on the processing time u_j $(p_j \leq u_j)$ is already revealed to the decision-maker, as well as the testing time $t_j \geq 0$. Executing any job j untested takes a total time of u_j, while testing and executing any job j takes a total time of $p_j + t_j$. The decision-maker can decide whether a job is executed in the tested or untested condition. Any decision, whether testing or executing, is irrevocable. Therefore, in combination with the fact that the true processing time of a job is initially hidden, we can consider this as an online problem. The goal is to schedule all jobs such that we minimise or maximise the objective function.

Dürr et al. [8] have studied this problem setting for cases where all jobs have a testing time of 1 and have proven upper and lower bounds for the objective function of minimising the total completion time and makespan on a single machine. We review these findings in the literature review. In this thesis, we extend the problem by introducing weights. In the weighted problem variant, each job j has a weight w_j and we consider the objective function of minimising the total weighted completion time, which presents a greater challenge compared to the unweighted case. The non-trivial decisions, in the unweighted case, are whether we should test a job and the order of operations (test and execute). In the weighted case, we need to determine the order of operations more carefully since any delay in the completion time of a job will be multiplied by the weight.

The purpose of the research is to analyse and develop an algorithm, such that the schedule is within a provably good factor of the optimal schedule. By introducing weights, we prove that the approach described in paper[8] cannot be generalised. Therefore, we propose a novel approach that addresses the weighted problem. We focus on a variant of the problem where all jobs have a testing time of one and a uniform upper limit, which makes them initially indistinguishable to the algorithm. We present two main algorithms, Delay-All and L-Delay-All, and prove the upper bound on their competitive ratio, considering the special case where we have only 2 weights and a more general case with multiple weights. We show and prove the competitive ratio of a greedy algorithm to compare the performance of our algorithms in both cases. We also rigorously analyse the case with two weights and present two more algorithms. The first one, Unified-Delay-All, is a combination of the two main algorithms, which, under certain circumstances, is constant-competitive. The second one, Postpone-L-Delay-All, is a variant of an L-Delay-All with an adjusted execution strategy and is constant-competitive for a specific type of instances. See Table 1 for the upper bound results. We also consider the lower bound on the two main algorithms. Dürr et al. [8] provided a lower bound of 1.8546 for the unweighted case. We refer to Table 2 for the lower bound results.

Upper bound		
Algorithm	Two weight(1 and $\alpha \in \mathbb{N}$)	Multiple weight
Greedy	$1 + \bar{u}$	$1 + \bar{u}$
Delay-All	$3 + \frac{1}{2} \times \alpha$	$1 + 2 \times w_{max}$
L-Delay-All	$3 + \frac{1}{2} \times \bar{u}$	$3 + \frac{5}{3} \times \bar{u}$
Unified-Delay-All	3 when $\alpha \geq \bar{u}$, otherwise $3 + \frac{1}{2} \times \alpha$	-
Postpone-L-Delay-All	3 when number of α-weighted jobs $\geq \frac{n}{\bar{u}}$	-

Table 1 Upper bound results for the case with two weights and the more general case where we have multiple weights. w_{max} is the maximum weight in a given instance

1.1 Thesis structure

In Section 2, we provide real-world examples that motivate our research, along with a review of the existing literature on scheduling and explorable uncertainty. In Section 3, we provide a formal definition of the scheduling problem with explorable uncertainty on a single machine and introduce the notation and properties that we use throughout the thesis. Section 4 is a

Lower bound	
Problem	1.8546 [8]
Delay-All	2.4
L-Delay-all	3.4

Table 2 Lower bound results on the algorithm and the problem.

pivotal part of our thesis, where we present our analysis approach and provide a lower bound of the optimal solution and insights into the optimal solution. In Section 5, we present a greedy execution strategy that serves as a benchmark for the more sophisticated algorithms presented later. In Section 6, we introduce our first sophisticated algorithm, Delay-All (DA), and prove its competitive ratio for the special case with two weights and the more general case with multiple weights. In Section 7, we propose another algorithm, L-Delay-All (L-DA), and prove its competitive ratio for the same cases described in Section 6. This chapter also includes an analysis of instance parameters, showing that an algorithm Unified-Delay-All (U-DA) is, under certain conditions, constant-competitive. In Section 8, we show that by adjusting the execution strategy of L-DA, we can achieve a constant-competitive ratio of 3 for specific instances with two weights. In Section 9, we observe the competitive ratio for different types of instances. Finally, we conclude with a summary of the main contributions and findings of our research, along with recommendations for future research directions.

2 Literature Review

2.1 Motivation

Scheduling with testing has many applications in real-world areas, such as data and resource management, manufacturing, construction and maintenance work and medical diagnosis. We will highlight several areas and give some explicit examples.

For the first example, consider a server which is responsible for uploading videos. The server can upload any video, regardless of its size; however, the upload time depends on the video size. The server has a function to potentially reduce upload time by compressing the video size. However, it may be the case that even with compression, the upload time cannot be reduced. Furthermore, the server has only enough computing power to either upload or compress a video. Videos that are not on the server are missing opportunities to gain views and some videos will attract more views, such as cat-related videos. In this example, there are three important decisions: Whether we should compress a video or not, which video we should upload first and what is the order of actions (compress and upload)

Another example is car maintenance. Let's assume that the engine of a car does not start anymore as a consequence of a defective battery. The simplest solution is to bring the car to the repair shop and swap the battery with a new one. The cost would be the new battery and the service cost of the repair shop. However, we can ask the repair shop to run a diagnosis instead of swapping the battery. If it turns out the battery has only a small malfunction, which is repairable, then we can simply repair the battery instead of swapping it with a new one. Naturally, repairing the battery is much cheaper than swapping it. The worst-case scenario is that the battery cannot be fixed. In that case, we paid some additional costs for running the diagnosis. This situation illustrates a basic instance of our problem with one job, where we have a simple trade-off for a single task that might become cheaper by running a diagnosis.

Medical diagnoses provide another example of the trade-off between investing time to reduce processing time and executing a task with the given processing time. Consider a doctor who routinely receives a general diagnosis report from patients who visit their clinic. While the doctor has their own diagnostic method that yields results as accurate as the general report, it requires additional working time. In order to begin treatment, the doctor needs to identify the specific disease, which requires analysing the diagnosis report. Using the general report would entail a longer analysis time, while the doctor's method may not always produce a better report. Moreover, while conducting their own diagnosis, the doctor cannot attend to other patients, and delaying treatment may have serious consequences for the patient.

2.2 Scheduling problems: state-of-the-arts

In 1910, Gantt published [13], [7]. the first edition of his book, in which he introduced a graphical representation of a project schedule known as the *Gantt* chart or diagram. Arguably, the area of scheduling traces back to the problems that Gantt encountered and was trying to solve. Scheduling theory is part of operation research that has been studied extensively since its inception around the 1950s [26]. Some of the earliest work on scheduling was done by Johnson [27]. and Smith [27]. Johnson considered a production model, which we call now the *flow-shop*, with two machines where each job must be processed by the two machines separately. He showed an algorithm, called *Johnson's* rule, which is optimal for the objective of minimising the makespan (the time when all jobs are finished). Smith addresses a simpler problem where we only have a single machine and showed that for the objective of minimising the total completion time (the time that a job is finished), an optimal algorithm which always executes the job with the shortest processing time. The algorithm of Smith is also called the *SPT rule*.

In the last few decades, scheduling has been studied thoroughly in the literature [21] and [25]. Graham et al. [16] introduced a three-field scheme notation, which has proven useful in indicating all possible machine scheduling problems. The notation takes the form $(\alpha|\beta|\gamma)$, where α denotes the machine environment, such as the number of machines that may be identical (each with the same speed) or different (with varying speeds). The β field indicates constraints, which might include specific processing times or release times for each job. The γ field represents the objective function.

J.K Lenstra et al. [22] used the terms "easy" and "hard" in their research to describe the complexity of scheduling problems. For "easy" problems, there is a polynomial-bounded algorithm available, while "hard" problems are NP-complete. They presented an overview of scheduling problems, identifying which ones are "hard" or "easy," and provided NP-completeness proofs for some problems. With these proofs, they offered insight into the boundary between "easy" and "hard" scheduling problems. They also listed some open problems that have yet to be proven NP-complete, such as the scheduling problem $1||\sum T_i$ for n jobs. Despite extensive investigations, there is currently no polynomial-bounded algorithm or reduction proving its NP-completeness.

Scheduling problems, such as minimising the total completion time and the weighted variant ($1||\sum_j c_j$ and $1||\sum_j w_j \times c_j$), are considered to be "easy". Both problems can be solved in polynomial time by applying the SPT rule and the Weighted Shortest Processing Time (WSPT) rule, respectively. However, by adding only the constraint of release dates r_j, both problems become "hard". Even for the unweighted case, the problem is strongly NP-hard [22]. For the unweighted variant ($1|r_j|\sum_j c_j$), Phillips, Stein, and Wein [24] presented the first constant-factor approximation algorithm with an approximation ratio of 2, which also

works for the online case (job j revealed at time r_j). Hoogeveen and Vestjens [18] presented a deterministic 2-approximation algorithm for the same scheduling problem described in the paper by Phillips et al. Furthermore, they showed that there is no online algorithm with a performance guarantee better than 2-competitive. Goemans et al. [14] provided a comprehensive review of online algorithms for the weighted case, including the cases of preemption (where the machine can cancel the current job without losing progress and continue with another job) and non-preemption settings.

Another classical scheduling problem is the objective of minimising the makespan. Graham [15] considered the case where we have multiple machines and proved that an algorithm, also called the *List Scheduling*, is at most $2 - \frac{1}{m}$ times the optimal makespan. The algorithm follows a greedy approach where we assign each job to the smallest loaded machine, breaking ties arbitrarily.

2.3 Explorable uncertainty

Explorable uncertainty, also known as query-able uncertainty, is a specific model for handling uncertainty. Kahan introduced this model in 1991 in his paper [19], which focused on selection problems, such as finding the minimum value given a set of intervals where the values within the intervals are only revealed after a query. The objective is to minimise the number of queries required to find the optimal solution. Since then, several other problems have been studied with this uncertainty model, including computing the average and the *k-th* smallest value in a set of uncertainty intervals, as done by Kahanna and Tan in [20]. Another example of explicit exploration cost in this model is the Pandora's Box Problem [29], where a set of random variables is explored to maximise the highest revealed value while minimising the costs of revealing a value.

In 2008, Hoffmann et al. considered the Minimum Spanning Tree (*MST*) problem under the explorable uncertainty model [17], referred to as *MST-EDGE-UNCERTAINTY*. They defined the problem as follows: Find an MST for a connected, undirected, weighted graph $G = (V, E)$, where the edge weights ω_e are uncertain and represented as an interval A_e of possible values. The objective is to find the MST of G with the least number of updates, where an update reveals the weight of an edge e. They presented a 2-competitive algorithm when all intervals A_e are either open or trivial. In 2015, Erlebach and Hoffmann published a survey [12] on query-competitive algorithms in the computing with uncertainty model, summarising known results and techniques for designing these algorithms and suggesting future directions.

In 2021, Albers et al. studied the online makespan minimisation problem with uncertain job processing times [5]. They presented a $(3\text{-}\frac{2}{m})$-competitive algorithm for the *Graham's Greedy Strategy* and a deterministic algorithm with a competitive ratio of 2.9052. The problem involves assigning jobs with regular processing times and additional processing times to m identical machines. Each machine can accommodate up to ρ job failures, where a failure requires additional processing time to complete the specific job.

Recently, in 2022, Erlebach et al. [11] proposed an interesting approach to the explorable uncertainty model. They considered the MST problem, where the algorithm can query the weight of an edge, and introduced *untrusted* predictions for the weights of each edge. They developed a set of algorithms parameterized by a confidence parameter γ that reflects the user's confidence in the accuracy of the predictor. For any integer $\gamma \geq 2$, they presented a $(1+\frac{1}{\gamma})$-consistent and γ-robust algorithm, where an algorithm is α-consistent if it is α-competitive when the predictions are correct, and it is β-robust if it is β-competitive no matter how wrong the predictions are. They showed that their algorithm achieved the best

possible trade-off between consistency and robustness.

2.4 Scheduling with testing

In 2018, Dürr et al. [8] proposed a novel model for single-machine scheduling problems with explorable uncertainty, where some preliminary tests can be conducted on jobs before execution, potentially reducing the processing time. The authors considered the following scenario; Given n number of jobs where each job j has a processing time u_j and a true processing time p_j, which is revealed after testing. The algorithm can decide, at any time, to either execute a job or test a job. The objective is to schedule all jobs to minimise the total completion time. They also considered the problem variant where all jobs have a uniform upper limit on the processing time of a job. In other words, all jobs appear to have the same processing time. However, the true processing time may differ. The authors presented a deterministic *THRESHOLD* algorithm that is 2-competitive. They also provided a deterministic lower bound of 1.8546. Their study was technically inspired by [28, 20], and they also developed a model for modifying instances in an adversarial manner. For any algorithm, they can obtain the worst-case instance by reducing the instance properties such that the competitive ratio is non-decreasing.

In 2020, Albers et al. [2] extended the problem of Dürr et al. [8] by considering non-uniform testing times. They presented a deterministic 4-competitive *SORT* algorithm. The *SORT* algorithm always chooses between the shortest testing action and the shortest execution action. They also consider the case where preemption is allowed, that is to cancel the current operation, while preserving the progress, (this is either testing or executing) of a job at any time and start working on a new job. Then they can improve the deterministic case to be 2φ-competitive, where $\varphi \approx 1.6180$ is the golden ratio.

In 2021, the aforementioned authors extended their work by considering multiple identical machines in their paper [3]. The objective function of this paper was to minimise the makespan. They presented a deterministic algorithm named *SBS* which achieved a competitive ratio of 3.1016 for the non-preemptive case as the number of machines approaches infinity. Moreover, in the case where preemption is allowed, they presented another deterministic algorithm that achieved a competitive ratio of 2. In the scenario where testing times are uniformly equal, they proved that *SBS* is 3-competitive.

In 2022, Chen et al. [23] considered the scheduling problem with multiple identical parallel machines, aiming to minimise the total completion time. They proposed several approximation algorithms with constant competitive ratios for various special cases, including a 2φ-competitive algorithm for the case of non-uniform testing times. They demonstrated that their 2φ-competitive algorithm outperforms the previous best 4-competitive algorithm *SORT* [2].

2.5 Semi-online scheduling

In semi-online scheduling, the decision-maker has access to additional information about the input, such as the order of jobs, the sum of all processing times, or the largest processing time. In contrast, the fully online model only reveals information as it becomes available. Albers and Hellwig (2012) studied the semi-online scheduling problem [4] in which the scheduler knows the sum of the jobs' processing time at any given time. They showed an improved lower bound for the objective of makespan minimisation and proved that no deterministic semi-online algorithm can achieve a competitive ratio smaller than 1.585. This

work significantly reduced the gap between the previous lower bound of the competitive ratio, which was 1.565, and the upper bound of 1.6.

