Online Appointment Scheduling in the Random Order Model

Oliver Göbel[1], Thomas Kesselheim[2,⋆], and Andreas Tönnis[1,⋆⋆]

[1] Department of Computer Science, RWTH Aachen University, Germany
{goebel,toennis}@cs.rwth-aachen.de

[2] Max-Planck-Institut für Informatik and Saarland University, Saarbrücken, Germany
thomas.kesselheim@mpi-inf.mpg.de

Abstract. We consider the following online appointment scheduling problem: Jobs of different processing times and weights arrive online step-by-step. Upon arrival of a job, its (future) starting date has to be determined immediately and irrevocably before the next job arrives, with the objective of minimizing the average weighted completion time. In this type of scheduling problem it is impossible to achieve non-trivial competitive ratios in the classical, adversarial arrival model, even if jobs have unit processing times. We weaken the adversary and consider random order of arrival instead. In this model the adversary defines the weight-processing time pairs for all jobs, but the order in which the jobs arrive online is a permutation drawn uniformly at random.

For the case of jobs with unit processing time we give a constant-competitive algorithm. We use this algorithm as a building block for the general case of variable job processing times and achieve competitive ratio $O(\log n)$. We complement these algorithms with a lower bound of $\Omega(n)$ for unit-processing time jobs in the adversarial input model.

1 Introduction

In scheduling problems there is a number of jobs given and the scheduler decides how, when and where each job is processed. The resulting schedule is evaluated under some objective function, typically minimizing a cost function on the completion time of the jobs. Often such schedules need to be computed online, meaning the scheduler has to make decisions without knowing the complete input for the problem instance. In a standard online variant, once a job is completed, the scheduler selects which job to process next from the jobs that have arrived in the meantime. Even worst-case competitive analysis admits surprisingly good results in this model (see Section 1.1 for an overview).

In stricter settings, worst-case analysis is overly pessimistic and therefore unable to produce meaningful results. Consider the following *appointment scheduling problem*: Jobs arrive online one after the other and directly upon arrival each

⋆ Supported in part by the DFG through Cluster of Excellence MMCI.

⋆⋆ Supported by the DFG GRK/1298 "AlgoSyn".

N. Bansal and I. Finocchi (Eds.): ESA 2015, LNCS 9294, pp. 680–692, 2015.
DOI: 10.1007/978-3-662-48350-3_57

job has to be assigned a starting and completion time in the future so as to optimize some objective function such as the weighted sum of completion times. As a motivation, just imagine a doctor's receptionist taking phone calls and making appointments for the next day. On the one hand he does not want to re-schedule appointments, on the other hand the most urgent cases should be treated first. In worst-case analysis, it is impossible to achieve any non-trivial competitive ratios for this kind of problem as we illustrate by giving a lower bound later on.

An interesting way to bypass these impossibility results is to incorporate a small stochastic component into the input model. A prime example of this phenomenon from a different domain is the *secretary problem*: A sequence of entities with different scores show up one after the other. After an entity has arrived, one has to make an irrevocable decision whether to keep this entity and to stop the sequence or to discard the entity and continue. Assuming a worst-case input, it is impossible to achieve a non-trivial competitive ratio. The situation is different if the adversary determines the scores but the arrival order of the entities is drawn uniformly from all possible permutations. Under these circumstances, it is possible to pick the highest-scored entity with probability $\frac{1}{e}$. Generalizing this problem, in many online maximization settings, it is possible to achieve a constant competitive ratio assuming a random input order whereas worst-case analyses would be pointless.

These positive results hold for maximization problems, whereas scheduling problems are typically cost-minimization problems. This can mean a big difference, particularly in a probabilistic setting: In a maximization problem it is possible to achieve reasonable (expected) competitive ratios, even if the algorithm returns no solution at all with probability $1/2$. Many algorithms indeed exploit this fact by using a constant fraction of the input only for "statistical" purposes and dropping it afterwards. In a minimization problem, this is generally impossible as one is usually required to satisfy certain cover contraints such that dropping input elements is not readily possible. We show that the random-order assumption makes a significant difference in online minimization problems nevertheless.

