The Power of Amortization on Scheduling with Explorable Uncertainty

Alison Hsiang-Hsuan Liu[1][0000−0002−0194−9360], Fu-Hong Liu[0000−0001−6073−8179], Prudence W.H. Wong[2][0000−0001−7935−7245]⋆, and Xiao-Ou Zhang[1]

[1] Utrecht University, the Netherlands
[2] University of Liverpool, the United Kingdom

Abstract. In this work, we study a scheduling problem with explorable uncertainty. Each job comes with an upper limit of its processing time, which could be potentially reduced by testing the job, which also takes time. The objective is to schedule all jobs on a single machine with a minimum total completion time. The challenge lies in deciding which jobs to test and the order of testing/processing jobs.
The online problem was first introduced with unit testing time [5,6] and later generalized to variable testing times [1]. For this general setting, the upper bounds of the competitive ratio are shown to be 4 and 3.3794 for deterministic and randomized online algorithms [1]; while the lower bounds for unit testing time stands [5,6], which are 1.8546 (deterministic) and 1.6257 (randomized).
We continue the study on variable testing times setting. We first enhance the analysis framework in [1] and improve the competitive ratio of the deterministic algorithm in [1] from 4 to $1 + \sqrt{2} \approx 2.4143$. Using the new analysis framework, we propose a new deterministic algorithm that further improves the competitive ratio to 2.316513. The new framework also enables us to develop a randomized algorithm improving the expected competitive ratio from 3.3794 to 2.152271.

Keywords: Explorable uncertainty, Online scheduling algorithms, Total completion time, Competitive analysis, Amortized analysis

1 Introduction

In this work, we study the single-machine *Scheduling with Uncertain Processing time* (SUP) problem with the minimized total completion time objective. We are given n jobs, where each job has a *testing time* t_j and an *upper limit* u_j of its *real processing time* $p_j \in [0, u_j]$. A job j can be executed (without testing), taking u_j time units. A job j can also be tested using t_j time units, and after it is tested, it takes p_j time to execute. Note that any algorithm needs to test a job j beforehand to run it in time p_j. The online algorithm does not know the exact value of p_j unless it tests the job. On the other hand, the optimal offline

⋆ The work is partially supported by University of Liverpool Covid Recovery Fund.

algorithm knows in advance each p_j even before testing. Therefore, the optimal strategy is to test job j if and only if $t_j + p_j \leq u_j$ and execute the shortest job first, where the processing time of a job j is $\min\{t_j + p_j, u_j\}$ [1,5,6]. However, since the online algorithm only learns about p_j after testing j, the challenge to the online algorithm is to decide which jobs to test and the order of tasks that could be testing, execution, or execution-untested.

It is typical to study uncertainty in scheduling problems, for example, in the worst case scenario for online or stochastic optimization. Kahan [15] has introduced a novel notion of explorable uncertainty where queries can be used to obtain additional information with a cost. The model of scheduling with explorable uncertainty studied in this paper was introduced by Dürr et al. recently [5,6]. In this model, job processing times are uncertain in the sense that only an upper limit of the processing time is known, and can be reduced potentially by testing the job, which takes a testing time that may vary according to the job. An online algorithm does not know the real processing time before testing the job, whereas an optimal offline algorithm has the full knowledge of the uncertain data.

One of the motivations to study scheduling with uncertain processing time is clinic scheduling [3, 16]. Without a pre-diagnosis, it is safer to assign each treatment the maximum time it may need. With pre-diagnosis, the precise time a patient needs can be identified, which can improve the performance of the scheduling. Other applications are, as mentioned in [5,6], code optimization [2], compression for file transmission over network [20], fault diagnosis in maintenance environments [17]. Application in distributed databases with centralized master server [18] is also discussed in [1].

In addition to its practical motivations, the model of explorable uncertainty also blurs the line between offline and online problems by allowing a restricted uncertain input. It enables us to investigate how uncertainty influences online decision quality in a more quantitative way. The concept of exploring uncertainty has raised a lot of attention and has been studied on different problems, such as sorting [13], finding the median [11], identifying a set with the minimum-weight among a given collection of feasible sets [8], finding shortest paths [10], computing minimum spanning trees [14], etc. More recent work and a survey can be found in [7,10,12]. Note that in many of the works, the aim of the algorithm is to find the optimal solution with the minimum number of testings for the uncertain input, comparing against the optimal number of testings.

Another closely related model is Pandora's box problem [4,9,19], which was based on the secretary problem, that was first proposed by Weitzman [19]. In this problem, each candidate (that is, the box) has an independent probability distribution for the reward value. To know the exact reward a candidate can provide, one can open the box and learn its realized reward. More specifically, at any time, an algorithm can either open a box, or select a candidate and terminate the game. However, opening a box costs a price. The goal of the algorithm is to maximize the reward from the selected candidate minus the total cost of opening boxes. The Pandora's box problem is a foundational framework for studying how the cost of revealing uncertainty affects the decision quality.