In 2022, Dwibedy and Mohanty conducted an extensive survey on semi-online scheduling [9]. They noted that only two insightful survey articles exist in the literature. The first one, by Albers in 2013 [1], discussed the concept of non-preemptive semi-online scheduling for identical machines with a makespan minimisation objective. Albers highlighted three primitive additional pieces of information: total processing time, job reassignment, and buffer reordering. The second survey, by Epstein in 2018 [10], focused on non-preemptive semi-online scheduling variants on uniformly related machines and non-trivial cases of identical machines with a makespan minimization objective. Epstein explored several well-known additional pieces of information. Dwibedy and Mohanty introduced a specific terminology, Extra Piece of Information (EPI), to indicate the concept of having additional pieces of information. The uniqueness of their survey lies in their attempt to classify the literature on semi-online scheduling based on EPI, facilitating identification of related works for various setups.

3 Preliminaries

3.1 Problem definition

The problem of scheduling with explorable uncertainty is defined as follows; We are given an instance I with n jobs and a weight set W where $W \subset \mathbb{N}$. Each job j has a uniform upper limit on the processing time $\bar{u} \in \mathbf{Q}^+$, denoted as the *predicted* processing time. Besides the predicted processing time, each job has also a *true* processing time $p_j \in \mathbf{Q}^+$, which is only revealed after testing. The true processing time is upper-bounded by the predicted processing time ($0 \leq p_j \leq \bar{u}$). Furthermore, each job j has a weight $w_j \in W$ which is not affected by the testing. The testing time for every job j is exactly one unit of time ($t_j = 1$). With all the given properties, we describe a job j as a tuple (u_j, p_j, t_j, w_j). We can decide for each job whether to test it or not. Given that we do not test a job, the time that the machine spends on the untested job is always the predicted processing time. In case we do test the job, spending 1 unit of time, then after its test, we can decide to either execute the job immediately or defer it. We consider the non-preemptive case. That is, while executing or testing, the machine cannot premature stop. The cost for any job j is the completion time C_j times the weight w_j. The goal is to schedule all jobs on a single machine such that we minimise the objective function of the total weighted completion time ($\sum_{j=1}^{n} C_j \times w_j$).

We consider two cases of the scheduling problem. The first one is the special case where we only have two weights: 1|semi-online, $t_j = 1$, $u_j = \bar{u}$, $w_j \in \{1, w_1\}| \sum_j C_j w_j$, and in the second, a more generalised, case we have multiple weights: 1|semi-online, $t_j = 1$, $u_j = \bar{u}$, $w_j \in \mathbb{N}| \sum_j C_j w_j$.

3.2 Notations and properties

The optimal (offline) solution. If the true processing time p_j is known beforehand, then it is easy to construct an optimal schedule. Testing and executing a job j takes $p_j + 1$ units of time, therefore it is beneficial to test if $p_j + 1 < \bar{u}$. Since the *Weighted Shortest Processing Times* (*WSPT rule*) is optimal for the objective function of minimising the total weighted completion time, jobs in the optimal solution are executed by their weighted processing time $\frac{p_j^*}{w_j}$, where $p_j^* = min\{\bar{u}, p_j + 1\}$, in a non-decreasing order.

Instance variations. Given any instance I, we create special instance variations. These instance variations consider the same set of jobs as I, with modified tuple values. In other words, for every job $j \in I$, its corresponding job in $\hat{I}$ has a tuple $(\bar{u}, p_j, w_j, t_j = 0)$. Similarly, for every job $j \in I$, the corresponding job $\in \bar{I}$ has a tuple $(\bar{u}, p_j = 0, w_j, t_j)$.

Schedule and algorithmic cost. Given any instance I and any deterministic algorithm ALG, let $ALG(I)$ denote the cost of applying algorithm ALG to instance I. We denote $S^{ALG}(I)$ as the schedule produced by ALG(I). If the context is clear or when we specifically specify it then, we denote the schedule as $ALG(I)$.

Furthermore, we denote that a solution produces a schedule with a specific order of execution. Given any instance I, let ALG_I be a solution with the testing strategy of ALG and the jobs are executed w.r.t the execution order in $S^{ALG}(I)$. Note that $ALG_I(I')$ only works if and only if $I' \subset I$. In other words, for any job $j \in I'$ there is a corresponding job $j \in I$ that may have a different tuple value. Observe that all the notations are also applicable to the optimal solution OPT since we replace ALG with OPT.

Heavy and unit jobs. All jobs j with a weight $w_j = 1$ are also considered to be weighted jobs. To differentiate these weighted jobs from the other ones, we denote the jobs with a weight $w_j > 1$ as *heavy jobs* and jobs with $w_j = 1$ as *unit jobs*.

Trivial and non-trivial jobs. One of the non-trivial decisions is whether we test a job or not. The only case where this decision is trivial is when the testing time of a job j is at least the upper limit $(t_j > u_j)$. Therefore, a job j is *non-trivial* if $u_j \geq 1$. Throughout the paper, we only consider non-trivial jobs.

▶ **Proposition 1.** *For all non-trivial jobs j, $p_j^* \geq p_j$.*

Proof. Consider two cases. **Case 1**: the optimal schedule test job j, $p_j^* = p_j + 1$. Therefore, $p_j^* \geq p_j$. **Case 2**: the optimal schedule does not test job j, $p_j^* = u_j$. Since $0 \leq p_j \leq u_j$ the proposition holds. ■

Performance analysis. To compare the performance of an algorithm ALG to the optimal solution we use competitive analysis. An algorithm ALG is ρ-competitive (or has competitive ratio at most ρ), if $\frac{ALG(I)}{OPT(I)} \leq \rho$ for all instances I of the problem [6].

3.3 Schedule structure

When analysing an algorithm or the optimal solution, we argue that the schedule has a certain structure with different blocks of operations such as testing or executing jobs. We calculate the cost of the different blocks separately and then sum the costs to obtain the total cost of the schedule. For example, we have an algorithm ALG that first tests all jobs, starting with the largest weighted job, and only executes a job j immediately if $p_j = 0$, otherwise, the job will be postponed. If there are no more untested jobs, then ALG executes all postponed jobs by the WSPT rule. Consider the following instance, let M and D be two disjoint sets of jobs such that $I = M \cup D$. For every job $j \in M$ the values are $(u_j = \bar{u}, p_j = \bar{u}, t_j = 1, w_j = \alpha)$. Similarly, for every job $j \in D$ the values are $(u_j = \bar{u}, p_j = 0, t_j = 1, w_j = \beta)$, where $\alpha > \beta$. The cost of $S^{ALG}(I)$ is as follows

$$ALG(I) = m(d \times \beta + m \times \alpha) + \frac{d(d+1)}{2} \times \beta + d(m \times \alpha) + \frac{m(m+1)}{2} \times \alpha \times \bar{u}$$

The first term, $m(d \times \beta + m \times \alpha)$, describes that there are m tests, that delay d β-weighted jobs and m α-weighted jobs. Recall that for every unit of delay the cost of completing a job is increased by its weight. The second term, $\frac{d(d+1)}{2} \times \beta$, describes the testing and execution of β-weighted jobs. The third term, $d(m \times \alpha)$, describes the delay cost of α-weighted jobs incurred by the testing and execution of β-weighted jobs. The last term, $\frac{m(m+1)}{2} \times \alpha \times \bar{u}$, describes the execution of the postponed jobs. We refer to Figure 1, for $S^{ALG}(I)$.

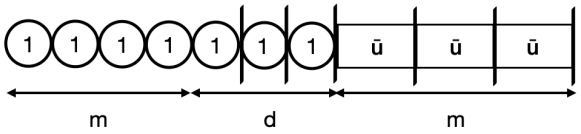

Figure 1 An illustration of a schedule produced by ALG where the circle represents a test, the rectangles represent the execution time and the black bar represents the completion of a job. We first test m jobs, this delays the execution of all jobs by m. Then we test and execute the β-weighted jobs, this will delay the execution of α-weighted jobs by d. At last, we execute the last m jobs by the WSPT rule.

4 Analysis Approach

In this section, we show that the approach in paper[8] cannot be generalised to the weighted case. Therefore, we provide a novel approach to analyse the algorithms. We also briefly mention another non-trivial approach from paper[2]. Furthermore, we provide structural properties of the optimal schedule and develop a series of lower bounds of the optimal cost which will be used in the later sections.

4.1 Schedule separation

Small upper limit. An important insight by Durr et al. was Lemma 1, regarding jobs with a small upper limit. For completeness, we attach its proof. The approach in the proofs of the competitive ratios in paper[8] was to describe the worst case ratio. Durr et al. relied on Lemma 1 to reduce worst case instances to a simple structure where the ratio is non-decreasing.

▶ **Lemma 1.** [8] *Without loss of generality, any algorithm ALG (deterministic or randomised) claiming competitive ratio ρ starts by scheduling all jobs j, untested, with $u_j < \rho$ in a non-decreasing order of u_j. Moreover, worst case instances for ALG consist solely of jobs j with $u_j \geq \rho$.*

Proof. Consider any algorithm ALG and any instance I where all testing times are equal to 1. Let $J \subset I$ be the set of jobs j which satisfy the condition $u_j < \rho$ and let $I' = I \backslash J$. Suppose we create a new algorithm *ALG'* from ALG by slightly changing the behaviour of ALG, such that ALG' first executes, without any test, all jobs $j \in I$ which satisfy the condition $u_j < \rho$ in non-decreasing order of the predicted processing time. Afterwards, ALG' schedules all remaining jobs $j \in I'$ according to the strategy of ALG. In the worst case, all jobs $j \in J$ have a true processing time of 0. We claim that the competitive ratio of ALG' is

at most ρ for scheduling jobs $j \in J$. First, observe that the claim holds for any job $j \in J$.

$$\frac{ALG'(j)}{OPT(j)} = \frac{u_j}{p_j^*} = \frac{u_j}{1} \leq \rho \qquad\qquad u_j < \rho$$

Recall that $p_j^* = min\{p_j + 1, u_j\}$. Now, for the set J

$$\frac{ALG'(J)}{OPT(J)} = \frac{\sum_{j=1}^{|J|}(\sum_{i=1}^{j} u_j)}{\sum_{j=1}^{|J|}(\sum_{i=1}^{j} p_i^*)} < \frac{\sum_{j=1}^{|J|} j \times \rho}{\sum_{j=1}^{|J|} j} \leq \rho$$

Let *len(A(I))* be the makespan of a schedule produced by algorithm A on an instance I, then by the same argument we have

$$\frac{len(ALG'(J))}{len(OPT(J))} = \frac{\sum_{j=1}^{|J|} u_j}{\sum_{j=1}^{|J|} p_j^*} < \frac{|J| \times \rho}{|J|} \leq \rho$$

Let $k = |I'|$, since I' only contains jobs j with an upper limit larger or equal to ρ we have that $ALG'(I') = ALG(I')$. Therefore,

$$ALG'(I) = ALG'(J) + k \times len(ALG'(J)) + ALG(I')$$
$$OPT(I) = OPT(J) + k \times len(OPT(J)) + OPT(I')$$

where $k \times len(ALG'(J))$ is the execution delay of jobs $j \in I'$. Note that the scheduling order is first executing all jobs $j \in J$, without any test, in non-decreasing order of weighted processing time. Afterwards, jobs $j \in I'$ are scheduled according to the strategy of ALG.

Observe that when adding the cost of scheduling jobs $j \in I'$ to both solutions in Equation (1), such that we have $\frac{ALG'(I)}{OPT(I)}$, the ratio is non-decreasing if

$$\frac{ALG'(I')}{OPT(I')} \geq \frac{ALG'(J) + k \times len(ALG'(J))}{OPT(J) + k \times len(OPT(J))}$$

Suppose that for all instances I, the ratio of $\frac{ALG(I)}{OPT(I)} \leq \rho$. Then we conclude the following: Adding the cost of scheduling jobs $j \in I'$ to both solutions in Equation (1), then by our assumption the ratio of $\frac{ALG'(I)}{OPT(I)}$ is at most ρ. Hence, if ALG(I) is ρ-competitive so is ALG'(I).

$$\frac{ALG'(J) + k \times len(ALG'(J))}{OPT(J) + k \times len(OPT(J))} \leq \rho \tag{1}$$

Now, suppose that for all instances I, the ratio of $\frac{ALG(I)}{OPT(I)} \geq \rho$. Then we conclude the following: Just like in the previous scenario, we add the cost of scheduling jobs $j \in I'$ to Equation (1). Observe that the ratio of $\frac{ALG'(I)}{OPT(I)}$ is non-decreasing. However, if $\frac{ALG(I)}{OPT(I)} > \rho$ then the ratio of $\frac{ALG'(I)}{OPT(I)}$ is always strictly smaller than $\frac{ALG(I)}{OPT(I)}$. Therefore, the worst case for ALG' is instances consisting only of jobs with $u_j \geq \rho$.

■

Intuitively, to extend the approach of Dürr et al. for the weighted case, we consider the weighted processing time $\frac{u_j}{w_j}$ as a threshold. However, Albers et al. [2] already proved that the approach does not generalise for the with non-uniform testing times. By using similar reasoning and arguments as in paper[2], we show through a counter-example that the approach of scheduling all jobs j untested with $\frac{u_j}{w_j} < \rho$ first leads to a bad schedule.