Formal Problem Statement. We assume there are $n \in \mathbb{N}$ jobs and a single processing unit. Each job i has a specific weight $w_i \in \mathbb{R}^+$ as well as a processing time $l_i \in \mathbb{N}$. All jobs are to be processed sequentially without preemption. Thus a feasible solution is a vector of starting times s such that in no time-step two jobs are processed simultaneously, i.e. $s_j \notin [s_i, s_i + l_i)$ for all $i, j \in [n]$, $i \neq j$. The objective is to minimize the weighted sum of completion times, i.e., $\sum_{j=1}^{n} w_j C_j = \sum_{j=1}^{n} w_j (s_j + l_j - 1)$. In the scheduling literature, this optimization problem is also referred to as $1|| \sum_{j=1}^{n} w_j C_j$ in the online list model.

Jobs arrive sequentially and scheduling decisions have to be made immediately and irrevocably. Upon its arrival, a job's processing time l_i and its weight w_i are revealed and the algorithm has to assign a position in the schedule to the job[1]. In

[1] Note that the *time slots* in this schedule are unrelated to the *rounds* in which the jobs arrive. Throughout the proofs, we will refer to the schedule slots by s and t, whereas the rounds will be referred to as r.

a fully adversarial input, even with identical job processing times, no randomized algorithm can be better than $\Omega(n)$-competitive (for a proof see Section 4), which is trivially achieved by any schedule without idle slots. Therefore, we consider the random-order model. Here an adversary constructs the instance and determines the processing times and weights of the jobs, but the arrival order is drawn at random. Technically, a permutation $\pi \in S_n$ is drawn uniformly at random and then the jobs are presented according to this permutation. The algorithm gets to know the processing time of the sequence n before the first round.

We evaluate our online algorithms in terms of the widespread *competitive ratio*. It is defined by $\frac{\mathbf{E}[\mathrm{ALG}]}{\mathrm{OPT}}$, where ALG and OPT denote the cost of the online and the optimal algorithm, respectively. The optimal algorithm, however, works offline and is assumed to know the whole instance in advance. In our algorithms, the expectation of their cost is only with respect to the random input.

Our Contribution. Our main contribution is an algorithm for the case of identical processing times. That is, similarly to the secretary problem, n entities of different weights are revealed online. We have to assign each entity to a slot 1, 2, ... so as to minimize the weighted sum of slot numbers. Our algorithm for this problem is 34-competitive. Specifically, the competitive factor holds for every single job and not only in expectation over all jobs. This means there is no job that suffers a bad position for the sake of the overall solution as each job is guaranteed to lose at most the competitive factor in expectation.

Upon arrival of a job, the algorithm computes the optimal schedule of all jobs seen so far. This solution includes a slot number for the currently considered job. We use this number as a guide to find the permanent slot for the online job. To this end, we scale the locally optimal solution with a factor depending on the fraction of the overall input that we have seen up to this point. This factor decreases as we learn more about the problem instance at hand. Finally we schedule the job to the first free slot after this tentative slot.

As a next step we generalize the setting toward jobs with different processing times. We present an $O(\log n)$-competitive algorithm. It divides the jobs into classes of almost equal processing times. For each processing time class it runs the algorithm for jobs with identical processing time as a subroutine. We devise a labeling scheme that associates slots to processing time classes and guarantees that every instance of the subroutine loses no more than a factor $2 \log n$ on top of its inherent competitive ratio.

We complement this algorithm with the simple lower bound that all online algorithms are $\Omega(n)$-competitive in the worst-case input order.

1.1 Related Work

In offline scheduling minimizing the weighted completion time is well understood. On a single machine the problem can be solved easily with Smith's ratio rule. For more complex versions with identical machines, related machines and release dates there are PTAS known [1,7]. A common online variant of the problem is as follows. The jobs are unknown to the algorithm until their respective release dates. At any point in time, the algorithm decides which of the released jobs to process next. This setting has been studied intensively when only a single machine or identical parallel machines are available as well as when preemption is allowed or not allowed, respectively; see [19,8,27,28,15] for detailed results. More recently a stochastic variant has been considered. Here the scheduler does not learn the true processing time of a job upon its arrival, but he learns a probability distribution over the processing time of the job instead; see [20,21] for the latest results. Stochastic processing times have also been considered in [5] regarding offline appointment scheduling. All these online results are in the adversarial input model, where an oblivious adversary creates the worst-case input sequence online.