More importantly, it suggests what information to acquire next after gaining some pieces of information.

Previous works. For the SUP problem, Dürr et al. studied the case where all jobs have the same testing time [5, 6]. In the paper, the authors proposed a THRESHOLD algorithm for the special instances. For the competitive analysis, the authors proposed a delicate *instance-reduction* framework. Using this framework, the authors showed that the worst case instance of THRESHOLD has a special format. An upper bound of the competitive ratio of 2 of THRESHOLD is obtained by the ratio of the special format instance. Using the instance-reduction framework, the authors also showed that when all jobs have the same testing time and the same upper limit, there exists a 1.9338-competitive BEAT algorithm. The authors provided a lower bound of 1.8546 for any deterministic online algorithm. For randomized algorithms, the authors showed that the expected competitive ratio is between 1.6257 and 1.7453.

Later, Albers and Eckl studied a more general case where jobs have variable testing time [1]. In the paper, the authors proposed a classic and elegant framework where the completion time of an algorithm is divided into contribution segments by the jobs executed prior to it. For the jobs with "correct" execution order as they are in the optimal solution, their total contribution to the total completion time is charged to twice the optimal cost by the fact that the algorithm does not pay too much for wrong decisions of testing a job or not. For the jobs with "wrong" execution order, their total contribution to the total completion time is charged to another twice the optimal cost using a *comparison tree* method, which is bound with the proposed (α, β)-SORT algorithm. The authors also provide a preemptive 3.2361-competitive algorithm and an expected 3.3794-competitive randomized algorithm.

In the works [1, 5, 6], the objective of minimizing the maximum completion time on a single machine was also studied. For the uniform-testing-time setting, Dürr et al. [5, 6] proposed a ϕ-competitive deterministic algorithm and a $\frac{4}{3}$-competitive randomized algorithm, where both algorithms are optimal. For a more general setting, Albers and Eckl [1] showed that variable testing time does not increase the competitive ratios of online algorithms.

Our contribution. We first analyze the (α, β)-SORT algorithm proposed in the work [1] in a more amortized sense. Instead of charging the jobs in the correct order and in the wrong order to the optimal cost separately, we manage to partition the tasks into groups and charge the total cost in each of the groups to the optimal cost regarding the group. The introduction of amortization to the analysis creates room for improving the competitive ratio by adjusting the values of α and β. The possibility of picking $\alpha > 1$ helps balance the penalty incurred by making a wrong guess on testing a job or not. On the other hand, the room for different β values allows one to differently prioritize the tasks that provide extra information and the tasks that immediately decide a completion time for a job. By this new analysis and the room of choosing different values of α and β, we improve the upper bound of the competitive ratio of (α, β)-SORT from 4 to $1 + \sqrt{2}$. With the power of amortization, we improve the algorithm by

	Testing time	Upper limit	Upper Bound	Lower bound
Deterministic	1	Uniform	1.9338 [5,6]	1.8546 [5,6]
		Variable	2 [5,6]	
	Variable	Variable	4 [1] → **2.414** (Theorem 1) **2.316513** (Theorem 2)	
		(Prmp.)	3.2361 [1] **2.316513** (Theorem 2)	
Randomized	1	Variable	1.7453 [5,6]	1.6257 [5,6]
	Variable	Variable	3.3794 [1] **2.152271** (Theorem 3)	

Table 1: Summary of the results. The results from this work are bold and in red.

further prioritizing different tasks using different parameters. The new algorithm, $\text{PCP}_{\alpha,\beta}$, is 2.316513-competitive. This algorithm is extended to a randomized version with an expected competitive ratio of 2.152271. Finally, we show that under the current problem setting, preempting the execution of jobs does not help in gaining a better algorithm. A summary of the results can be found in Table 1.

Paper organization. In Section 2, we introduce the notation used in this paper. We also review the algorithm and analysis of the (α, β)-SORT algorithm proposed in the work [1]. In Section 3, we elaborate on how amortized analysis helps to improve the competitive analysis of (α, β)-SORT (Subsection 3.1). Upon the new framework, we propose a better algorithm, $\text{PCP}_{\alpha,\beta}$, in Subsection 3.2. In Subsection 3.3, we argue that the power of preemption is limited in the current model. Finally, we show how amortization helps to improve the performance of randomized algorithms. For the sake of the page limit, we leave the proofs in the full version.

2 Preliminary

Given n jobs $1, 2, \cdots, n$, each job j has a *testing time* t_j and an *upper limit* u_j of its *real processing time* $p_j \in [0, u_j]$. A job j can be executed-untested in u_j time units or be tested using t_j time units and then executed in p_j time units. Note that if a job is tested, it does not need to be executed immediately. That is, for a tested job, there can be tasks regarding other jobs between its testing and its execution.