▶ **Lemma 2.** *Any deterministic algorithm ALG that starts by scheduling all untested jobs j with $\frac{u_j}{w_j} < \rho$, where $\rho \geq 0$, in non-decreasing order of $\frac{u_j}{w_j}$ the competitive ratio is unbounded.*

Proof. Given an integer m and a small real number $\epsilon > 0$, we consider m number of jobs, where the values of each job j are $(u_j = \rho^2, p_j = 0, w_j = \rho, t_j = 1)$. These jobs are not going to be scheduled untested at the beginning of the schedule, since they do not satisfy the condition $(\frac{u_j}{w_j} < \rho)$. Now consider that we have one extra job l, with $(u_l = m^2, p_l = 0, w_l = \frac{m^2}{\rho} + \epsilon, t_l = 1)$. Any algorithm which obeys the small upper-limit rule schedules job l untested, since the weighted processing time is smaller than ρ:

$$\begin{aligned}\frac{u_l}{w_l} &= \frac{m^2}{\frac{m^2}{\rho} + \epsilon} \\ &= \frac{m^2}{\frac{m^2 + \rho \times \epsilon}{\rho}} \\ &= \frac{m^2 \times \rho}{m^2 + \rho \times \epsilon} < \rho\end{aligned}$$

Afterwards, assume that the remaining jobs will be scheduled optimally. The completion time of job l is $C_l = m^2$ and for the other jobs, we have $C_j = m^2 + j$. The total cost of the algorithm is:

$$\begin{aligned}ALG &= C_l \times w_l + \sum_{j=1}^{m} C_j \times w_j \\ &= C_l \times w_l + \sum_{j=1}^{m} (m^2 + j) \times \rho \\ &= C_l \times w_l + m^3 \times \rho + \frac{m \times (m+1)}{2} \times \rho \\ &= C_l \times w_l + m^3 \times \rho + \frac{m \times \rho}{2} + \frac{m^2 \times \rho}{2} \\ &= m^2 \times (\frac{m^2}{\rho} + \epsilon) + m^3 \times \rho + \frac{m \times \rho}{2} + \frac{m^2 \times \rho}{2} \\ &= m^4 \times \frac{1}{\rho} + m^3 \times \rho + m^2 \times (\frac{\rho}{2} + \epsilon) + m \times \frac{\rho}{2}\end{aligned}$$

On the contrary, the optimal solution starts with testing and executing job l. After scheduling job l, it will test and execute the remaining jobs in any order. Since all m jobs have the same true processing time, the scheduling order within m jobs is irrelevant. The

total cost of the optimum solution is:

$$\begin{aligned} OPT &= C_l \times w_l + \sum_{j=1}^{m} C_j \times \omega_j \\ &= 1 \times (\frac{m^2}{\rho} + \epsilon) + \sum_{j=1}^{m} C_j \times \omega_j \\ &= 1 \times (\frac{m^2}{\rho} + \epsilon) + \sum_{j=1}^{m} (1+j) \times \rho \\ &= (\frac{m^2}{\rho} + \epsilon) + m \times \rho + (\frac{m^2}{2} + \frac{m}{2}) \times \rho \\ &= m^2 \times (\frac{1}{\rho} + \frac{\rho}{2}) + m \times (\rho + \frac{\rho}{2}) + \epsilon \end{aligned}$$

Now, if we let $m \longrightarrow \infty$ and $\epsilon \longrightarrow 0$ then the competitive ratio is

$$\frac{ALG(I)}{OPT(I)} \longrightarrow \infty$$

■

Lower bound approach. By Lemma 2, we have shown that Lemma 1 cannot be generalised to the weighted case. Our approach, *Schedule separation*, exploits the structural behaviour of any algorithm such that, we use the lower bound of the optimal solution to bound the algorithmic cost from above. More specifically, we split the structure of a schedule into three main parts: the delay cost of completing a job incurred by testing (TD), the delay cost of completing a job incurred by some execution of another job (ED) and the cost of executing the jobs (E). Within a main part, there may exist several smaller components. We upper bound all the different parts and sum them all up such that we have an upper bound on the algorithm.

Framework Schedule separation

Any deterministic algorithm ALG is $(\beta + \delta + \theta)$-competitive if $ALG_{TD} \leq \beta \times OPT$, $ALG_{ED} \times \leq \delta \times OPT$ and $ALG_E \leq \omega \times OPT$ where β, δ and $\theta \ \geq 0$.

An advantage of such an approach is detecting the parts that are difficult to upper bound or increase the overall algorithmic cost the most. Such information tells us which part of the algorithm we should analyse more in-depth. The disadvantage is you need to find the lower bound on the optimal solution, which is already difficult to find, and upper bounding only a specific part can lead to an overestimation of the competitive ratio.

Cross-examining. Albers et al. [2] presented a non-trivial approach called *cross-examining* for minimising the total completion time with non-uniform testing times. This approach shows great potential, as the authors have proven that an algorithm, called *SORT*, is 4-competitive. We observed that the algorithm SORT is at most $(2 \times \frac{w_{max}}{w_{min}} + 2)$-competitive. However, we were unable to generalise this approach, and therefore, we adopted the Schedule separation framework as an alternative.

4.2 Optimal solution behaviour

An interesting behaviour of the optimal solution, which is easily overlooked, is that if there is a test for any job j, then the execution of job j occurs immediately after its test

▶ **Theorem 3.** *If the optimal solution decides to test any job j, then the execution of job j occurs immediately after its test.*

Proof. Assume on the contrary, that there is an optimal solution S that does not always execute the jobs after its test. Let job k be the first job in the schedule of S where its test and execution are not adjacent. Then, there must be at least one operation (either a test or an execution of another job l) between the test and the execution of k. By performing swaps on the operations of job k and l, we show that the schedule S, compared to the new schedule S', cannot be optimal.

Case 1: There is a test of job l right before the execution of job k. Suppose that we swap the test of job l with the execution of job k. Observe that all the other jobs remain in their original position. The completion time of job k is, in S', $C_k^{'} = T + t_k + p_k < T + t_k + t_l + p_k = C_k$, where T is the time before testing job k, see Figure 2. Hence, S is not optimal.

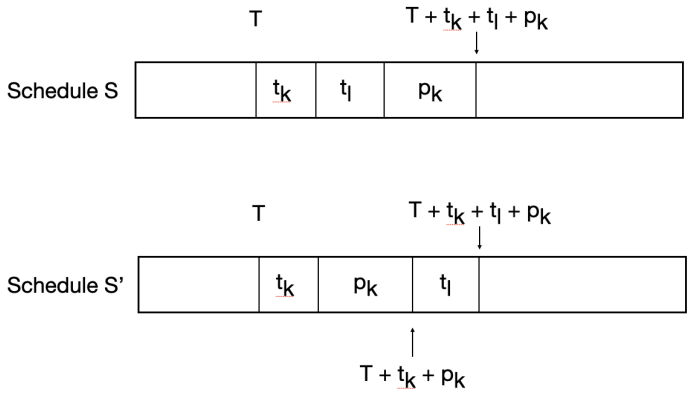

Figure 2 Two schedules, where in schedule s there is a test t_l between the test (t_l) and execution (p_k) of job k. After the swap of t_l and p_k, we have the newly obtained schedule S'. T is denoted as the time before we start operation t_k.

Case 2: There is an execution of job l right before the execution of job k. Since in this problem set the execution of a job j can be either u_j or $p_j + t_j$, therefore let the execution of job l be $p_l^* = min(u_l, p_l + t_l)$. We only swap the execution of job l with the execution of job k, if and only if $\frac{p_k + t_k}{w_k} \leq \frac{p_l^*}{w_l}$. Assume in schedule S we swapped p_l^* with t_k. Now, consider a schedule S' where we swapped p_l^* with p_k, we refer to Figure 3. Under schedule S, the cost for jobs k and l, respectively, is

$$(T + p_l^* + t_k + p_k) \times w_k + (T + p_l^*) \times w_l \tag{2}$$

Similarly under schedule S'

$$(T + t_k + p_k) \times w_k + (T + t_k + p_k + p_l^*) \times w_l \tag{3}$$

We subtract Equation (2) from Equation (3) to obtain

$$(T + p_l^* + t_k + p_k) \times w_k + (T + p_l^*) \times w_l - (T + t_k + p_k) \times w_k - (T + t_k + p_k + p_l^*) \times w_l$$

By simplifying this equation, the majority of the terms cancel each other out. We obtain

$$w_k \times p_l^* - w_l \times (t_k + p_k)$$

As $\frac{p_k+t_k}{w_k} \leq \frac{p_l^*}{w_l}$, observe that S' is either equal to S or strictly smaller. Hence, either schedule S cannot be optimal or we can change schedule S to S' such that the execution of job k occurs immediately after its test. By the same arguments, we can prove that schedule S is optimal when $\frac{p_k+t_k}{w_k} > \frac{p_l^*}{w_l}$.

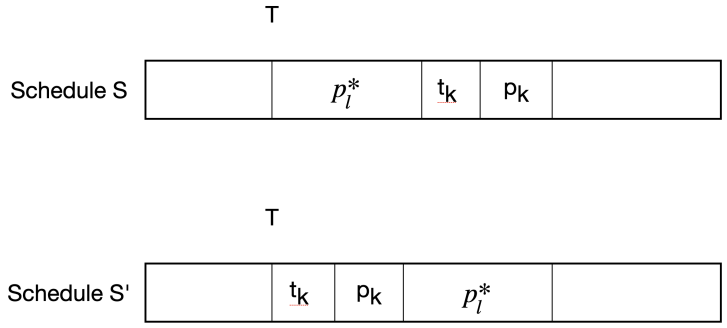

Figure 3 Two schedules, where in schedule S we swapped the execution of job l with the test of job k and in schedule S' we swapped the execution of job k with the execution of job l. T is the time before the operations of job l in S and k in S', and $p_l^* = min(u_l, p_l + t_l)$.

To show that S' is a feasible schedule in all cases, we can first assume that S is already a feasible schedule. At all times we swap two adjacent operations of two distinct jobs. Therefore, the operation order of any job j remains the same. ■

4.3 Lower bound of optimal cost

In this subsection, we provide several lower bounds of the optimal solution cost. In the proofs, we use the instance variations $\bar{I}$ and $\hat{I}$ extensively. Recall that for every job $j \in I$, its corresponding job in $\hat{I}$ has a tuple $(\bar{u}, p_j, w_j, t_j = 0)$. Similarly, for every job $j \in I$, the corresponding job $\in I$ has a tuple $(\bar{u}, p_j = 0, w_j, t_j)$. Furthermore, we also use the notation OPT_I which describes a solution following the testing strategy of OPT and executes the jobs w.r.t the execution order in $S^{OPT}(I)$. Note that $OPT_I(I')$ only works if and only if $I' \subset I$. In other words, for any job $j \in I'$ there is a corresponding job $j \in I$ that may have a different tuple value.

▶ **Lemma 4.** *Given any instance I, $OPT(I) \geq OPT_I(\hat{I})$.*

Proof. We prove Lemma 4 by showing the following claim: $\forall j$, $C_j^* \geq C_j^{OPT_I(\hat{I})}$, where $C_j^{OPT_I(\hat{I})}$ is the completion time of job $j \in \hat{I}$ w.r.t the execution order of OPT(I).

Observe that for any job $j \in I$ there is a corresponding job $j \in \hat{I}$. Therefore, the number of jobs in both sets is equal. Furthermore, $OPT_I(\hat{I})$ executes the jobs in the same order as $OPT(I)$. Hence, by Proposition 1 we have for any job j that $C_j^* = \sum_{i=1}^{j} p_i^* \geq \sum_{i=1}^{j} p_i = C_j^{OPT_I(\hat{I})}$. ■

▶ **Lemma 5.** *Given any weighted instance I with n jobs and l heavy weights ($w^{(1)} > w^{(2)} > ... > w^{(l)}$). Furthermore, let k_i be the number of jobs j with a weight w_j equal to $w^{(i)}$ where $i = 1...l$. Then, OPT is at least*

$$\sum_{j=1}^{l} \frac{k_j^2 + k_j}{2} \times w^{(j)} + \frac{(n-k)(n-k+1)}{2} + \sum_{j=1}^{l}((\sum_{i=1}^{j-1} k_i) \times k_j \times w^{(j)}) + k(n-k) \tag{4}$$

where k is the number of heavy jobs.

Proof. We prove Lemma 5 by showing that the following claims hold:

$$OPT(I) \geq OPT_I(\bar{I}) \tag{1}$$
$$OPT(\bar{I}) = Equation\ (4) \tag{2}$$

Proof of claim (1). We derive from Proposition 1 that for all jobs j, $p_j^* \geq 1$. By the execution order of $OPT_I(\bar{I})$ and the values for each job $j \in \bar{I}$,

$$C_j^* = \sum_{i=1}^{j} p_i^* \geq \sum_{i=1}^{j} 1 = C_j^{OPT_I(\bar{I})}$$

Hence, $OPT(I) \geq OPT_I(\bar{I})$. Moreover, we have $OPT(\bar{I}) \leq OPT_I(\bar{I}) \leq OPT(I)$.

Proof of claim (2). We show that the second claim holds, by observing the behaviour of OPT on the instance $\bar{I}$. Since each job j has a true processing time of 0, each job j contributes exactly one unit of time and has a weighted processing time of $\frac{1}{w_j}$. Then by the WSPT rule, we schedule the jobs in non-increasing order by their weights. So, the execution cost of heavy jobs is

$$\sum_{j=1}^{l} \frac{k_j^2 + k_j}{2} \times w^{(j)} \qquad \textit{execution of heavy jobs}$$

Similarly, the execution cost of unit jobs is

$$\frac{(n-k)(n-k+1)}{2} \qquad \textit{execution of unit jobs}$$

Each set of jobs with the same weight $w^{(j)}$ will be delayed by other sets of jobs with a weight $w^{(i)}$, where $i < j$, illustrated in Figure 4. Therefore, the execution delay for any set of jobs with a weight $w^{(j)}$ is

$$\sum_{j=1}^{l}(\sum_{i=1}^{j-1} k_i) \times k_j \times w^{(j)} \qquad \textit{execution delay for any } k_j$$

Executing all heavy jobs takes exactly k units of time. Therefore, the execution delay cost of the unit jobs is $k(n-k)$.

■

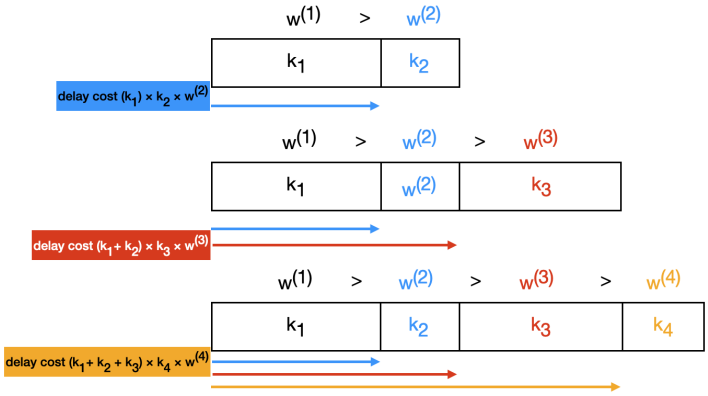

Figure 4 An illustration of jobs that are scheduled by their weights. The delay cost for every job $j \in k_i$ is the amount of delay multiplied by their weight.

▶ **Lemma 6.** *Suppose in any instance I, there are m disjoint sets J such that $J_1 \cup J_2 \cup ... \cup J_m = I$. For each set J_i, let the set H_i be $\{(u_j, p_j, w_j, t_j = 0 | \forall j \in J_i)\}$. $OPT(I)$ is at least $\sum_{i=1}^{m} OPT(H_i)$, where $i = 1...m$.*

Proof. We prove Lemma 6 by showing that the following claims hold:

$$OPT(I) \geq OPT_I(\bigcup_{i=1}^{m} H_i) \tag{1}$$

$$OPT(\bigcup_{i=1}^{m} H_i) \geq \sum_{i=1}^{m} OPT(H_i) \tag{2}$$

Proof of claim (1). For the first claim, observe that for any set H_i the value structure for every job $j \in H_i$ is the same as the jobs $j \in \hat{I}$ since both instances consider a testing time of 0. Furthermore, for any job $j \in I$ there is corresponding job $j \in (\bigcup_{i=1}^{m} H_i)$. Therefore, $(\bigcup_{i=1}^{m} H_i)$ is equivalent to $\hat{I}$. So, by Lemma 4 we have that $OPT(I) \geq OPT_I(\hat{I})$.