Note that this setting (both the deterministic and the stochastic variant) is significantly different from the one studied in this paper. The crucial difference is that in the traditional model scheduling is an ad-hoc decision. It only affects (at most) the time until completing the currently processed job. In contrast, in our problem every job needs to be assigned its starting date irrevocably immediately upon its arrival. To the best of our knowledge, there is only a single previous result in this "online list scheduling model". Fiat and Woeginger [13] show that with worst-case input there is no $O(\log(n))$-competitive randomized online list scheduling algorithm, even on a single machine and with unit weights. We show that in the random-order model we can get constant- or $O(\log n)$-competitive even if jobs have weights.

The random-order model has been studied mostly in the context of packing problems. Motivated by the classical secretary problem, Babaioff et al. [4] introduced the matroid secretary problem and conjectured that it is $O(1)$-competitive. Toward this end several $O(\log\log(\rho))$-competitive algorithms have been proposed recently [18,12]. Here ρ is the rank of the considered matroid. Further variants of the secretary problem are given in [3], among them also one that aims at minimizing the sum of the accepted ranks. Another branch of research focuses on linear constraints. This includes (generalizations of) matching [22,9] and general packing LPs [11,2,26,16]. It is a common assumption in these problems that capacities are large compared to the consumption in a single round. In this case, there are even $1-\epsilon$-competitive algorithms.

Apart from scheduling, a few other min-sum online optimization problems have been studied. For *facility location* there is a deterministic lower bound of $\Omega(\log(n))$, while Meyerson [23] shows a constant competitive factor in the random order model. For *network design* [25] and the *parking permit problem* [24] only adversarial input has been studied. They admit $O(k)$-competitive algorithms where k is the number of options available. A general framework for linear online covering problems with adversarial input has been presented by Buchbinder and Naor [6]. However, naturally in weighted settings, the competitive ratio is limited by strong impossibility results, such as $\Omega(n)$ in our case.

A number of alternative online input models that combine adversarial and stochastic components have been studied. Devanur et al. [10] use the i.i.d. model and introduce generalization, the adversarial stochastic model. Kleinberg and Weinberg [17] consider the prophet inequality model, in which weights are stochastic but the arrival order is adversarial. In [14], a unifying graph sampling model is introduced that contains the random-order and prophet inequality model as special cases.

2 Jobs with Uniform Processing Time

We start with investigating the online appointment scheduling where all jobs' processing times are uniform, i.e. we consider them to be normalized to 1. The optimal solution is an ordering of the jobs, decreasing in their weight. In this setting the challenge is to order the online incoming jobs according to their weight when they arrive. Given a sufficiently large fraction of the unknown input, a job's relative position in this fraction is a quite good representative for its position in the complete input. This is why Algorithm 1 uses a local solution to guide the online computation. Generally, it works as follows: Upon the arrival of an element i the optimal ordering $\tilde{s}^{(r)}$ on set J containing all jobs that have arrived so far is computed. Afterwards, this local solution is scaled by a factor f_r in order to create sufficiently large gaps between the jobs. This steady distribution is essential for later insertions of jobs as they tend to be ranked between those ones scheduled up to now. The incoming job i is assigned a so-called *tentative slot* $hats_i$ in this scaled solution. As tentative slots are not unique, we solve eventual conflicts by assigning i to the first free slot s_i after its tentative position. At the end of the analysis it will become clear how to choose the parameters c and d.

Algorithm 1. Algorithm for Uniform Jobs

```
Let $J$ be the set of jobs arrived so far, initially $J = \emptyset$
for each round $r$ with incoming job $i$ do
    $J := J \cup \{i\}$;
    $\tilde{s}^{(r)} :=$ optimal solution in round $r$ on set $J$;
    $\hat{s}_i := \left\lceil f_r \cdot \tilde{s}_i^{(r)} \right\rceil$, where $f_r = c\left(\frac{n}{r}\right)^{1/d}$;
    $s_i :=$ earliest free slot after $\hat{s}_i$;
```

Theorem 1. *The algorithm for jobs with uniform processing time schedules every single job* 34*-competitive in expectation.*

The proof of this theorem will be split into three parts. First, in Section 2.1, we will bound the expected tentative slot number that is assigned to a job. Next, in Section 2.2, we bound the amount by which a job is shifted due to collisions, i.e., by how much the final slot differs from the tentative slot. Finally, in Section 2.3, we combine these insights to prove the claim.