We denote by p_j^A the time spent by an algorithm A on job j, i.e., $p_j^A = t_j + p_j$ if A tests j, and $p_j^A = u_j$ otherwise. Similarly, we denote by p_j^* the time spent by OPT, the optimal algorithm. Since OPT knows p_j in advance, it can decide optimally whether to test a job, i.e., $p_j^* = \min\{u_j, t_j + p_j\}$, and execute the jobs in the ascending order of p_j^*. We denote by $cost(A)$ the total completion time of any algorithm A.

The *tasks* regarding a job j are the testing, execution, or execution-untested of j (taking t_j, p_j, or u_j, respectively). We follow the notation in the work of Albers and Eckl [1] and denote $c(k, j)$ as the *contribution* of job k in the completion time of job j in the online schedule A. That is, $c(k, j)$ is the total time of the tasks regarding job k before the completion time of job j. The *completion time* of job j in the schedule A is then $\sum_{k=1}^{n} c(k, j)$. Similarly, we define $c^*(k, j)$ as the contribution of job k in the completion time of job j in the optimal schedule. As observed, OPT schedules in the order of p^*, $c^*(k, j) = 0$ if k is executed after j in the optimal schedule, and $c^*(k, j) = p_k^*$ otherwise.

We denote by $i <_o j$ if the optimal schedule executes job i before job j. We also define $i >_o j$ and $i =_o j$ similarly (in the latter case, job i and job j are the same job). The completion time of job j in the optimal schedule is denoted by $c_j^* = \sum_{i \leq_o j} p_i^*$. The total completion time of the optimal schedule is then $\sum_{j=1}^{n} c_j^*$. Note that there is an optimal strategy where $p_i^* \leq p_j^*$ if $i \leq_o j$.

2.1 Review (α, β)-SORT algorithm [1]

For completeness, we summarise the (α, β)-SORT algorithm and its analysis proposed in the work of Albers and Eckl [1].

Intuitively, the algorithm tests a job j if and only if $u_j \geq \alpha \cdot t_j$. Depending on whether a job is tested or not, the job is transformed into one task (execution-untested task) or two tasks (testing task and execution task). These tasks are then maintained in a priority queue for the algorithm to decide their processing order. More specifically, a testing task has a weight of $\beta \cdot t_j$, an execution task has a weight of p_j, and an execution-untested task has a weight of u_j. (See Algorithm 1.) After the tasks regarding the jobs are inserted into the queue, the algorithm executes the tasks in the queue and deletes the executed tasks, starting from the task with the shortest (weighted) time. If the task is a testing of a job j, the resulting p_j is inserted into the queue after testing. (See Algorithm 2.) Intuitively, both α and β are at least 1. The precise values of α and β will be decided later based on the analysis.

Analysis [1]. Recall that $c(k, j)$ is the contribution of job k of the completion time of job j, and the completion time of job j is $c_j^A = \sum_{k=1}^{n} c(k, j)$. The key

Algorithm 1 (α, β)-SORT algorithm [1]

```
Initialize a priority queue $Q$
for $j = 1, 2, 3, \cdots, n$ do
    if $u_j \geq \alpha \cdot t_j$ then
        Insert a testing task with weight $\beta \cdot t_j$ into $Q$
    else
        Insert an execution-untested task with weight $u_j$ into $Q$
Queue-Execution($Q$)                                    ▷ See Algorithm 2
```

Algorithm 2 Procedure **Queue-Execution** (Q)

```
procedure Queue-Execution($Q$)
    while $Q$ is not empty do
        $x \leftarrow$ Extract the smallest-weight task in $Q$
        if $x$ is a testing task for a job $j$ then
            Test job $j$                                        ▷ It takes $t_j$ time
            Insert an execution task with weight $p_j$ into $Q$
        else if $x$ is an execution task for a job $j$ then
            Execute (tested) job $j$                            ▷ It takes $p_j$ time
        else                            ▷ $x$ is an execution-untested task for a job $j$
            Execute job $j$ untested                            ▷ It takes $u_j$ time
```

idea of the analysis is that given job j, partitioning the jobs (say, k) that are executed before j into two groups, $k \leq_o j$ or $k >_o j$. Since the algorithm only tests a job j when $u_j \geq \alpha t_j$, $p_k^A \leq \max\{\alpha, 1 + \frac{1}{\alpha}\} \cdot p_k^*$. Therefore, the total cost incurred by the first group of jobs is at most $\max\{\alpha, 1 + \frac{1}{\alpha}\} \cdot \mathit{cost}(\mathrm{OPT})$. Note that the ratio, in this case, reflects the penalty to the algorithm that makes a wrong guess on testing a job or not.