Proof of claim (2). For the second claim, we show that for any set H_i the inequality $OPT_{\hat{I}}(H_i) \geq OPT(H_i)$ holds. Consider a schedule produced by OPT on the instance $\hat{I}$ and any set H_i, then there may be some jobs $d \notin H_i$ that are scheduled in between or before the jobs $j \in H_i$. Let g be the index of the last executed job $j \in H_i$ in the schedule $OPT(\hat{I})$. The cost of executing the jobs $j \in H_i$ in the schedule of $OPT(\hat{I})$ is

$$OPT_{\hat{I}}(H_i) = \sum_{j=1}^{g} p_j \times (\sum_{i=j | i \in H_i}^{g} w_i)$$

The cost of executing the jobs $j \in H_i$ independent of the other jobs $d \in \hat{I} \setminus H_i$ is

$$OPT(H_i) = \sum_{j=1}^{|H_i|} w_j \times (\sum_{i=1}^{j} p_j)$$

By definition of OPT, the execution order of jobs $j \in H_i$ in the schedule $OPT_{\hat{I}}(H_i)$ is the same as in the schedule $OPT(H_i)$. Hence, if there are no jobs $d \notin H_i$ with $\frac{p_d}{w_d} > 0$ in between

or before the jobs $j \in H_i$ then $OPT_{\hat{I}}(H_i) = OPT(H_i)$. W.l.o.g if there are at least two jobs $a \in H_i$ and $b \in H_m$ with $0 < \frac{p_b}{w_b} < \frac{p_a}{w_a}$, where $i \neq m$. Then, $OPT_{\hat{I}}(H_i) > OPT(H_i)$. ■

▶ **Lemma 7.** *Given any two-weight instance I with n jobs, $OPT(I)$ is at least*

$$\frac{k^2+k}{2} \times \alpha + \frac{(n-k)(n-k+1)}{2} + k(n-k)$$

where k is the number of heavy jobs, $l = 1$ and $\alpha = w^{(l)}$.

Proof. Since we only have 1 type of heavy jobs, then we simplify the following terms of Lemma 5

$$\sum_{j=1}^{l} \frac{k_j^2 + k_j}{2} \times w^{(j)} = \frac{k^2+k}{2} \times \alpha \qquad \textit{cost for executing heavy jobs}$$

$$\sum_{j=1}^{l}((\sum_{i=1}^{j-1} k_i) \times k_j \times w^{(j)}) = 0 \qquad \textit{execution delay of heavy jobs}$$

Hence, we have our desired lower bound. ■

▶ **Lemma 8.** *Given any unit-weight instance I with n jobs, $OPT(I)$ is at least*

$$\frac{(n)(n+1)}{2}$$

Proof. Since we only have unit jobs, we ignore the terms related to the heavy jobs of Lemma 5. Hence, we have the desired lower bound. ■

5 Greedy Algorithm

In this section, we present an algorithm which follows a greedy strategy. The competitive ratio of the Greedy algorithm will be used as a performance measure for the other algorithms introduced in the later sections.

5.1 Test and execute strategy

Algorithm Greedy: Apply the following procedure to each job j in non-increasing order of w_j. Test job j and execute immediately. The cost of *Greedy* is

$$\begin{aligned} ALG^T(I) = &\sum_{j=1}^{k} w^{(j)} \times \sum_{i=1}^{j} p_i + 1 && \textit{execution of heavy jobs} \\ &+ \sum_{j=1}^{n-k} \sum_{i=1}^{j} p_i + 1 && \textit{execution of unit jobs} \\ &+ (\sum_{j=1}^{k} p_j + 1) \times (n-k) && \textit{execution delay of unit jobs} \end{aligned}$$

The most interesting aspect of Greedy is the similarity with the strategy of the optimal solution, we refer to Theorem 3. In both solutions, jobs are executed immediately after their test. We prove that Greedy is $(\bar{u}+1)$-competitive for the cases with two- and multiple weights. For simplicity, we prove it only for the case with two weights. The same reasoning and arguments can be used to prove the multiple-weight case.

▶ **Theorem 9.** *Algorithm Greedy is $(1+\bar{u})$-competitive, where $\bar{u}$ is the uniform upper limit.*

Proof. Observe that we can separate the cost of the schedule into two parts: one for only testing and one for executing. Since we always execute the job after its test, the cost of delaying the execution of other jobs is already included. Now, if we separate the two parts we have

$$\begin{aligned} ALG^T = &\sum_{j=1}^{k} \alpha \times \sum_{i=1}^{j} p_i \\ &+ \sum_{j=1}^{n-k} \sum_{i=1}^{j} p_i \\ &+ (\sum_{j=1}^{k} p_j) \times (n-k) \\ &+ \frac{k^2+k}{2} \times \alpha + \frac{(n-k)(n-k+1)}{2} + k(n-k) \qquad \textbf{testing} \end{aligned}$$

where α is the only heavy weight. By Lemma 7, the testing part (last row) is at most OPT. For the three remaining lines observe that all of them have a (p_j) term. By definition, we have for all jobs j, $p_j \leq \bar{u}$. By upper bounding the three lines with $\bar{u}$ we have

$$\begin{aligned} ALG^T = \bar{u} \times &\sum_{j=1}^{k} \alpha \times \sum_{i=1}^{j} j \\ &+ \bar{u} \times \sum_{j=1}^{n-k} \sum_{i=1}^{j} j \\ &+ \bar{u} \times (\sum_{j=1}^{k} 1) \times (n-k) \end{aligned}$$

If we simplify the terms, it is clear that the remaining terms are together at most $OPT \times \bar{u}$. Hence, *Greedy* is $(\bar{u}+1)$-competitive. ■

6 Delay-All Algorithm

In this section, we present our first algorithm where we prove the upper bound of the competitive ratio for the case with two weights and the case with multiple weights. Furthermore, we provided several inequalities that upper bounds a specific proportion of the schedule cost.

6.1 Unit-weighted case

There exists an algorithm that performs reasonably well for the unit-weighted case (where all jobs j have weight $w_j = 1$). **Algorithm Delay-All**, denoted as *DA*: First, test all jobs

in any order. If there are no more untested jobs, execute the tested jobs in non-decreasing order of their weighted processing time.

Algorithm 1 DA

$\delta_j = \varnothing$,
$T \leftarrow \varnothing$
foreach $j \in [n]$ **do**
  test j;
  add j to T;
  $\delta_j = \frac{p_j}{w_j}$;
end
EXE-WSPT(δ, T);

Algorithm 2 EXE-WSPT(δ, E)

while $E \neq \varnothing$ **do**
  choose $j_{min} \in \arg\min_{j \in E} \delta_j$;
  execute j_{min};
  remove j_{min} from E;
end

The cost of DA is

$$DA(I) = \sum_{j=1}^{n} p_j \qquad \textit{execution } DA_E$$
$$+ n^2 \qquad \textit{testing delay } DA_{TD} \tag{5}$$

▶ **Lemma 10.** *Given any unit-weight instance I with n jobs, $n^2 \leq 2 \times OPT$.*

Proof. By Lemma 8 we have the following inequalities

$$OPT \geq \frac{n(n+1)}{2}$$
$$2 \times OPT \geq n^2 + n$$
$$n^2 \leq 2 \times OPT - n$$
$$n^2 \leq 2 \times OPT$$

■

▶ **Theorem 11.** *Algorithm DA is 3-competitive.*

Proof. By the framework Schedule separation, we split the cost of DA into an execution (DA_E) and a testing delay (DA_{TD}) part, we refer to Equation (5). By Lemma 6, DA_E is at most OPT. Furthermore, by Lemma 10, DA_{TD} is at most $2 \times OPT$. Hence, we have our desired competitive ratio of 3. ■

A peculiar observation is that the upper limit does not affect the competitive ratio. So, even if the upper limit is arbitrarily large DA is still 3-competitive.

6.2 Two weights

To prove the competitive ratio for the two cases, we provide several structural properties that upper bound a specific proportion of the schedule. For some specific proportions, the upper bound proves to be an overestimation. We show that the overestimation is being utilised in the later sections. The cost of DA is

$$\boldsymbol{DA}: \sum_{j=1}^{n} w_j \times (\sum_{i-1}^{j} p_i) + n \times \sum_{j=1}^{k} \alpha + n(n-K) \tag{6}$$

where we denote the only heavy jobs as α-weighted jobs.

▶ **Lemma 12.** *Given any two-weight instance I with n jobs, $n(n-k) \leq 2 \times OPT$, where k is the number of heavy jobs.*

Proof. By Lemma 7 we have the following inequalities

$$\begin{aligned}
OPT &\geq \frac{k^2+k}{2} \times \alpha + \frac{(n-k)(n-k+1)}{2} + k(n-k) \\
2 \times OPT &\geq (k^2+k) \times \alpha + (n-k)(n-k+1) + 2 \times k(n-k) \\
&\geq (k^2+k) \times \alpha + (n-k) \times n + (n-k)(1-k) + 2 \times k(n-k) \\
n(n-k) &\leq 2 \times OPT - (k^2+k) \times \alpha - (n-k)(1-k) - 2 \times k(n-k) \\
&\leq 2 \times OPT - (k^2+k) \times \alpha - n - k^2 - 2 \times nk + nk + k + 2k^2 \\
&\leq 2 \times OPT - k^2 \times \alpha - n - nk + k^2 && \alpha > 1 \\
&\leq 2 \times OPT - k^2 \times \alpha - n && k \leq n \\
&\leq 2 \times OPT
\end{aligned}$$

■

▶ **Corollary 13.** *Given any two-weight instance I with n jobs, $OPT(I) \geq \frac{n^2+n}{2}$.*

Proof. Observe that there exist several lower bounds of the optimal cost for a given instance, although some lower bounds are weaker in the sense that the gap between the lower bound and the actual optimal cost is large. By Lemma 7 we have the following inequalities

$$\begin{aligned}
OPT &\geq \frac{k^2+k}{2} \times \alpha + \frac{(n-k)(n-k+1)}{2} + k(n-k) \\
OPT &\geq \frac{k^2+k}{2} \times \alpha + \frac{n^2+n+k^2}{2} - nk - \frac{k}{2} + nk - k^2 \\
OPT &\geq \frac{k^2+k}{2} \times \alpha + \frac{n^2+n}{2} - \frac{k^2}{2} - \frac{k}{2} \\
OPT &\geq \frac{n^2+n}{2} && \alpha > 1
\end{aligned}$$

■

In some cases, it is difficult or not so evident how to upper bound a specific part. Therefore, we present a framework *Optimal bounding* for such cases.

Framework Optimal bounding

Given any instance I and an algorithm ALG. Let A express a proportion of $ALG(I)$, D be a lower bound of OPT and r be a positive constant. If we can show that $r \times D - A \geq 0$, then the following inequality holds

$$\begin{aligned} 0 &\leq r \times OPT - D \times r \\ A &\leq r \times OPT - D \times r + A \\ A &\leq OPT \times r \end{aligned}$$

▶ **Lemma 14.** *Given any two-weight instance I with n jobs, k(n-k) $\leq \frac{1}{2} \times OPT$, where k is the number of heavy jobs.*

Proof. By the framework Optimal bounding, we prove Lemma 14 by showing that r is at least $\frac{1}{2}$ where $A = k(n-k)$ and Corollary 13 be the lower bound D.

$$\begin{aligned} & r \times D - A \\ &= \frac{n^2+n}{2} \times r - k(n-k) \\ &= \frac{n^2+n}{2} \times r + k^2 - nk \end{aligned}$$

Consider a case distinction on the number of α-weighted jobs. Let $k = \frac{n \pm c}{2}, 0 \leq c \leq n$ where $0 \leq c \leq n$. Note that $0 \leq k \leq n$

$$\begin{aligned} & \frac{n^2+n}{2} \times r + \frac{n^2 \pm 2nc + c^2}{4} - \frac{n^2 \pm nc}{2} \\ &= n^2 \times (\frac{r}{2} + \frac{1}{4} - \frac{1}{2}) + n \times (\frac{r}{2} + (\pm\frac{c}{2}) - (\pm\frac{c}{2})) + \frac{c^2}{4} \end{aligned}$$

Case $k = \frac{n+c}{2}$

$$\begin{aligned} & n^2 \times (\frac{r}{2} + \frac{1}{4} - \frac{1}{2}) + n \times (\frac{r}{2} + \frac{c}{2} - \frac{c}{2}) + \frac{c^2}{4} \\ &= n^2 \times (\frac{r}{2} + \frac{1}{4} - \frac{1}{2}) + n \times (\frac{r}{2}) + \frac{c^2}{4} \\ &\geq n^2 \times (\frac{r}{2} + \frac{1}{4} - \frac{1}{2}) \\ & r \geq \frac{1}{2} \end{aligned}$$

Case $k = \frac{n-c}{2}$

$$\begin{aligned} & n^2 \times (\frac{r}{2} + \frac{1}{4} - \frac{1}{2}) + n \times (\frac{r}{2} - \frac{c}{2} + \frac{c}{2}) + \frac{c^2}{4} \\ &= n^2 \times (\frac{r}{2} + \frac{1}{4} - \frac{1}{2}) + n \times (\frac{r}{2}) + \frac{c^2}{4} \\ &\geq n^2 \times (\frac{r}{2} + \frac{1}{4} - \frac{1}{2}) \\ & r \geq \frac{1}{2} \end{aligned}$$

■

▶ **Theorem 15.** *Algorithm DA is* $(3 + \frac{1}{2} \times \alpha)$*-competitive, where* $\alpha > 1$.

Proof. By the framework Schedule Separation, the total schedule cost of DA, Equation (6), is divided into the following parts.

$$\sum_{j=1}^{n} w_j \times (\sum_{i=1}^{j} p_i) \qquad \textit{execution } (DA_E)$$

$$n(n-k) + n \times \sum_{j=1}^{k} \alpha \qquad \textit{test delay } (DA_{TD})$$

By Lemma 6, DA_E is at most OPT. Observe that by Lemma 12 and Lemma 14, DA_{TD} is at most $(2 + \frac{1}{2} \times \alpha) \times OPT$.

$$\begin{aligned} DA_{TD} &= n(n-k) + n \times \sum_{j=1}^{k} \alpha \\ &\leq 2 \times OPT - k^2 \times \alpha + n \times \sum_{j=1}^{k} \alpha && \textit{Lemma 12} \\ &\leq 2 \times OPT - k^2 \times \alpha + nk \times \alpha \\ &\leq 2 \times OPT + k(n-k) \times \alpha \\ &\leq 2 \times OPT + \frac{1}{2} \times OPT \times \alpha && \textit{Lemma 14} \end{aligned}$$

Hence, we have the desired competitive ratio of $3 + \frac{1}{2} \times \alpha$. ■

Overestimation. An interesting observation from the competitive ratio analysis is when we consider an instance with only unit jobs. By Theorem 11, the competitive ratio is 3-competitive while according to Theorem 15 it is $(3 + \frac{1}{2})$-competitive. The gap between the competitive ratio is caused by the usage of several different upper bounds in the analysis of Theorem 15, which all have some form of overestimation.