Note that the job indices are irrelevant for the algorithm. Therefore, in the analysis, we assume that these indices are assigned such that $w_1 > w_2 > \ldots > w_n$. By this assumption, the optimal offline solution is simply $1, 2, \ldots, n$.

2.1 Bound on Tentative Slot Numbers

As a first step, for a fixed job i, we bound its tentative slot number. Note that, by our assumption $w_1 > w_2 > \ldots > w_n$ its slot number in the offline optimum would be i.

Lemma 2. *The expected tentative slot of job i is $\mathbf{E}[\hat{s}_i] \leq \frac{cd}{d-1} \cdot i + 1 + O\left(n^{1/d - 1}\right)$.*

Proof. The algorithm sets $\hat{s}_i = \left\lceil f_{\pi(i)} \tilde{s}_i^{(\pi(i))} \right\rceil \leq f_{\pi(i)} \tilde{s}_i^{(\pi(i))} + 1$. So

$$\mathbf{E}[\hat{s}_i - 1] \leq \mathbf{E}\left[f_{\pi(i)} \tilde{s}_i^{(\pi(i))}\right] = cn^{1/d} \sum_{r=1}^{n} \mathbf{Pr}[r = \pi(i)] \frac{1}{r^{1/d}} \mathbf{E}\left[\tilde{s}_i^{(\pi(i))} \,\middle|\, \pi(i) = r\right] ,$$

where π is the random permutation the jobs are presented in. Observe that $\tilde{s}_i^{(\pi(i))} - 1$ is exactly the number of jobs whose weight is larger than w_i that come in rounds before r. Conditioning on $\pi(i) = r$, the order of the remaining $n - 1$ jobs is still uniform. Out of these exactly $i - 1$ have a weight larger than w_i. In expectation, a $\frac{r-1}{n-1}$ fraction of these are assigned to rounds $1, \ldots, r - 1$. This gives us $\mathbf{E}\left[\tilde{s}_i^{(\pi(i))} \,\middle|\, \pi(i) = r\right] = (i - 1)\frac{r-1}{n-1} + 1$. Using that furthermore $\mathbf{Pr}[\pi(i) = r] = \frac{1}{n}$ for all r, we get

$$\mathbf{E}\left[f_{\pi(i)} \tilde{s}_i^{(\pi(i))}\right] \leq cn^{1/d} \left(\frac{i-1}{n} \sum_{r=1}^{n} \frac{r^{1-1/d}}{n} + \frac{1}{n} \sum_{r=1}^{n} \frac{1}{r^{1/d}} \right) .$$

We approximate both sums by the corresponding integrals (see full version). Regarding the bound on $\mathbf{E}[\hat{s}_i - 1]$, this gives us

$$\begin{aligned} \mathbf{E}[\hat{s}_i - 1] &\leq cn^{1/d} \left(\frac{i-1}{n^2} \left(\frac{1}{2 - \frac{1}{d}} n^{2-1/d} + n^{1-1/d} \right) + \frac{1}{n} + \frac{1}{n^{1/d}} \frac{1}{1 - \frac{1}{d}} \right) \\ &= c\left((i-1)\frac{1}{2 - \frac{1}{d}} + \frac{i-1}{n} + \frac{1}{n^{1-1/d}} + \frac{1}{1 - \frac{1}{d}} \right) \leq ci\frac{1}{1 - \frac{1}{d}} + \frac{c}{n^{1-1/d}} \end{aligned}$$

and therefore we get $\mathbf{E}[\hat{s}_i] \leq \frac{c}{1-\frac{1}{d}} \cdot i + \frac{c}{n^{1-1/d}} + 1$. □

2.2 From Tentative to Actual Slots

It still remains to bound the number of the actual slot that is assigned to a job i. To this end, we will use the following intuition. Imagine the schedule to be a queue, first all jobs that have tentative slots between slot 1 and $\hat{s}_i$ arrive. While processing these jobs, the arrival continues and more jobs come in. These new jobs are also processed before we start to work off i. We will use the fact that the average expected number of jobs tentative assigned to a slot is bounded by some $q < 1$. This causes the effects of this cascade to be bounded in expectation. To formalize this, we use the following technical lemma for a queueing process.