For the second group of jobs, the authors proposed a classic and elegant *comparison tree* framework to charge each $c(k, j)$ with $k >_o j$ to the time that the optimal schedule spends on job j. More specifically, $c(k, j) \leq \max\{(1 + \frac{1}{\beta})\alpha, 1 + \frac{1}{\alpha}, 1 + \beta\} \cdot p_j^*$ for any k and j. Hence, the total cost incurred by the second group of jobs can be charged to $\max\{(1 + \frac{1}{\beta})\alpha, 1 + \frac{1}{\alpha}, 1 + \beta\} \cdot \mathit{cost}(\mathrm{OPT})$.

By summing up the $c(k, j)$ values for all pairs of k and j, the total completion time of the algorithm is at most

$$\max\{\alpha, 1 + \frac{1}{\alpha}\} + \max\{(1 + \frac{1}{\beta}) \cdot \alpha, 1 + \frac{1}{\alpha}, 1 + \beta\}.$$

When $\alpha = \beta = 1$ (which is the optimal selection), the competitive ratio is 4.

2.2 Our observation

As stated by Albers and Eckl [1], $\alpha = \beta = 1$ is the optimal choice in their analysis framework. Therefore, it is not possible to find a better α and β to tighten the competitive ratio under the current framework. However, the framework can be improved via observations.

For example, given that $\alpha = \beta = 1$, consider two jobs k and j, where $(t_k, u_k, p_k) = (1 + \varepsilon, 1 + 3\varepsilon, 1 + 3\varepsilon)$ and $(t_j, u_j, p_j) = (1, 1 + 4\varepsilon, 1 + 2\varepsilon)$. By the (α, β)-SORT algorithm, both k and j are tested. The order of the tasks regarding these two jobs is t_j, t_k, p_j, and finally p_k. On the other hand, in the optimal schedule, $p_k^* = u_k = 1 + 3\varepsilon$ and $p_j^* = u_j = 1 + 4\varepsilon$. Since $k \leq_o j$, as shown in Figure 1, both $c(k, j)$ and $c(j, k)$ are charged to $2p_k^*$, separately. Note that although $c(k, j) = t_k$ in this example, the worst-case nature of the analysis framework fails to capture the fact that the contribution from the tasks regarding k to the completion time of j is even smaller than p_k^*. This observation motivates us to establish a new analysis framework.

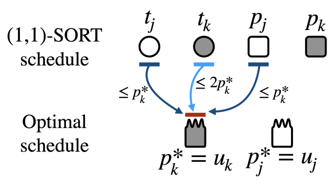


Fig. 1: An example where p_k^* is charged four times. The light blue and dark blue segments represent $c(k, j)$ and $c(j, k)$, respectively. The red segment represents p_k^*.

3 Deterministic algorithms

In this section, we first enhance the framework by equipping it with amortized analysis in Subsection 3.1. Using amortized arguments, for any two jobs $k \leq_o j$, we manage to charge the sum of $c(k, j) + c(j, k)$ to p_k^*. The new framework not only improves the competitive ratio but also creates room for adjusting α and β.

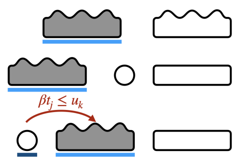
Finally, in Subsection 3.2, we improve the (α, β)-SORT algorithm based on our enhanced framework.

3.1 Amortization

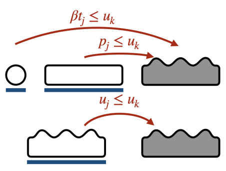
We first bound $c(k, j) + c(j, k)$ for all pairs of jobs k and j with $k \leq_o j$ by a function $r(\alpha, \beta) \cdot c^*(k, j)$. Then, we can conclude that the algorithm is $r(\alpha, \beta)$-competitive by the following argument:

$$\begin{aligned} cost((\alpha,\beta)\text{-SORT}) &= \sum_{j=1}^{n}\sum_{k=1}^{n} c(k,j) = \sum_{j=1}^{n}\Big(\sum_{k<_o j}(c(k,j)+c(j,k)) + c(j,j)\Big) \\ &\leq \sum_{j=1}^{n} r(\alpha,\beta)\cdot\Big(\sum_{k<_o j} c^*(k,j) + c^*(j,j)\Big) = r(\alpha,\beta)\cdot cost(\text{OPT}) \end{aligned}$$

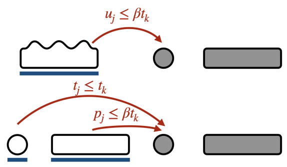
To bound $c(k, j) + c(j, k)$ by the cost of tasks k, we first observe that it is impossible that $c(k, j) = p_k^A$ and $c(j, k) = p_j^A$ at the same time. More specifically, depending on whether the jobs k and j are tested or not, the last task regarding these two jobs does not contribute to $c(k, j) + c(j, k)$. Furthermore, the order of these jobs' tasks in the priority queue provides a scheme to charge the cost of the tasks regarding j to the cost of tasks regarding k.