6.3 Multiple weights

In this case, the weight of a job j can be any value of the set of natural numbers ($w_j \in \mathbb{N}$). The cost of DA is

$$\mathbf{DA(I)} = \sum_{j=1}^{n} w_j \times (\sum_{i=1}^{j} p_i) + n \times \sum_{j=1}^{k} w_j + n(n-k) \tag{7}$$

Notice that the lower bound of the optimal cost in the multiple-weight case contains a complicated summation term, we refer to Lemma 5. In Corollary 16 and Observation 17, we show another possible lower bound of the optimal cost and a way to simplify and utilise the summation term.

▶ **Corollary 16.** *Given any multiple-weight instance I with n jobs, $OPT(I)$ is at least*

$$\sum_{j=1}^{l} k_j + \frac{(n-k)(n-k+1)}{2} + k(n-k) + (\sum_{j=1}^{l}(\sum_{i=1}^{j} k_i) \times k_j) + (\sum_{j=1}^{l}(\sum_{i=1}^{j-1} k_i) \times k_j)$$

where k is the number of heavy jobs and l is the number of heavy weights.

Proof. By Lemma 5, we have the following inequalities

$$\begin{aligned} OPT(I) &\geq \sum_{j=1}^{l} \frac{k_j^2 + k_j}{2} \times w^{(j)} + \frac{(n-k)(n-k+1)}{2} + \sum_{j=1}^{l}((\sum_{i=1}^{j-1} k_i) \times k_j \times w^{(j)}) + k(n-k) \\ &\geq \sum_{j=1}^{l} k_j^2 + k_j + \frac{(n-k)(n-k+1)}{2} + 2(\sum_{j=1}^{l}(\sum_{i=1}^{j-1} k_i) \times k_j) + k(n-k) \qquad w_j \geq 2 \end{aligned}$$

Observe the term $\sum_{j=1}^{l}(\sum_{i=1}^{j-1} k_i) \times k_j$, for every summation of j we miss exactly k_j^2. Hence,

$$OPT \geq \sum_{j=1}^{l} k_j + \frac{(n-k)(n-k+1)}{2} + (\sum_{j=1}^{l}(\sum_{i=1}^{j} k_i) \times k_j) + (\sum_{j=1}^{l}(\sum_{i=1}^{j-1} k_i) \times k_j) + k(n-k)$$

■

The most important aspect of Corollary 16 is that we can create the summation $(\sum_{j=1}^{l}(\sum_{i=1}^{j} k_i) \times k_j)$. Notice, that the weights of heavy jobs are at least two since for any weight $w_j \in \mathbb{N}$ and excluding the unit weight. Furthermore, observe that the lower bound Corollary 16 is weaker than Lemma 5. However, we can preserve the weights by multiplying the optimal cost twice.

▶ **Observation 17.** *By combining and simplifying the two summations in Corollary 16 we have,*

$$(\sum_{j=1}^{l}(\sum_{i=1}^{j} k_i) \times k_j) + (\sum_{j=1}^{l}(\sum_{i=1}^{j-1} k_i) \times k_j) = k^2$$

where k is the number of heavy jobs and l is the number of heavy weights.

Proof. We prove the equality by showing a visualisation of k^2. Mathematically, $k^2 = (k_1+k_2+k_3+...+k_{l-1}+k_l)(k_1+k_2+k_3+...+k_{l-1}+k_l)$. Consider the summation $\sum_{j=1}^{l}(\sum_{i=1}^{j} k_i) \times k_j$, to complete the summation, such that we have k^2, we need at $\sum_{j=1}^{l}(\sum_{i=j}^{l} k_i) \times k_j$,s see Figure 5. Now, consider the summation $\sum_{j=1}^{l}(\sum_{i=1}^{j-1} k_i) \times k_j$. The occurrence of any value k_j is $l-j$. Therefore, we can rewrite the summation to $\sum_{j=1}^{l}(\sum_{i=j}^{l} k_i) \times k_j$ which covers exactly the missing parts of the summation $\sum_{j=1}^{l}(\sum_{i=1}^{j} k_i) \times k_j$, we refer to Figure 6. ■

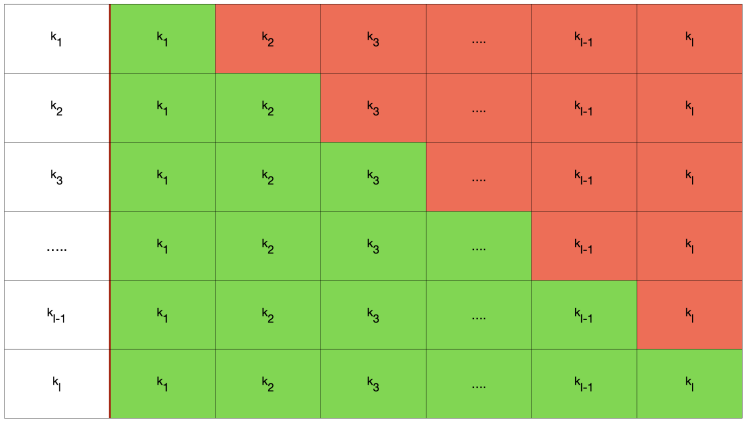

Figure 5 Visual representation of k^2, where the green squares represent $\sum_{j=1}^{l}(\sum_{i=1}^{j} k_i) \times k_j$ and the red ones are the missing values $(\sum_{j=1}^{l}(\sum_{i=j}^{l} k_i) \times k_j)$.

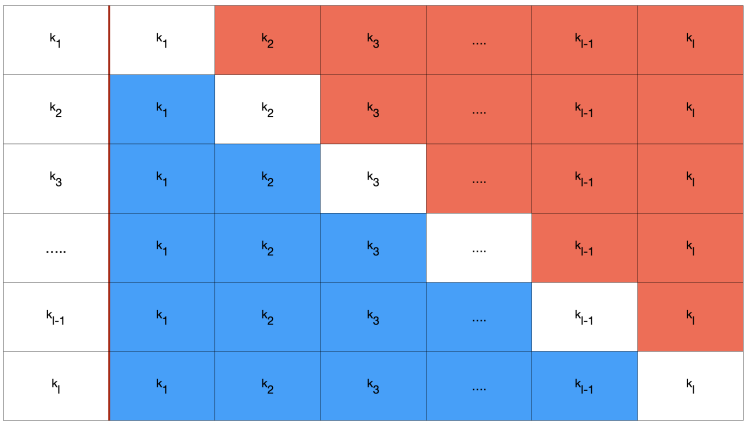

Figure 6 Visual representation of k^2, where the blue squares represent $\sum_{j=1}^{l}(\sum_{i=1}^{j-1} k_i) \times k_j$ and the red ones are $(\sum_{j=1}^{l}(\sum_{i=j}^{l} k_i) \times k_j)$

.

▶ **Lemma 18.** *Given any multiple-weight instance I with n jobs, $n(n-k) \leq 2 \times OPT$, where k is the number of heavy jobs.*

Proof. By Corollary 16 and Observation 17 we have

$$
\begin{aligned}
OPT &\geq \sum_{j=1}^{l} k_j + \frac{(n-k)(n-k+1)}{2} + k^2 + k(n-k) \\
2 \times OPT &\geq 2(\sum_{j=1}^{l} k_j) + (n-k)(n-k+1) + 2k^2 + 2k(n-k) \\
&= 2(\sum_{j=1}^{l} k_j) + n^2 + n + k^2 - k - 2nk + 2k^2 + 2nk - 2k^2 \\
&= 2(\sum_{j=1}^{l} k_j) + n^2 - nk + n + k^2 - k + nk \\
&= 2(\sum_{j=1}^{l} k_j) + n^2 - nk + n + k^2 - k + nk \\
&\geq 2(\sum_{j=1}^{l} k_j) + n^2 - nk + k^2 + nk && k \leq n \\
n^2 - nk &\leq 2 \times OPT - 2(\sum_{j=1}^{l} k_j) - k^2 - nk \\
n(n-k) &\leq 2 \times OPT
\end{aligned}
$$

■

▶ **Theorem 19.** *Algorithm DA is $(1 + 2 \times w_{max})$-competitive, where w_{max} is the largest heavy weight.*

Proof. By the framework Schedule separation, the total schedule cost of DA, Equation (7), is divided into the following parts.

$$
\begin{aligned}
&\sum_{j=1}^{n} w_j \times (\sum_{i=1}^{j} p_i) && \textbf{execution } (DA_E) \\
&+ n \times \sum_{j=1}^{k} w_j + n(n-k) && \textbf{testing delay } (DA_{TD})
\end{aligned}
$$

By Lemma 6, we have that DA_E is at most OPT. Additionally, we can use Lemma 18 to upper bound the testing delay, DA_{TD}, of the unit jobs. To upper bound the testing delay of the heavy jobs, we can substitute all the weights with the largest weight (w_{max}) in the set.

Therefore, DA_{TD} is at most $2 \times w_{max}$.

$$\begin{aligned} DA_{TD} &= n(n-k) + n \times \sum_{j=1}^{k} w_j \\ &\leq 2 \times OPT - nk + n \times \sum_{j=1}^{k} w_j && \textit{Lemma 18} \\ &\leq 2 \times OPT - nk + nk \times w_{max} && \forall j, w_j \leq w_{max} \\ &\leq (2 \times OPT - nk) \times w_{max} + nk \times w_{max} \end{aligned}$$

Hence, we have the desired competitive ratio of $1 + 2 \times w_{max}$. ■

Using the largest weight. Suppose we have a multiple-weight instance I with $l = 2$, where $w^{(1)}$ is relatively small and $w^{(2)}$ is enormous. Furthermore, let the number of heavy jobs j with $w_j = w^{(2)}$ be exactly one and the number of heavy jobs j with $w_j = w_1$ is relatively large. Then according to the competitive ratio analysis, this results in an enormous overestimation. Therefore, using the largest weight is considered to be too pessimistic.

6.4 Lower bound

We show that there exists an instance, such that DA is at least 2.4-competitive. Observe, that DA will not stop testing even if, after a test, the true processing time of a job is equal to zero. Consider an instance I with $n\delta$ jobs having a tuple value $(u_j = \bar{u}, p_j = \epsilon, t_j = 1, w_j = w_1)$ and $n(1-\delta)$ jobs having a tuple value $(u_j = \bar{u}, p_j = \beta, t_j = 1, w_j = 1)$, where $w^{(1)} \geq 2$, $0 < \epsilon < \bar{u} - 1$ and $0 < \beta < \bar{u} - 1$. The algorithmic value of DA is

$$\begin{aligned} DA(I) &= \sum_{j=1} w_j \times (\sum_{i=1}^{j} p_i) && \textit{execution} \\ &\quad + n(n\delta) \times w_1 + n(1-\delta)n && \textit{testing delay} \end{aligned}$$

Comparatively, the optimal solution will test all jobs before their execution. However, we do not know the order of the optimal schedule. This leads to a cost of $OPT(I) = \sum_{j=1}^{n} w_j \times (\sum_{i=1}^{j} p_i + 1)$. Therefore, the ratio is

$$\frac{DA(I)}{OPT(I)} = \frac{n(n\delta)w_1 + n(1-\delta)n + \sum_{j=1} w_j \times (\sum_{i=1}^{j} p_i)}{\sum_{j=1}^{n} w_j \times (\sum_{i=1}^{j} p_i + 1)}$$

Now, we let ϵ and β approach 0 and n approach ∞. We take the term to *WolframAlpha*, where we minimise the ratio

$$\frac{DA(I)}{OPT(I)} \xrightarrow[\epsilon,\beta \to 0]{n \to \infty} \begin{cases} 2 & \delta = 1 \vee \delta = 0, w^{(1)} = 3 \\ \frac{2(\delta+1)}{\delta^2+1} & 0 < \delta < 1, w^{(1)} = 2 \\ \infty & (otherwise). \end{cases}$$

When we have either only heavy jobs, with a weight value of 3, or only unit jobs, the competitive ratio is at least 2. For different values of δ and with a weight value of 2, we can observe, see Figure 7, that the largest increase in the competitive ratio (≈ 2.4) is when the fraction of heavy jobs is approximately 0.41.

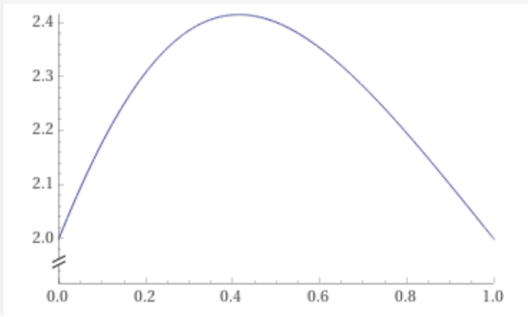

Figure 7 The function $f(\delta) = \frac{2(\delta+1)}{\delta^2+1}$ over values of δ, where δ is the number of heavy jobs

.

7 L-Delay-All Algorithm

In this section, we design a new algorithm based on the results of the algorithm DA. We prove the upper bound of the competitive ratio and show for a certain types of instances the new algorithm is constant-competitive.

7.1 Early execution strategy

By the framework Schedule separation, we can observe that the largest increase in cost of DA is the testing delay. In both cases, the two- and multiple-weight, the competitive ratio is dependent on the largest weight. A solution to reduce the testing delay cost is by executing some jobs earlier. So, we design a new algorithm L-DA that first tests all jobs of a specific weight and then executes the tested jobs by the WSPT rule. The algorithm prioritises the largest weighted jobs first since delaying these jobs can significantly increase the cost. One of the advantages of the structural properties provided in Section 6 is that they are dependent on the instance type. Therefore, most of the properties are applicable for L-DA.

Algorithm 3 L-DA

$K \leftarrow [int : \varnothing], \quad \delta_j = \varnothing$
foreach $j \in [n]$ **do**
  add j to $K[w_j]$;
end
$keysSorted \leftarrow sort(K.keys, non - increasing)$

foreach $key \in keysSorted$ **do**
  $T = \varnothing$;
  foreach $j \in K[key]$ **do**
    test j;
    add j to T;
    $\delta_j = \frac{p_j}{w_j}$;
  end
  EXE-WSPT(δ, T);
  $i = i$ +1;
end

The algorithm L-DA is divided into two phases. In the first phase, it sorts all jobs into sets K based on their weights. In the second phase, we apply the following procedure to each set K in non-increasing order of weight. Start testing all jobs and if there are no more untested jobs, execute the tested jobs by the WSPT rule.