Lemma 3. *Consider non-negative integer random variables A_t such that there is $q \in (0,1)$ with the property that for any $t \in \mathbb{N}$ and $a_1, \ldots, a_{t-1} \in \mathbb{N}$, we have $\mathbf{E}\left[A_t \mid A_1 = a_1, \ldots, A_{t-1} = a_{t-1}\right] \leq q$. Furthermore, let $Q_0 \in \mathbb{N}$ and $Q_{t+1} = \max\left\{0, Q_t + A_{t+1} - 1\right\}$ and $T = \min\left\{t \mid Q_t = 0\right\}$, then $\mathbf{E}\left[T\right] \leq \frac{1}{1-q} Q_0$.*

Proof. We divide the time[2] waiting for $Q_t = 0$ into phases as follows: If phase p ends at time T_{p-1}, then phase p lasts exactly for $Q_{T_{p-1}}$ steps. The intuition is that we are waiting for a FIFO queue to become empty. The initial queue length is Q_0. After Q_0 steps, the initial elements of the queue have been processed. During this processing, additional elements may have arrived that need to be processed. This process continues until $Q_t = 0$ for the first time.

Formally, we set $T_p = T_{p-1} + Q_{T_{p-1}}$, $T_0 = 0$. By this definition we have $Q_{T_p} = Q_{T_{p-1}} + \sum_{t=T_{p-1}+1}^{T_p} A_t + (T_p - T_{p-1}) = \sum_{t=T_{p-1}+1}^{T_p} A_t$. Now induction gives us $\mathbf{E}\left[Q_{T_p} \mid A_1, \ldots, A_{T_{p-1}}\right] = \mathbf{E}\left[\sum_{t=T_{p-1}+1}^{T_p} A_t \,\middle|\, A_1, \ldots, A_{T_{p-1}}\right] \leq q(T_p - T_{p-1}) = Q_{T_{p-1}}$ using the condition on the expectation. This implies $\mathbf{E}\left[Q_{T_p}\right] \leq q\mathbf{E}\left[Q_{T_{p-1}}\right]$ and by induction $\mathbf{E}\left[Q_{T_p}\right] \leq q^p Q_0$.

We have $T = \max_p T_p$ and therefore $T = \sum_{p=0}^{\infty} Q_{T_p}$. By linearity of expectation, we get $\mathbf{E}\left[T\right] = \sum_{p=0}^{\infty} \mathbf{E}\left[Q_{T_p}\right] \leq \sum_{p=0}^{\infty} q^p Q_0 = \frac{1}{1-q} Q_0$. □

Using this lemma, we will show that the index of the actual slot a job is mapped to is at most a constant factor larger than the tentative slot. This proof is still technically involved because we have to be careful with dependencies. Besides, in each round there are only a few possible options for the respectively assigned tentative slot. So, the arrival is not as balanced as in Lemma 3.

Lemma 4. *Fix a job i and a round r. Conditioned on the event that job i comes in round r and gets tentative slot $\hat{s}_i$, the expected first feasible slot s_i is given by $\mathbf{E}\left[s_i \mid \pi(i) = r, \hat{s}_i\right] \leq \frac{1}{1-q} \hat{s}_i$ with $q = \left(\frac{r}{n}\right)^{1/d} \cdot \frac{2d}{c}$.*

Proof. Let A'_t be the random variable counting the number of jobs that are tentatively allocated onto slot t by the end of round $r - 1$. Analogously to Lemma 3 we define $Q'_t = \max\left\{0, Q'_{t-1} + A'_t - 1\right\}$ and $T' = \min\left\{t \mid Q'_t = 0\right\}$. If we set $Q'_0 = \hat{s}_i$ then $T' \geq s_i$ is an upper bound for the first feasible slot after $\hat{s}_i$. Unfortunately, Lemma 3 cannot be applied here because the A'_t are mutually dependent. To apply the lemma nevertheless, we define a set of variables A_t that are coupled to A'_t in such a way, that $\sum_{t' \leq t} A'_{t'} \leq \sum_{t' \leq t} A_{t'}$ holds for all t. Furthermore we choose $Q_0 = Q'_0$. It is easy to see that by this definition $Q'_t \leq Q_t$ for all t and therefore we also have $T' \leq T$. Thus it suffices to consider A_t to prove the lemma.