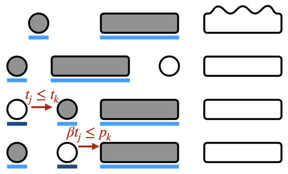
(a) k is not tested and $c(k,j) = u_k$



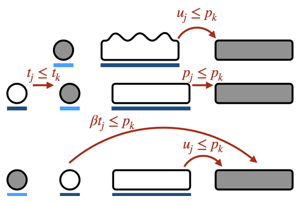
(b) k is not tested and $c(k,j) = 0$



(c) k is tested and $c(k,j) = 0$



(d) k is tested and $c(k,j) = t_k + p_k$



(e) k is tested and $c(k,j) = t_k$

Fig. 2: The red arrows illustrate how to charge $c(k,j) + c(j,k)$ to the cost of tasks regarding k. Each row in the sub-figures is a permutation of how the tasks are executed. The circles and rectangles are testing tasks and execution tasks after testing, respectively. The rectangles with curly tops are execution tasks without testing. The tasks in gray are from the job k, and the tasks in white are from the job j. The light blue and dark blue line segments under the tasks represent the contribution $c(k,j)$ and $c(j,k)$, respectively.

Figure 2 shows how the charging is done. Each row in the subfigures is a permutation of how the tasks regarding job j and k are executed. The gray objects are tasks regarding k, and the white objects are tasks regarding j. The circles, rectangles, and rectangles with the wavy top are testing tasks, execution tasks, and execution-untested tasks, respectively. The horizontal lines present the values of $c(k,j)$ (light blue) and $c(j,k)$ (dark blue). The red arrows indicate how the cost of a task regarding j is charged to that of a task regarding k according to the order of the tasks in the priority queue. The charging $c(k,j) + c(j,k)$ to the cost of tasks regarding k results in Lemmas 1 and 2. For the sake of space, the proof is provided in the full paper.

Lemma 1. *If (α,β)-SORT does not test job k,*

$$c(k,j)+c(j,k)\le(1+\frac{1}{\beta})u_k.$$

Lemma 2. *If (α,β)-SORT tests job k,*

$$c(k,j)+c(j,k)\le\max\{2t_k+p_k,(1+\beta)t_k,t_k+(1+\frac{1}{\beta})p_k\}.$$

Now, we can bound the competitive ratio of the (α,β)-SORT (Theorem 1). The idea is, depending on whether job k is tested or not by the optimal schedule, the expressions in Lemmas 1 and 2 can be written as a function of α, β, and p^*_k. By selecting the values of α and β carefully, we can balance the worst case ratio in the scenario where k is executed-untested by the algorithm (Lemma 1) and that in the scenario where k is tested by the algorithm (Lemma 2).

Theorem 1. *The competitive ratio of (α,β)-SORT is at most*

$$\max\{\alpha(1+\frac{1}{\beta}),1+\frac{1}{\alpha}+\frac{1}{\beta},1+\beta,2,1+\frac{2}{\alpha}\} \tag{1}$$

By choosing $\alpha=\beta=\sqrt{2}$, (α,β)-SORT algorithm is $(1+\sqrt{2})$-competitive. The choice is optimal for expression (1).

Note that by Theorem 1, the (α,β)-SORT algorithm is 3-competitive when $\alpha=\beta=1$, which matches the observation in Figure 1.

Our analysis framework provides room for adjusting the values of α and β. By selecting the values of α and β, we can tune the cost of tasks regarding k that is charged. By selecting a value of α other than 1, we can balance the penalty of making a wrong decision on testing a job or not. The capability of selecting a value of β other than 1 allows us to prioritize the testing tasks (which are scaled by β) and the execution tasks (which immediately decide a completion time of a job). Finally, the performance of the algorithm is tuned by finding the best values of α and β.

However, recall that the parameter α encodes the penalty for making a wrong guess on testing a job or not. When $\alpha=\sqrt{2}$, the penalty for testing a job we should not test is more expensive than that for executing-untested a job that we should test. It inspires us to improve the algorithm further.

3.2 An improved algorithm

Surprisingly, the introduction of amortization even sheds light on further improvement of the algorithm. We propose a new algorithm, *Prioritizing-Certain-Processing-time* (PCP$_{\alpha,\beta}$). The main difference between PCP$_{\alpha,\beta}$ and (α,β)-SORT is that in the PCP$_{\alpha,\beta}$ algorithm after a job j is tested, an item with weight t_j+p_j is inserted into the queue instead of p_j (see Algorithm 3). Intuitively, we prioritize a job by its certain (total) processing time p^A_j, which can be t_j+p_j or u_j.