7.2 Two weights

The cost of *L-DA* is as follows

$$\textbf{\textit{L-DA}} : \sum_{j=1}^{k} \alpha \times (\sum_{i-1}^{j} p_i) + \sum_{j=1}^{n-k} (\sum_{i=1}^{j} p_i) + k \times \sum_{j=1}^{k} \alpha + (n + \sum_{j=1}^{k} p_j) \times (n-k) \tag{8}$$

where we denote the only heavy jobs as α-weighted jobs.

▶ **Corollary 20.** *Given any two-weight instance I with n jobs,*

$$(n-k) \times \sum_{j=1}^{k} p_j \leq \frac{1}{2} \times \bar{u} \times OPT$$

where k is the number of heavy jobs and $\bar{u}$ is the uniform upper limit.

Proof. By simplifying the term $((n-k) \times \sum_{j=1}^{k} p_j)$ from Equation (6) and by the upper bound of the processing time ($0 \leq p_j \leq \bar{u}$), we have the following inequalities

$$(n-k) \times \sum_{j=1}^{k} p_j \leq (n-k)k \times \bar{u} \qquad p_j \leq \bar{u}$$

$$(n-k)k \times \bar{u} \leq (\frac{1}{2} \times OPT) \times \bar{u} \qquad \textit{Lemma } 14$$

■

▶ **Theorem 21.** *Algorithm L-DA is $(3 + \frac{1}{2} \times \bar{u})$-competitive, where $\bar{u}$ is the uniform upper limit.*

Proof. By the framework Schedule separation, the total schedule cost of L-DA, Equation (8), is divided into the following components.

$$\sum_{j=1}^{k} \alpha \times (\sum_{i=1}^{j} p_i) + \sum_{j=1}^{n-k} \sum_{i=1}^{j} p_i \qquad \textit{execution } (\textit{L-DA}_E)$$

$$k \times \sum_{j=1}^{k} \alpha + n(n-k) \qquad \textit{testing delay } (\textit{L-DA}_{TD})$$

$$(n-k) \times \sum_{j=1}^{k} p_j \qquad \textit{execution delay } (\textit{L-DA}_{ED})$$

By Lemma 6, L-DA$_E$ is at most OPT. Observe that by simplifying L-DA$_{TD}$, then according to Lemma 12 we have

$$L\text{-}DA_{TD} = k^2 \times \alpha + n(n-k) \leq 2 \times OPT - k^2 \times \alpha$$

Therefore, L-DA$_{TD}$ is at most $2 \times OPT$. By definition of the processing time, the makespan of executing all α-weighted jobs is at most $k \times \bar{u}$. Therefore, by Corollary 20, L-DA$_{ED}$ is at most $\frac{1}{2} \times OPT \times \bar{u}$. Hence, we have the desired competitive ratio of $3 + \frac{1}{2} \times \bar{u}$. ■

7.3 Multiple weights

The cost of L-DA is defined as follows

$$\begin{aligned} \boldsymbol{L\text{-}DA}(\mathbf{I}) = &\sum_{j=1}^{l} w^{(j)} \times (\sum_{i-1}^{k_j} p_i) + \sum_{j=1}^{n-k} (\sum_{i=1}^{j} p_i) \\ &+ \sum_{j=1}^{l} (\sum_{i=1}^{j} k_i) \times k_j \times w^{(j)} + \sum_{j=1}^{l} (\sum_{i=1}^{j-1} m_i) \times k_j \times w^{(j)} \\ &+ (n + \sum_{j=1}^{k} p_j) \times (n-k) \end{aligned} \tag{9}$$

where $i = 1...l$ and m_i is the total time that we spend on executing jobs j with a weight $w_j = w^{(i)}$.

Notice that in the multiple-weight case, the heavy jobs are now delayed by the execution and testing of other heavy jobs, we refer to Equation (9). The execution delay of heavy jobs, incurred by executing other heavy jobs, is upper-bounded by $\bar{u}$.

▶ **Observation 22.** *The execution cost of heavy jobs is upper-bounded by $\bar{u}$,*

$$\sum_{j=1}^{l} (\sum_{i=1}^{j-1} m_i) \times k_j \times w^{(j)} \leq (\sum_{j=1}^{l} (\sum_{i=1}^{j-1} k_i) \times k_j \times w^{(j)}) \times \bar{u}$$

where k is the number of heavy jobs, m_i is the makespan of executing all jobs with a weight $w_j = w^{(i)}$ and $\bar{u}$ is the upper limit.

Proof. From Equation (9), observe that the number of jobs j with a weight $w_j = w^{(i)}$ is k_i and the true processing time is at most $\bar{u}$. Therefore, m_i is at most $k_i \times \bar{u}$.

■

▶ **Lemma 23.** *Given any multiple-weight instance I with n jobs,*

$$\sum_{j=1}^{l} (\sum_{i=1}^{j} k_i) \times k_j \times w^{(j)} + n(n-k) \leq 2 \times OPT$$

where k is the number of heavy jobs and l is the number of heavy weights.

Proof. By Lemma 5 and Corollary 16, we have the following inequalities

$$OPT \geq \sum_{j=1}^{l} \frac{k_j^2 + k_j}{2} \times w^{(j)} + \frac{(n-k)(n-k+1)}{2} + \sum_{j=1}^{l}((\sum_{i=1}^{j-1} k_i) \times k_j \times w^{(j)}) + k(n-k)$$

$$\begin{aligned} 2 \times OPT \geq & \sum_{j=1}^{l}(\sum_{i=1}^{j} k_i) \times k_j \times w^{(j)} + \sum_{j=1}^{l}(\sum_{i=1}^{j-1} k_i) \times k_j \times w^{(j)} + \sum_{j=1}^{l} k_j \times w^{(j)} \\ & + (n-k)(n-k+1) + 2k \times (n-k) \end{aligned}$$

Observe that if we add the term $-n(n-k)+n(n-k)$, to the right-hand side of the inequality and show that $(n-k)(n-k+1) + 2k \times (n-k) - n(n-k) \geq 0$. Then, Lemma 23 holds.

$$\begin{aligned} & (n-k)(n-k+1) + 2k \times (n-k) - n(n-k) \\ & = n(n-k) - nk + n + k^2 - k + 2nk - 2k^2 - n(n-k) \\ & = -nk + n + k^2 - k + 2nk - 2k^2 \\ & = n - k + nk - k^2 \\ & \geq 0 \qquad\qquad k \leq n \end{aligned}$$

Therefore,

$$\begin{aligned} 2 \times OPT \geq & \sum_{j=1}^{l}(\sum_{i=1}^{j} k_i) \times k_j \times w^{(j)} + \sum_{j=1}^{l}(\sum_{i=1}^{j-1} k_i) \times k_j \times w^{(j)} + \sum_{j=1}^{l} k_j \times w^{(j)} \\ & + (n-k)(n-k+1) + 2k \times (n-k) \\ \geq & \sum_{j=1}^{l}(\sum_{i=1}^{j} k_i) \times k_j \times w^{(j)} + \sum_{j=1}^{l}(\sum_{i=1}^{j-1} k_i) \times k_j \times w^{(j)} + \sum_{j=1}^{l} k_j \times w^{(j)} \\ & + (n-k)(n-k+1) + 2k \times (n-k) + n(n-k) - n(n-k) \end{aligned}$$

$$\begin{aligned} n(n-k) + \sum_{j=1}^{l}(\sum_{i=1}^{j} k_i) \times k_j \times w^{(j)} \leq & 2 \times OPT - \sum_{j=1}^{l}(\sum_{i=1}^{j-1} k_i) \times k_j \times w^{(j)} - \sum_{j=1}^{l} k_j \times w^{(j)} \\ & - ((n-k)(n-k+1) + 2k \times (n-k) - n(n-k)) \end{aligned}$$

■

From Equation (9), we observe that L-DA has a cost term of $(\sum_{j=1}^{k} p_j) \times (n-k)$. In the two-weight case, we upper-bounded the same cost term by Corollary 20. We have not proven this for the multiple-weight case.

▶ **Lemma 24.** *Given any multiple-weight instance I with n jobs, $k(n-k) \leq \frac{8}{12} \times OPT$, where k is the number of heavy jobs.*

Proof. By the framework Optimal bounding, we prove Lemma 14 by showing that r is at least $\frac{8}{12}$ where $A = k(n-k)$ and by Corollary 16 and Observation 17 we have our lower bound D

$$OPT \geq \sum_{j=1}^{l} k_j + \frac{(n-k)(n-k+1)}{2} + k(n-k) + k^2$$
$$OPT \geq k + \frac{(n-k)(n-k+1)}{2} + k(n-k) + k^2$$
$$OPT \geq k + \frac{n^2+n+k^2-k}{2} - nk + nk - k^2 + k^2$$
$$OPT \geq \frac{n^2+n+k^2-k}{2}$$

Consider a case distinction on the number of heavy jobs.

$$r \times \frac{n^2+n+k+k^2}{2} - (k(n-k))$$
$$= r \times \frac{n^2+n+k^2+k}{2} + k^2 - nk$$
$$r \times (\frac{n^2+n}{2} + \frac{n^2 \pm 2nc + c^2}{8} + \frac{n \pm c}{4}) + \frac{n^2 \pm 2nc + c^2}{4} - \frac{n^2 \pm nc}{2}$$

Let $k = \frac{n \pm c}{2}, 0 \leq c \leq n$ where $0 \leq c \leq n$. Note that $0 \leq k \leq n$

$$n^2 \times (\frac{r}{2} + \frac{r}{8} + \frac{1}{4} - \frac{1}{2}) + n \times (\frac{r}{2} \pm r \times \frac{c}{4} + \frac{r}{4} \pm \frac{c}{2} - (\pm\frac{c}{2})) + c \times (r \times \frac{c}{8} + \frac{c}{4} \pm \frac{1}{4})$$

Case $k = \frac{n+c}{2}$

$$n^2 \times (\frac{r}{2} + \frac{r}{8} + \frac{1}{4} - \frac{1}{2}) + n \times (\frac{r}{2} + r \times \frac{c}{4} + \frac{r}{4} + \frac{c}{2} - (+\frac{c}{2})) + c \times (r \times \frac{c}{8} + \frac{c}{4} + \frac{1}{4})$$
$$= n^2 \times (\frac{r}{2} + \frac{r}{8} - \frac{1}{4}) + n \times (\frac{r}{2} + r \times \frac{c}{4} + \frac{r}{4}) + c \times (r \times \frac{c}{8} + \frac{c}{4} + \frac{1}{4})$$
$$\geq n^2 \times (\frac{r}{2} + \frac{r}{8} - \frac{1}{4})$$
$$r \geq \frac{8}{20}$$

Case $k = \frac{n-c}{2}$

$$n^2 \times (\frac{r}{2} + \frac{r}{8} + \frac{1}{4} - \frac{1}{2}) + n \times (\frac{r}{2} - r \times \frac{c}{4} + \frac{r}{4} - \frac{c}{2} - (-\frac{c}{2})) + c \times (r \times \frac{c}{8} + \frac{c}{4} - \frac{1}{4})$$
$$= n^2 \times (\frac{r}{2} + \frac{r}{8} + \frac{1}{4} - \frac{1}{2}) + n \times (\frac{r}{2} - r \times \frac{c}{4} + \frac{r}{4}) + c \times (r \times \frac{c}{8})$$

Observe that bounding the term $(n \times (-r \times \frac{c}{4}))$ is difficult with only (n) terms. By definition of c, we treat the term $(n \times (-r \times \frac{c}{4}))$ as a (n^2). Therefore, we have $n^2 \times (r \times \frac{5}{8} - \frac{1}{4} - r \times \frac{1}{4})$. Hence, $r \geq \frac{8}{12}$. ■

▶ **Theorem 25.** *Algorithm L-DA is* $(3 + \frac{5}{3} \times \bar{u})$*-competitive, where* $\bar{u}$ *is the uniform upper limit.*

Proof. By the framework Schedule separation, the total schedule cost of L-DA, Equation (9), is divided into the following components.

$$\sum_{j=1}^{l} w^{(j)} \times (\sum_{i=1}^{k_j} p_i) + \sum_{j=1}^{n-k} (\sum_{i=1}^{j} p_i) \qquad \textit{execution (L-DA}_E\textit{)}$$

$$+ \sum_{j=1}^{l} (\sum_{i=1}^{j} k_i) \times k_j \times w^{(j)} + n(n-k) \qquad \textit{testing delay (L-DA}_{TD}\textit{)}$$

$$+ \sum_{j=1}^{l} (\sum_{i=1}^{j-1} m_i) \times k_j \times w^{(j)} + (n-k) \times (\sum_{j=1}^{k} p_j) \qquad \textit{execution delay (L-DA}_{ED}\textit{)}$$

By applying Lemma 6 and Lemma 23, we can see that L-DA$_E$ is at most OPT and L-DA$_{TD}$ is at most $2 \times OPT$, respectively. Furthermore, according to Lemma 24, the execution delay of the unit jobs is at most $\frac{8}{12} \times OPT \times \bar{u}$. As for the execution delay of the heavy jobs, we simplified the summation term using Observation 22 and by Lemma 5 the upper bound is at most $OPT \times \bar{u}$. Therefore, L-DA$_{ED}$ is at most $\frac{5}{3} \times OPT \times \bar{u}$.

$$\begin{aligned}
\textit{L-DA}_{ED} &= \sum_{j=1}^{l} (\sum_{i=1}^{j-1} m_i) \times k_j \times w^{(j)} + (n-k) \times (\sum_{j=1}^{k} p_j) \\
&\leq \sum_{j=1}^{l} (\sum_{i=1}^{j-1} m_i) \times k_j \times w^{(j)} + (n-k)k \times \bar{u} && p_j \leq \bar{u} \\
&\leq \sum_{j=1}^{l} (\sum_{i=1}^{j-1} m_i) \times k_j \times w^{(j)} + \frac{8}{12} \times OPT \times \bar{u} && \textit{Lemma 24} \\
&\leq (\sum_{j=1}^{l} (\sum_{i=1}^{j-1} k_i) \times k_j \times w^{(j)}) \times \bar{u} + \frac{8}{12} \times OPT \times \bar{u} && \textit{Observation 22} \\
&\leq (OPT) \times \bar{u} + \frac{8}{12} \times OPT \times \bar{u} && \textit{Lemma 5} \\
&\leq \frac{5}{3} \times OPT \times \bar{u}
\end{aligned}$$

Hence, we have the desired competitive ratio of $3 + (\frac{5}{3} \times \bar{u})$. ■

Greedy algorithm better. For the case with multiple weights, we can observe that both algorithms DA and L-DA increases in ratio. Although, the increase of DA is much more than L-DA. However, the competitive ratio of L-DA is still worse compared to the Greedy.