We choose A_t in such a way that we divert mass away from A'_t onto the $A'_{t'}$ with $t' < t$. As a first step, we balance the load between different slots, which we will exploit later. To this end, let $U_{r'}$ be drawn independently uniformly from $[0, f_{r'}]$, where $f_{r'} = c\left(\frac{n}{r}\right)^{1/d}$ is the scaling factor used in round r'.

[2] Note that the notion of time within this proof refers to the queueing perspective, not to the algorithm's input.

Define $Z_{r'} = f_{r'}\tilde{s}^{(r')}_{\pi^{-1}(r')} - U_{r'}$. So, $Z_{r'}$ is a real-valued random variable taking values on $[0, f_{r'}r']$. Indeed it is uniformly distributed on this interval because, conditioned on the set J in a round r', $\pi^{-1}(r')$ can be considered drawn uniformly from J. So, $\tilde{s}^{(r')}_{\pi^{-1}(r')}$ is drawn uniformly from $\{1, 2, \ldots, r'\}$. We will use $\lceil Z_{r'} \rceil$ as a lower bound on the tentative slot used in round r'. Therefore, define $X_{t,r'}$ to be 1 if $Z_{r'}\rceil = t$ and 0 otherwise. Now, let

$$A_t = \sum_{r' \leq r} \left(\frac{1}{b_{r'}} + \left(1 - \frac{t}{b_{r'}}\right) \cdot X_{t,r'} \right) \qquad \text{with } b_{r'} = \lceil f_{r'}r' \rceil = \left\lceil c \left(\frac{n}{r'}\right)^{1/d} r' \right\rceil \ .$$

To complete the proof of the lemma, it now remains to show the following two claims. (a) $\sum_{t' \leq t} A_{t'} \geq \sum_{t' \leq t} A'_{t'}$ for all t. (b) Given arbitrary numbers $a_1, \ldots, a_{t-1} \in \mathbb{N}$, we have $\mathbf{E}\left[A_t \mid A_1 = a_1, \ldots, A_{t-1} = a_{t-1}\right] \leq q$. Then Lemma 3 directly gives the desired result. Due to space limitations, the formal proof of Claim (a) can only be found in the full version.

Proof of Claim (b) For an arbitrary matrix $x = (x_{t',r'})_{t',r' \in \mathbb{N}}$, $x_{t',r'} \in \{0, 1\}$, let $\mathcal{E}_x$ be the event that $X_{t',r'} = x_{t',r'}$ for all $t' < t$ and all $r' \leq r$. We now upper-bound the value of $\mathbf{Pr}\left[X_{t,r'} = 1 \mid \mathcal{E}_x\right]$. Observe that $\lceil Z_{r'} \rceil \leq \lceil f_{r'}r' \rceil$. Therefore, if $t > f_{r'}r'$, we immediately have $X_{t,r'} = 0$. Furthermore, if $x_{t',r'} = 1$ for some $t' < t$, then also $X_{t,r'} = 0$. So, let us, without loss of generality, assume that $t \leq \lceil f_{r'}r' \rceil$ and $x_{t',r'} = 0$ for all $t' < t$.