Algorithm 3 Procedure **Updated Queue-Execution** (Q)

```
procedure UPDATED QUEUE-EXECUTION($Q$)
    while $Q$ is not empty do
        $x \leftarrow$ Extract the smallest-weight task in $Q$
        if $x$ is a testing task for a job $j$ then
            Test job $j$                                        ▷ It takes $t_j$ time
            Insert an execution task with weight $t_j + p_j$ into $Q$
        else if $x$ is an execution task for a job $j$ then
            Execute (tested) job $j$                            ▷ It takes $p_j$ time
        else                          ▷ $x$ is an execution-untested task for a job $j$
            Execute job $j$ untested                            ▷ It takes $u_j$ time
```

Then, we can charge the total cost of tasks regarding a wrong-ordered j to βt_k or p_k^A all at once.

The new algorithm PCP$_{\alpha,\beta}$ (Algorithm 1 combined with Algorithm 3) has an improved estimation of $c(k,j) + c(j,k)$ when $c(j,k) = t_j + p_j$. However, when there is only one task regarding j contributing to $c(j,k)$, the estimation of $c(k,j) + c(j,k)$ may increase. Formally, we have the following two lemmas.

Lemma 3. *Given two jobs $k \leq_o j$, if PCP$_{\alpha,\beta}$ does not test job k,*

$$c(k,j) + c(j,k) \leq (1 + \frac{1}{\beta})u_k.$$

Lemma 4. *Given two jobs $k \leq_o j$, if PCP$_{\alpha,\beta}$ tests job k,*

$$c(k,j) + c(j,k) \leq \max\{2t_k + p_k, \beta t_k, (1 + \frac{1}{\beta})(t_k + p_k)\}.$$

Similar to the proof of Theorem 1, we have the following competitiveness results of the PCP$_{\alpha,\beta}$ algorithm.

Theorem 2. *The competitive ratio of PCP$_{\alpha,\beta}$ is at most*

$$\max\{\alpha(1 + \frac{1}{\beta}), 1 + \frac{1}{\alpha} + \frac{1}{\beta} + \frac{1}{\alpha\beta}, \beta, 2, 1 + \frac{2}{\alpha}\} \ . \tag{2}$$

By choosing $\alpha = \frac{1+\sqrt{5}}{2}$ and $\beta = \frac{1+\sqrt{5}+\sqrt{2(7+5\sqrt{5})}}{4}$, the competitive ratio of PCP$_{\alpha,\beta}$ is $\frac{1+\sqrt{5}+\sqrt{2(7+5\sqrt{5})}}{4} \leq 2.316513$. The choice is optimal for expression (2).

The selection of golden ratio α balances the penalty of making a wrong guess for testing a job or not.

Note that using the analysis proposed in the work of Albers and Eckl [1] on the new algorithm that put $t_j + p_j$ back to the priority list after testing job j, the competitive ratio is $\max\{\alpha, 1 + \frac{1}{\alpha}\} + \max\{\alpha, 1 + \frac{1}{\alpha}, \beta\}$. The best choice of the values is $\alpha = \phi$ and $\beta \in [1, \phi]$, and the competitive ratio is at most 2ϕ.

Algorithm 4 Rand-PCP$_\beta$ algorithm

```
Initialize a priority queue $Q$
for $j = 1, 2, 3, \cdots, n$ do
    Let $r_j \leftarrow \frac{u_j}{t_j}$
    if $r_j < 1$ then
        $\mathbb{P}_j \leftarrow 0$
    else if $r_j > 3$ then
        $\mathbb{P}_j \leftarrow 1$
    else
        $\mathbb{P}_j = \frac{3r_j^2 - 3r_j}{3r_j^2 - 4r_j + 3}$
    Choose one of $\beta t_j$ and $u_j$ randomly with probability $\mathbb{P}_j$ for $\beta t_j$ and $1 - \mathbb{P}_j$ for $u_j$
    Insert a testing task with weight $\beta t_j$ into $Q$ if $\beta t_j$ is chosen, and insert an execution-untested task with weight $u_j$ into $Q$ otherwise
Updated Queue-Execution($Q$)                                    ▷ See Algorithm 3
```

3.3 Preemption

We show that preempting the tasks does not improve the competitive ratio. Intuitively, we show that given an algorithm A that generates a preemptive schedule, we can find another algorithm B that is capable of simulating A and performs the necessary merging of preempted parts. The simulation may make the timing of A's schedule gain extra information about the real processing times earlier due to the advance of a testing task. However, a non-trivial A can only perform better by receiving the information earlier. Thus, B's non-preemptive schedule has a total completion time at most that of A's schedule.

Lemma 5. *In the SUP problem, if there is an algorithm that generates a preemptive schedule, then we can always find another algorithm that generates a non-preemptive schedule and performs as well as the previous algorithm in terms of competitive ratios.*

4 Randomized algorithm

The amortization also helps improve the performance of randomized algorithms. We combine the PCP$_{\alpha,\beta}$ algorithm with the framework in the work of Albers and Eckl [1], where instead of using a fixed threshold α, a job j is tested with probability $\mathbb{P}_j$, which is a function of u_j, t_j, and β.