7.4 Lower bound

We show that L-DA is at least 3.4-competitive. Intuitively, the worst-case instance for L-DA is when all heavy jobs have a true processing time equal to the upper limit. While the unit jobs have a true processing time equal to zero. In essence, the order of execution and the decision to test the heavy jobs are sub-optimal.

Consider an instance I with $n(1-\delta)$ jobs have a tuple value of $(u_j = \bar{u}, p_j = \epsilon, t_j = 1, w_j = 1)$ and $n\delta$ jobs have a tuple value of $(u_j = \bar{u}, p_j = u_j, t_j = 1, w_j = w_1)$, where $w_1 \geq 2$, $\bar{u} > w_1$ and $\epsilon + 1 < \bar{u}$. The algorithmic cost is

$$\begin{aligned} \textit{L-DA(I)} = (n^2\delta^2)w_1 + \frac{n\delta(n\delta+1)}{2} \times \bar{u} \times w_1 \quad & \textit{testing delay and execution of heavy jobs} \\ + (n\delta) \times \bar{u} \times (1-\delta)n \quad & \textit{execution delay of unit jobs} \\ + n(1-\delta)n + \frac{(1-\delta)n((1-\delta)n+1)}{2} \quad & \textit{testing delay and execution of unit jobs} \end{aligned}$$

The optimal schedule will only test the unit jobs and execute the heavy jobs directly. Now, let ϵ approach 0 such that the optimal schedule will execute the unit jobs first.

$$\begin{aligned} OPT(I) = \frac{(1-\delta)n((1-\delta)n+1)}{2} \quad & \textit{unit jobs testing and execution} \\ + (1-\delta)n(\delta)n \times w_1 + \frac{\delta n(\delta n+1)}{2} \times \bar{u} \times w_1 \quad & \textit{heavy jobs delay and execution} \end{aligned}$$

We let n approach ∞ and take the term to *WolframAlpha* where we minimise the ratio

$$\begin{aligned} \frac{\textit{L-DA(I)}}{OPT(I)} \xrightarrow[\epsilon \to 0]{n \to \infty} & \frac{2\delta^2 w_1 + \delta^2\bar{u}w_1 + 2\delta(1-\delta)\bar{u} + 2(1-\delta)}{(1-\delta)^2 + 2\delta(1-\delta)w_1 + \delta^2\bar{u}w_1} \\ & \frac{2 + 2\delta^2 w_1 + \delta^2\bar{u}w_1 + 2\delta w_1 - 2\delta^2\bar{u} - 2\delta}{1 + \delta^2 + \delta^2\bar{u}w_1 - 2\delta^2 w_1 - 2\delta} \end{aligned}$$

$$\begin{aligned} & \frac{2 + 2\delta^2 w_1 + \delta^2\bar{u}w_1 + 2\delta w_1 - 2\delta^2\bar{u} - 2\delta}{1 + \delta^2 + \delta^2\bar{u}w_1 - 2\delta^2 w_1 - 2\delta} && \textit{adding the terms} \\ & + \frac{(\delta^2) - (\delta^2) + (-2\delta^2 w_1) - (-2\delta^2 w_1)}{1 + \delta^2 + \delta^2\bar{u}w_1 - 2\delta^2 w_1 - 2\delta} && (\delta^2) - (\delta^2) + (-2\delta^2 w_1) - (-2\delta^2 w_1) \end{aligned}$$

$$\begin{aligned} & 1 + \frac{1 + 4\delta^2 w_1 + 2\delta\bar{u} - 2\delta^2\bar{u} - \delta^2}{1 + \delta^2 + \delta^2\bar{u}w_1 - 2\delta^2 w_1 - \delta^2} \\ & 1 + 1 && \delta = 0, \bar{u} = 4, w_1 = 3 \end{aligned}$$

By setting $\delta = 0$, we have that L-DA is at least 2-competitive. We find a stronger lower bound by filling in the values $\bar{u}$ and w_1 and minimise the new term

$$\frac{1 + 4\delta^2 w_1 + 2\delta\bar{u} - 2\delta^2\bar{u} - \delta^2}{1 + \delta^2 + \delta^2\bar{u}w_1 - 2\delta^2 w_1 - \delta^2} \xrightarrow[w_1=3]{\bar{u}=4} \frac{\delta(3\delta+8)+1}{6\delta^2+1}$$

From Figure 8, we can observe that the largest increase in the competitive ratio (≈ 2.4) is when $\delta \approx 0.35$. Hence, L-DA is at least 3.4-competitive.

7.5 Analysis of α and $\bar{u}$

In this subsection, we investigate the relationship between the parameters α and $\bar{u}$ in the context of the two-weight case. Specifically, we show that the adversary will never construct

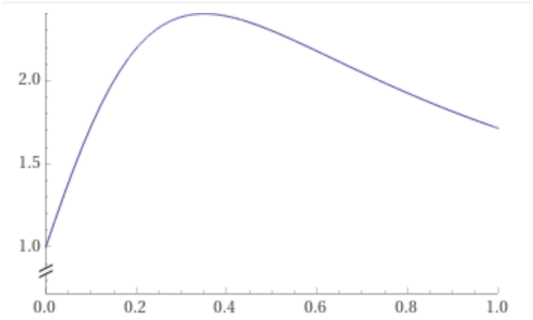

Figure 8 The function $f(\delta) = \frac{\delta(3\delta+8)+1}{6\delta^2+1}$ over values of δ, where δ is the number of heavy jobs

.

an instance with $\alpha \geq \bar{u}$. Additionally, we present a novel algorithm called U-DA and establish its competitiveness as 3 when $\alpha \geq \bar{u}$, otherwise a competitive ratio of $3 + \frac{1}{2} \times \alpha$.

▶ **Proposition 2.** *Given any two-weight instance I with $l = 1$ and $\alpha > \bar{u}$, the optimal solution always schedules the α-weighted jobs first, where $\alpha = w_l$.*

Proof. Assume on the contrary, that the optimal solution executes the unit jobs first. Observe that the weighted processing time for the unit jobs j is at least $\frac{p_j^*}{1} = \frac{1}{1}$. While for the α-weighted jobs k, the weighted ratio is at most $\frac{p_k^*}{\alpha} = \frac{\bar{u}}{\alpha} < \frac{\bar{u}}{\bar{u}}$. By the $WSPT$ rule, the optimal solution would not execute the unit jobs first. Hence, a contradiction. ■

Note that if $\alpha = \bar{u}$, then the weighted processing time of the α-weighted jobs is at most 1. Furthermore, if all jobs j have the same weighted processing time of $\frac{1}{1} = \frac{\bar{u}}{\alpha}$, then by the WSPT rule, the order of their execution is irrelevant. Therefore, when $\alpha = \bar{u}$, we assume that the optimal solution follows the execution order of L-DA. For different values of $\frac{\bar{u}}{\alpha}$ we observe which algorithm (L-DA or DA) results in a better competitive ratio, we refer to Figure 9. When $\frac{\bar{u}}{\alpha} \geq 1$, the competitive ratio becomes constant-competitive.

Algorithm Unified-Delay-All, denoted as U-DA: If $\alpha < \bar{u}$: use DA. Otherwise, use L-DA.

▶ **Theorem 26.** *Algorithm U-DA is 3-competitive, when $\alpha \geq \bar{u}$.*

Proof. By Proposition 2, we can deduce that in the optimal schedule, the execution of any unit job must be delayed by the execution of all α-weighted jobs. Hence, we can infer that the optimal schedule incurs an additional cost of $(n-k) \times \sum_{j=1}^{k} p_j^*$, in addition to the cost of executing the jobs. Combining this with Lemma 6, we can establish the following inequality:

$$\sum_{j=1}^{k} \alpha \times \sum_{i=1}^{j} p_i + \sum_{j=1}^{n-k} \sum_{i=1}^{j} p_i + (n-k) \times \sum_{j=1}^{k} p_j \leq OPT$$

The testing delay of the α-weighted and unit jobs is by Lemma 12 at most $2 \times OPT$. Hence, we have the desired competitive ratio of 3. ■

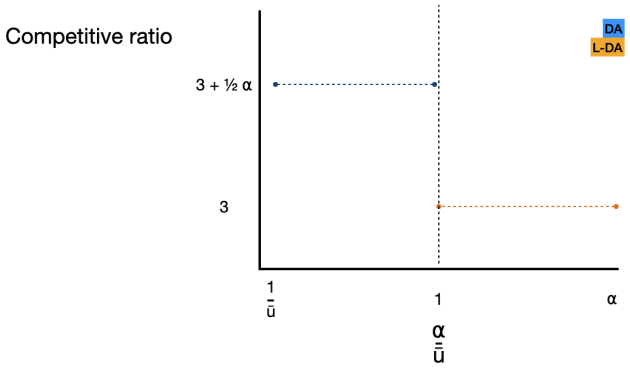

Figure 9 The competitive ratio of the two algorithms depends on the ratio $\frac{\alpha}{\bar{u}}$, where the blue line represents the algorithm DA and the orange line represents the algorithm L-DA. $\bar{u}$ denotes the uniform upper limit and α the only heavy weight.

8 Postpone-L-Delay-All Algorithm

In this section, we design a new algorithm which is a variant of L-DA with an adjusted execution strategy. We prove that the new algorithm is constant-competitive for the special case with two weights.

8.1 Postpone strategy

By Theorem 21, we can observe that the largest increase in cost is the execution delay of the unit jobs, incurred by the execution of the α-weighted jobs. The execution delay is at most $nk \times \bar{u}$, see Figure 10. A possible solution is to postpone the execution of some α-weighted

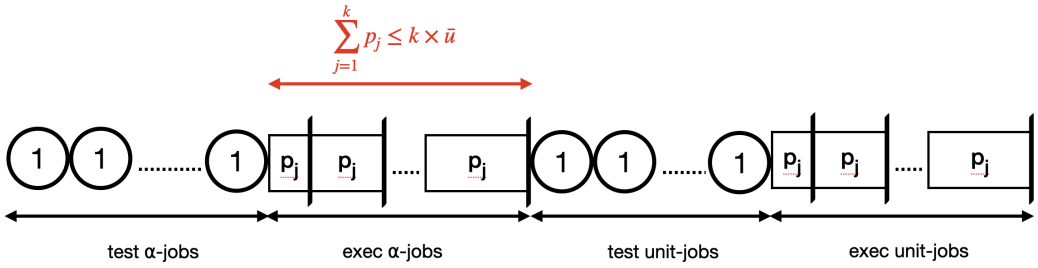

Figure 10 A representation of a schedule produced by *K-DA* where the circle represents a test, the rectangles represent the execution time and the black bar represents the completion of a job. The execution of α-weighted jobs will push the unit jobs by at most $k \times \bar{u}$.

jobs such that the time spend on executing these jobs will be reduced.

Algorithm *Postpone-L-DA*, also denoted as PL-DA, follows mostly the strategy of L-DA, except after the test phase of α-weighted jobs it has only a budget of m units of time

to executing α-weighted jobs. The cost is defined as follows

$$\begin{aligned} \textit{PL-DA} = &\sum_{j=1}^{x} \alpha \times (\sum_{i=1}^{j} p_i) + \sum_{j=1}^{n-x} w_j \times (\sum_{i=1}^{j} p_i) && \textit{execution (PL-DA}_E\textit{)} \\ &+ n(n-k) + k(x) \times \alpha + n(n_p) \times \alpha && \textit{test delay (PL-DA}_{TD}\textit{)} \\ &+ m(n-k) + m(n_p) \times \alpha && \textit{execution delay (PL-DA}_{ED}\textit{)} \end{aligned} \tag{10}$$

where x is the number of executed α-weighted jobs that are not postponed and n_p is the number of postponed α-weighted jobs. Since all jobs have a uniform upper limit, x is at least $\frac{m}{\bar{u}}$ and n_p is at most $k - \frac{m}{\bar{u}}$. The critical part is defining what m should be. If m is too large, then we end up in the same situation as L-DA. When m is too small, then the execution delay of the postponed jobs will increase by α for every unit of time.

▶ **Lemma 27.** *Given any two-weight instance I with n jobs,*

$$\text{PL-DA}_E + \text{PL-DA}_{TD} \leq 3 \times OPT + n(n_p) \times \alpha$$

where $\alpha = w^{(1)}$.

Proof. By the framework Schedule separation, we split the schedule such that we have Equation (10). Consider only the execution (PL-DA$_E$) and the testing delay (PL-DA$_{TD}$). Observe that the number of executed jobs α-weighted jobs, that are not postponed, is at most k. Then by Lemma 6 and Lemma 12, we have the the following

$$\begin{aligned} \textit{PL-DA}_E + \textit{PL-DA}_{TD} = &\sum_{j=1}^{x} \alpha \times (\sum_{i=1}^{j} p_i) + \sum_{j=1}^{n-x} w_j \times (\sum_{i=1}^{j} p_i) \\ &+ n(n-k) + k(x) \times \alpha + n(n_p) \times \alpha \\ \leq &OPT + n(n-k) + k(x) \times \alpha + n(n_p) \times \alpha && \textit{Lemma 6} \\ \leq &OPT + 2 \times OPT + n(n_p) \times \alpha && x \leq k \textit{ and Lemma 12} \end{aligned}$$

Hence, we have the desired competitive ratio of $3 + n(n_p) \times \alpha$. ■

▶ **Lemma 28.** *Given any two-weight instances I,*

$$\text{PL-DA}_{ED} \leq 0 \times OPT - n(n_p) \times \alpha$$

where n_p is the number of postponed jobs and α the only heavy weight.