Our randomized algorithm. For any job j with $\frac{u_j}{t_j} < 1$ or $\frac{u_j}{t_j} > 3$, we insert u_j or βt_j into the queue, respectively. For any job j with $1 \le \frac{u_j}{t_j} \le 3$, we insert βt_j into the queue with probability $\mathbb{P}_j$ and insert u_j with probability $1 - \mathbb{P}_j$. Once a testing task t_j is executed, we insert $t_j + p_j$ into the queue. (See Algorithms 4 and 3.)

Analysis. The following lemma can be proven using Lemma 3 and Lemma 4.

Lemma 6. *The expected total completion time of the n jobs is at most*

$$\sum_j \sum_{k \leq_o j} (1+\frac{1}{\beta})u_k(1-\mathbb{P}_k) + \max\{2t_k+p_k, \beta t_k, (1+\frac{1}{\beta})(t_k+p_k)\}\mathbb{P}_k,$$

where $\mathbb{P}_k$ is the probability that job k is tested.

Depending on whether the jobs are tested or not in the optimal schedule, the expected total completion time can be expressed by functions with the variables: the probability, the parameters of the jobs, and β. We design the probability $\mathbb{P}_k$ by balancing the costs between the worst cases where $p_k^* = u_k$ or $p_k^* = t_k + p_k$. Note that there are cases where the "ideal" value of $\mathbb{P}_k$ is outside the range $[0, 1]$. We take care of these special cases by setting $\mathbb{P}_k$ as 0 or 1 if the ideal value is smaller than 0 or larger than 1, respectively.

Theorem 3. *Let r_k denote $\frac{u_k}{t_k}$. The expected competitive ratio of Rand-PCP$_\beta$ is at most*

$$\max_k \frac{(1+\frac{1}{\beta})u_k(1-\mathbb{P}_k) + \max\{2t_k+p_k, \beta t_k, (1+\frac{1}{\beta})(t_k+p_k)\}\mathbb{P}_k}{p_k^*}, \text{ where}$$

$$\mathbb{P}_k = \frac{(\beta+1)(r_k-1)}{\beta(\max\{\frac{2}{r_k}+1, \frac{\beta}{r_k}, (1+\frac{1}{\beta})(1+\frac{1}{r_k})\} - \max\{2, \beta, 1+\frac{1}{\beta}\} + r_k - 1) + r_k - 1}$$

if $r_k \in [1, 3]$, $\mathbb{P}_k = 0$ if $r_k < 1$, and $\mathbb{P}_k = 1$ if $r_k > 3$. By choosing $\beta = 2$, the ratio is $\frac{3(7+3\sqrt{6})}{20} \leq 2.152271$. The choice of β is optimal.

5 Conclusion

In this work, we study a scheduling problem with explorable uncertainty. We enhance the analysis framework proposed in the work [1] by introducing amortized perspectives. Using the enhanced analysis framework, we are able to balance the penalty incurred by different wrong decisions of the online algorithm. In the end, we improve the competitive ratio significantly from 4 to 2.316513 (deterministic) and from 3.3794 to 2.152271 (randomized). An immediate open problem is if one can further improve the competitive ratio by a deeper level of amortization.

Additionally, we show that preemption does not improve the competitive ratio in the current problem setting, where all jobs are available at first. It may not be true in the fully online setting, where jobs can arrive at any time. Thus, another open problem is to study the problem in the fully online model.

References

1. Albers, S., Eckl, A.: Explorable uncertainty in scheduling with non-uniform testing times. In: Kaklamanis, C., Levin, A. (eds.) Approximation and Online Algorithms - 18th International Workshop, WAOA 2020, Virtual Event, September 9-10, 2020, Revised Selected Papers. Lecture Notes in Computer Science, vol. 12806, pp. 127–142. Springer (2020), https://doi.org/10.1007/978-3-030-80879-2_9

2. Cardoso, J.M.P., Diniz, P.C., Coutinho, J.G.F.: Embedded computing for high performance: Efficient mapping of computations using customization, code transformations and compilation. Morgan Kaufmann Publishers (2017)
3. Caruso, S., Galatà, G., Maratea, M., Mochi, M., Porro, I.: Scheduling pre-operative assessment clinic via answer set programming. In: Benedictis, R.D., Maratea, M., Micheli, A., Scala, E., Serina, I., Vallati, M., Umbrico, A. (eds.) Proceedings of the 9th Italian workshop on Planning and Scheduling (IPS'21) and the 28th International Workshop on "Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion" (RCRA'21) with CEUR-WS co-located with 20th International Conference of the Italian Association for Artificial Intelligence (AIxIA 2021), Milan, Italy (virtual), November 29th-30th, 2021. CEUR Workshop Proceedings, vol. 3065. CEUR-WS.org (2021), https://ceur-ws.org/Vol-3065/paper3_196.pdf
4. Ding, B., Feng, Y., Ho, C., Tang, W., Xu, H.: Competitive information design for pandora's box. In: Bansal, N., Nagarajan, V. (eds.) Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023. pp. 353–381. SIAM (2023), https://doi.org/10.1137/1.9781611977554.ch15
5. Dürr, C., Erlebach, T., Megow, N., Meißner, J.: Scheduling with explorable uncertainty. In: Karlin, A.R. (ed.) 9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA. LIPIcs, vol. 94, pp. 30:1–30:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2018), https://doi.org/10.4230/LIPIcs.ITCS.2018.30
6. Dürr, C., Erlebach, T., Megow, N., Meißner, J.: An adversarial model for scheduling with testing. Algorithmica **82**(12), 3630–3675 (2020), https://doi.org/10.1007/s00453-020-00742-2
7. Erlebach, T., Hoffmann, M.: Query-competitive algorithms for computing with uncertainty. Bull. EATCS **116** (2015), http://eatcs.org/beatcs/index.php/beatcs/article/view/335
8. Erlebach, T., Hoffmann, M., Kammer, F.: Query-competitive algorithms for cheapest set problems under uncertainty. Theor. Comput. Sci. **613**, 51–64 (2016), https://doi.org/10.1016/j.tcs.2015.11.025
9. Esfandiari, H., Hajiaghayi, M.T., Lucier, B., Mitzenmacher, M.: Online pandora's boxes and bandits. In: The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019. pp. 1885–1892. AAAI Press (2019), https://doi.org/10.1609/aaai.v33i01.33011885
10. Feder, T., Motwani, R., O'Callaghan, L., Olston, C., Panigrahy, R.: Computing shortest paths with uncertainty. J. Algorithms **62**(1), 1–18 (2007), https://doi.org/10.1016/j.jalgor.2004.07.005
11. Feder, T., Motwani, R., Panigrahy, R., Olston, C., Widom, J.: Computing the median with uncertainty. In: Yao, F.F., Luks, E.M. (eds.) Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA. pp. 602–607. ACM (2000), https://doi.org/10.1145/335305.335386
12. Gupta, M., Sabharwal, Y., Sen, S.: The update complexity of selection and related problems. Theory Comput. Syst. **59**(1), 112–132 (2016), https://doi.org/10.1007/s00224-015-9664-y

13. Halldórsson, M.M., de Lima, M.S.: Query-competitive sorting with uncertainty. Theor. Comput. Sci. **867**, 50–67 (2021), https://doi.org/10.1016/j.tcs.2021.03.021
14. Hoffmann, M., Erlebach, T., Krizanc, D., Mihalák, M., Raman, R.: Computing minimum spanning trees with uncertainty. In: Albers, S., Weil, P. (eds.) STACS 2008, 25th Annual Symposium on Theoretical Aspects of Computer Science, Bordeaux, France, February 21-23, 2008, Proceedings. LIPIcs, vol. 1, pp. 277–288. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany (2008), https://doi.org/10.4230/LIPIcs.STACS.2008.1358
15. Kahan, S.: A model for data in motion. In: Koutsougeras, C., Vitter, J.S. (eds.) Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5-8, 1991, New Orleans, Louisiana, USA. pp. 267–277. ACM (1991), https://doi.org/10.1145/103418.103449
16. Lopes, J., Vieira, G., Veloso, R., Ferreira, S., Salazar, M., Santos, M.F.: Optimization of surgery scheduling problems based on prescriptive analytics. In: Gusikhin, O., Hammoudi, S., Cuzzocrea, A. (eds.) Proceedings of the 12th International Conference on Data Science, Technology and Applications, DATA 2023, Rome, Italy, July 11-13, 2023. pp. 474–479. SCITEPRESS (2023), https://doi.org/10.5220/0012131700003541
17. Nicolai, R.P., Dekker, R.: Optimal Maintenance of Multi-component Systems: A Review, pp. 263–286 (2008)
18. Olston, C., Widom, J.: Offering a precision-performance tradeoff for aggregation queries over replicated data. In: Abbadi, A.E., Brodie, M.L., Chakravarthy, S., Dayal, U., Kamel, N., Schlageter, G., Whang, K. (eds.) VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt. pp. 144–155. Morgan Kaufmann (2000), http://www.vldb.org/conf/2000/P144.pdf
19. Weitzman, M.L.: Optimal search for the best alternative. Econometrica **47**(3), 641–654 (1979), http://www.jstor.org/stable/1910412
20. Wiseman, Y., Schwan, K., Widener, P.M.: Efficient end to end data exchange using configurable compression. ACM SIGOPS Oper. Syst. Rev. **39**(3), 4–23 (2005), https://doi.org/10.1145/1075395.1075396