Proof. We prove Lemma 28 by setting m as follows

$$m = n(1 + \frac{\bar{u}}{\alpha} - \frac{1}{\alpha}) \qquad\qquad k \geq \frac{n}{\bar{u}}$$

First, observe the following

$$\begin{aligned} \textit{PL-DA}_{ED} &\leq 0 \times OPT - n(n_p) \times \alpha \\ \textit{PL-DA}_{ED} + n(n_p) \times \alpha &\leq 0 \times OPT \\ \textit{PL-DA}_{ED} + n(n_p) \times \alpha &= n(n_p) \times \alpha + m(n-k) + m(n_p) \times \alpha \\ &\leq m(n - k + k \times \alpha) - \frac{\alpha}{\bar{u}} \times m^2 + nk \times \alpha - \frac{\alpha}{\bar{u}} \times mn && n_p \leq k - \frac{m}{\bar{u}} \\ &\leq mk(\bar{u} + \alpha - 1) + k^2 \times \alpha \times \bar{u} - \frac{\alpha}{\bar{u}} \times m^2 - \frac{\alpha}{\bar{u}} \times mn && n \leq k \times \bar{u} \end{aligned}$$

Now, we show that $m \geq n + n(\frac{\bar{u}}{\alpha} - \frac{1}{\alpha})$

$$\begin{aligned}
\frac{\alpha}{\bar{u}} \times m^2 &\geq mk(\alpha + \bar{u} - 1) \\
\frac{\alpha}{\bar{u}} \times m &\geq k(\alpha + \bar{u} - 1) \\
m &\geq \frac{\bar{u}}{\alpha} \times k(\alpha + \bar{u} - 1) \\
m &\geq \frac{\bar{u}}{\alpha} \times (\frac{n}{\bar{u}})(\alpha + \bar{u} - 1) && k \geq \frac{n}{\bar{u}} \\
m &\geq n(1 + \frac{\bar{u}}{\alpha} - \frac{1}{\alpha})
\end{aligned}$$

By setting m to $n(1 + \frac{\bar{u}}{\alpha} - \frac{1}{\alpha})$, we have that $mk(\alpha + \bar{u} - 1) - \frac{\alpha}{\bar{u}} \times m^2 = 0$. Now, for the remaining terms we have

$$\begin{aligned}
& k^2 \times \alpha \times \bar{u} - \frac{\alpha}{\bar{u}} \times n(n + n(\frac{\bar{u}}{\alpha} - \frac{1}{\alpha})) \\
&= k^2 \times \alpha \times \bar{u} + \frac{1}{\bar{u}} \times n^2 - n^2 - \frac{\alpha}{\bar{u}} \times n^2 \\
&\leq k^2 \times \alpha \times \bar{u} + k^2 \times \bar{u} - k^2 \times \alpha \times \bar{u} - k^2 \times \bar{u}^2 && n \leq k \times \bar{u} \\
&\leq k^2 \times \bar{u} - k^2 \times \bar{u}^2 \leq 0
\end{aligned}$$

If $k \geq \frac{n}{\bar{u}}$ and $\sum_{j=1}^{k} p_j = m \geq n(1 + \frac{\bar{u}}{\alpha} - \frac{1}{\alpha})$ holds, then $\textit{PL-DA}_{ED} + n(n_p) \times \alpha \leq 0 \times OPT$. ■

▶ **Theorem 29.** *Algorithm PL-DA is 3-competitive, where* $k \geq \frac{n}{\bar{u}}$ *and* $\sum_{j=1}^{k} p_j \geq n(1 + \frac{\bar{u}}{\alpha} - \frac{1}{\alpha})$.

Proof. The total schedule cost of PL-DA and the schedule separation is given by Equation (10). Observe by Lemma 27 and Lemma 28, we have

$$\begin{aligned}
\textit{PL-DA} &\leq 3 \times OPT + \textit{PL-DA}_{ED} + n(n_p) \times \alpha && \textit{Lemma 27} \\
&\leq 3 \times OPT && \textit{Lemma 28}
\end{aligned}$$

Hence, we have the desired competitive ratio of 3. ■

9 Further Observations

We observed that certain types of instances are more favourable to specific algorithms, such as those outlined in Theorem 26 for L-DA and Theorem 29 for PL-DA. Despite both algorithms being 3-competitive, their analyses differ from each other. Examining the performance of these algorithms on various types of instances can provide new insights. In this subsection, we present several different instance types for the case with two weights.

9.1 Heavy instances

Using the Schedule Separation framework, we observe that the terms related to heavy jobs (α-weighted jobs) contribute the most to the total cost. As a result, we perform a case distinction on the number of heavy jobs.

Case $k \leq \sqrt{n}$. If we assume that $\bar{u} \leq \frac{\sqrt{n}}{2}$. Then we conclude that L-DA$_{ED}$ is at most OPT, making L-DA 4-competitive:

$$\begin{aligned} L\text{-}DA_{ED} &= (n-k) \times \sum_{j=1}^{k} p_j \\ &\leq (n-k)k \times \bar{u} \\ &\leq n(\sqrt{n}) \times \frac{\sqrt{n}}{2} \\ &\leq OPT \qquad\qquad \textit{Corollary } 20 \end{aligned}$$

Case $k \leq \frac{n}{\bar{u}}$. Assume that $k > \sqrt{(n)}$ and the parameters (α and $\bar{u}$) are constant relative to the number of jobs (n and k), we consider the case where $k \leq \frac{n}{\bar{u}}$ and $\bar{u}$ can be any value. Then, L-DA$_{ED}$ is at most $2 \times OPT$, making L-DA 5-competitive:

$$\begin{aligned} L\text{-}DA_{ED} &= (n-k) \times \sum_{j=1}^{k} p_j \\ &\leq (n-k)k \times \bar{u} \\ &\leq n(\frac{k}{\bar{u}}) \times \bar{u} \\ &\leq 2 \times OPT \qquad\qquad \textit{Corollary } 20 \end{aligned}$$

Case $k \geq \frac{n}{\bar{u}}$. If $k \geq \frac{n}{\bar{u}}$ and the makespan on executing the tested heavy jobs ($\sum_{j=1}^{k} p_j$) is at least or equal to $n(1 + \frac{\bar{u}}{\alpha} - \frac{1}{\alpha})$, then PL-DA is 3-competitive, we refer to Theorem 29.

10 Conclusion

Dürr et al. [8] introduced the problem of scheduling with testing on a single machine, for the objective of minimising the total completion time. In this thesis, we extended the problem by considering the objective of minimising the total weighted completion time. Our main focus was on the problem variant where all jobs have a uniform upper limit $\bar{u}$, which makes the decisions of the algorithm independent of the predicted processing time. We showed that the approach used in paper[8] could not be generalised for the weighted case. Therefore, we approached the analysis from a different direction. We observed that the schedule produced by any algorithm can be divided into different parts or components, each of which is individually upper bounded by the lower bound of the optimal solution.

We presented **Algorithm DA**, where we proved a competitive ratio of $3 + \frac{1}{2} \times \alpha$ and $1 + 2 \times w_{max}$ for the special case with two weights (1 and α) $\in \mathbb{N}$ and the more generalised case with multiple weights, respectively. For the lower bound, we showed that there exists an instance where the competitive ratio is at least 2.4. Building on the findings of DA, we introduced a more sophisticated **Algorithm L-DA**. We proved a competitive ratio of $3 + \frac{1}{2} \times \bar{u}$ and $3 + \frac{5}{3} \times \bar{u}$ for the special case with only two weights and the multiple weight case, respectively. By combining the two algorithms, we showed that for the case with two weights, **Algorithm U-DA** is at most 3-competitive when $\alpha \geq \bar{u}$; otherwise, it is $3 + \frac{1}{2} \times \alpha$. For the lower bound, we showed for DA there exists where exists an instance where the competitive ratio is at least 2.4. Similarly, we showed for L-DA a competitive ratio of at least 3.4. The most significant result is **Algorithm PL-DA**, which is mostly based on L-DA, but has an adjusted execution strategy. We observed that when m is large enough, we achieve a

competitive ratio of 3 for the case with two weights.

Smaller m. The results on PL-DA open many questions. Currently, PL-DA is 3-competitive if: $\sum_{j=1}^{k} p_j = m \geq n(1+\frac{\bar{u}}{\alpha}-\frac{1}{\alpha})$. It would be interesting to investigate whether the algorithm is still constant-competitive when m is smaller, or whether a different competitive ratio emerges. If a different ratio does emerge, it would be valuable to understand the factors that influence it. Another interesting question is whether m should be static or dynamic for the case with multiple weights.

Test strategy. In the thesis, all the algorithms have a static testing strategy. Namely, we always test a non-trivial job. This is based on the fact that the rule *smaller upper limit* leads to a bad solution. Intuitively, when the upper limit of a job is enormous it is always better to test since the testing time (uniformly equal to 1) is compared to the upper limit relatively small. Developing a rule that takes into account multiple conditions rather than just one is a promising direction for future work.

Analysis approach. Another questionable aspect is the analysis approach. Using the lower bound of the optimal cost to upper bound the cost of any algorithm leads to an overestimation of the competitive ratio. Furthermore, in many inequality proofs, we did not fully utilise the weights. Especially in the multiple-weight case where Greedy is performing better than L-DA. The approach *cross-examining* used in paper[2] is an alternative to the framework schedule separation.

Density ratio. Another possible generalisation of the problem is to relax the restriction on the upper limit. We considered the special case where we have two upper limits ($\bar{u}$ and $\bar{u} \times d$) and two weights (β and $\beta \times d$) where $\beta \in \mathbb{N}$ and $d \in \mathbb{N}$. The weighted processing time $\frac{u_j}{w_j}$ of a job j can be any combination between the values of the upper limit and the weights. Therefore, we have four possible combinations where two have the same density ratio. We observed the performance of L-DA on such an instance and noticed that upper bounding the different parts tends to be difficult. Although, we observed that L-DA is at most $(3+2\times\bar{u}+\bar{u}\times d)$-competitive but the observation is still too pessimistic.

Extensions. Further interesting directions for future work are the extensions of the problem. For example, non-uniform testing times, multiple machines or the preemptive case.

11 Acknowledgements

We thank Dr. H.H. Liu for her detailed feedback on the earlier draft of this thesis and for the constructive discussion regarding scheduling and online algorithms.

References

1 Susanne Albers. Recent advances for a classical scheduling problem. In *International Colloquium on Automata, Languages, and Programming*, pages 4–14. Springer, 2013.

2 Susanne Albers and Alexander Eckl. Explorable uncertainty in scheduling with non-uniform testing times. *CoRR*, abs/2009.13316, 2020. URL: `https://arxiv.org/abs/2009.13316`, `arXiv:2009.13316`.

3 Susanne Albers and Alexander Eckl. Scheduling with testing on multiple identical parallel machines. *CoRR*, abs/2105.02052, 2021. URL: `https://arxiv.org/abs/2105.02052`, `arXiv:2105.02052`.

4 Susanne Albers and Matthias Hellwig. Semi-online scheduling revisited. *Theoretical Computer Science*, 443:1–9, 2012. URL: `https://www.sciencedirect.com/science/article/pii/S0304397512002939`, `doi:https://doi.org/10.1016/j.tcs.2012.03.031`.

5 Susanne Albers and Maximilian Janke. Online makespan minimization with budgeted uncertainty. In *Workshop on Algorithms and Data Structures*, pages 43–56. Springer, 2021.

6 Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. cambridge university press, 2005.

7 Wallace Clark. The gantt chart: A working tool of management. *New York, NY: Ronald Press*, page 150, 1922.

8 Christoph Dürr, Thomas Erlebach, Nicole Megow, and Julie Meißner. Scheduling with explorable uncertainty. *CoRR*, abs/1709.02592, 2017. URL: `http://arxiv.org/abs/1709.02592`, `arXiv:1709.02592`.

9 Debasis Dwibedy and Rakesh Mohanty. Semi-online scheduling: A survey. *Computers & Operations Research*, 139:105646, 2022.

10 Leah Epstein. A survey on makespan minimization in semi-online environments. *Journal of Scheduling*, 21(3):269–284, 2018.

11 Thomas Erlebach, Murilo Santos de Lima, Nicole Megow, and Jens Schlöter. Learning-augmented query policies for minimum spanning tree with uncertainty, 2022. URL: `https://arxiv.org/abs/2206.15201`, `doi:10.48550/ARXIV.2206.15201`.

12 Thomas Erlebach, Michael Hoffmann, et al. Query-competitive algorithms for computing with uncertainty. *Bulletin of EATCS*, 2(116), 2015.

13 H. L. Gantt. Work, wages, and profits. *Hive Pub. Co.*, 1974.

14 Michel X. Goemans, Maurice Queyranne, Andreas S. Schulz, Martin Skutella, and Yaoguang Wang. Single machine scheduling with release dates, 1999.

15 Ronald L Graham. Bounds for certain multiprocessing anomalies. *Bell system technical journal*, 45(9):1563–1581, 1966.

16 Ronald Lewis Graham, Eugene Leighton Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Annals of discrete mathematics*, volume 5, pages 287–326. Elsevier, 1979.

17 Michael Hoffmann, Thomas Erlebach, Danny Krizanc, Matús Mihal'ák, and Rajeev Raman. Computing Minimum Spanning Trees with Uncertainty. In Susanne Albers and Pascal Weil, editors, *25th International Symposium on Theoretical Aspects of Computer Science*, volume 1 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 277–288, Dagstuhl, Germany, 2008. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. URL: `http://drops.dagstuhl.de/opus/volltexte/2008/1358`, `doi:10.4230/LIPIcs.STACS.2008.1358`.

18 J. A. Hoogeveen and A. P. A. Vestjens. Optimal on-line algorithms for single-machine scheduling. In William H. Cunningham, S. Thomas McCormick, and Maurice Queyranne, editors, *Integer Programming and Combinatorial Optimization*, pages 404–414, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

19 S. Kahan. A model for data in motion. *ACM Symposium on Theory of Computing (STOC)*, 23, 1991.

20 Sanjeev Khanna and Wang-Chiew Tan. On computing functions with uncertainty. In *Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '01, page 171–182, New York, NY, USA, 2001. Association for Computing Machinery. `doi:10.1145/375551.375577`.

21 Jan Karel Lenstra and David B. Shmoys. Elements of scheduling, 2020. URL: `https://arxiv.org/abs/2001.06005`, `doi:10.48550/ARXIV.2001.06005`.

22 J.K. Lenstra, A.H.G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. In P.L. Hammer, E.L. Johnson, B.H. Korte, and G.L. Nemhauser, editors, *Studies*

in Integer Programming, volume 1 of *Annals of Discrete Mathematics*, pages 343–362. Elsevier, 1977. URL: https://www.sciencedirect.com/science/article/pii/S016750600870743X, doi:https://doi.org/10.1016/S0167-5060(08)70743-X.

23 Zhi-Zhong Chen Minyang Gong and Kuniteru Hayashi. Approximation algorithms for multiprocessor scheduling with testing to minimize the total job completion time. 2022.

24 Cynthia A. Phillips, Clifford Stein, and Joel Wein. Minimizing average completion time in the presence of release dates. *Mathematical Programming*, 82:199–223, 1998.

25 Michael L Pinedo. *Scheduling*, volume 29. Springer, 2012.

26 C. N. Potts and V. A. Strusevich. Fifty years of scheduling: A survey of milestones. *The Journal of the Operational Research Society*, 60:s41–s68, 2009. URL: http://www.jstor.org/stable/40206725.

27 Wayne E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2):59–66, 1956. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.3800030106, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/nav.3800030106, doi:https://doi.org/10.1002/nav.3800030106.

28 Zhankun Sun, Nilay Tanık Argon, and Serhan Ziya. When to Triage in Service Systems with Hidden Customer Class Identities? *Production and Operations Management*, 31(1):172–193, January 2022. URL: https://ideas.repec.org/a/bla/popmgt/v31y2022i1p172-193.html, doi:10.1111/poms.13494.

29 M. Weitzman. Optimal search for the best alternative. *Econometrica*, 3(47), 1979.