We also observe that the algorithm only uses relative ranks to determine the slot allocation: In each round r'' the solution s always allocates the same slots S', regardless of the actual job weights. Therefore, $Z_{r'}$ can considered to be independent of all $Z_{r''}$, $r'' \neq r'$. Consequently, conditioned on $\mathcal{E}_x$, $Z_{r'}$ is uniformly distributed on $(t-1, b_{r'}]$. Therefore, we get

$$\mathbf{Pr}\left[X_{t,r'} = 1 \mid \mathcal{E}_x\right] \leq \frac{1}{b_{r'} - (t-1)} \ .$$

Given arbitrary numbers $a_1, \ldots, a_{t-1} \in \mathbb{N}$, we now bound the conditioned expectation $\mathbf{E}\left[A_t \mid A_1 = a_1, \ldots, A_{t-1} = a_{t-1}\right]$. To this end observe, that the event $A_1 = a_1, \ldots, A_{t-1} = a_{t-1}$ can equivalently be expressed by a set $\mathcal{X}$ of 0/1 matrices x with the property that $A_1 = a_1, \ldots, A_{t-1} = a_{t-1}$ if and only if there is $x \in \mathcal{X}$ such that $X_{t',r'} = x_{t',r'}$ for all $t' < t$ and all $r' \leq r$.

Using the above bound on $\mathbf{Pr}\left[X_{t,r'} = 1 \mid \mathcal{E}_x\right]$, we get

$$\begin{aligned}\mathbf{E}\left[A_t \mid \mathcal{E}_x\right] &= \sum_{r' \leq r} \left(\frac{1}{b_{r'}} + \left(1 - \frac{t}{b_{r'}}\right) \cdot \mathbf{Pr}\left[X_{t,r'} = 1 \mid \mathcal{E}_x\right] \right) \\ &\leq \sum_{r' \leq r} \left(\frac{1}{b_{r'}} + \left(1 - \frac{t-1}{b_{r'}}\right) \cdot \frac{1}{b_{r'} - (t-1)} \right) = \sum_{r' \leq r} 2\frac{1}{b_{r'}} = \frac{2}{cn^{1/d}} \sum_{r' \leq r} (r')^{1/d - 1} \ .\end{aligned}$$

As this bound holds for all $x \in \mathcal{X}$, we also have

$$\mathbf{E}\left[A_t \mid A_1 = a_1, \ldots, A_{t-1} = a_{t-1}\right] \leq \frac{2}{cn^{1/d}} \sum_{r' \leq r} (r')^{1/d - 1} \leq \left(\frac{r}{n}\right)^{1/d} \cdot \frac{2d}{c} \ . \qquad \square$$

2.3 Putting the Pieces Together

Using the insights from the previous sections, the proof of Theorem 1 is relatively straightforward.

Proof (of Theorem 1). Combining Lemmas 2 and 4, we get that for every job i

$$\mathbf{E}\left[s_i\right] \leq \sum_{i=1}^{n} w_i \cdot \frac{1}{1-\frac{2d}{c}} \mathbf{E}\left[\hat{s}_i\right] \leq \sum_{i=1}^{n} w_i \frac{1}{1-\frac{2d}{c}} \cdot \left(\frac{cd}{d-1} i + 1 + O\left(n^{1/d-1}\right)\right) .$$

We omit the O-term since it tends towards 0 for large n. Setting $c = 8$ and $d = 2$, we get $\frac{1}{1-\frac{2d}{c}} \cdot \left(\frac{cd}{d-1} i + 1\right) = 2(16i+1) \leq 34i$. Therefore for every job i the allocated slot s_i in expectation only deviates by a factor of 34 from its optimal slot s_i^* in the offline schedule. □

3 General Jobs

With the constant competitive algorithm for jobs with uniform processing time as a subroutine we devise an algorithm for jobs with variable processing times. We use several instances of Algorithm 1 to schedule jobs with similar processing time. We sort jobs into processing time classes where every class $\lambda = 2^b$ for $b \in \mathbb{N}$ contains the jobs with processing times between two powers of two $2^{b-1} < l_i \leq 2^b$. To this end we define a labeling scheme that maps the sub-schedules of the different processing time classes onto the overall schedule in such a way, that no job is pushed back by more than a factor of $2\log(n)$ compared to his position in the sub-schedule.

Theorem 5. *The algorithm for jobs with general processing time is $2\alpha\log(n)$-competitive where α is the competitive factor of the algorithm for jobs with uniform jobs used as subroutine.*

We start out with the labeling scheme that allows us to group slots to meta-slots, each associated to a single processing time class. Without loss of generality let the total number of jobs n be a power of two. Let T_k be a complete binary tree of height $\log(n)$. We call the level of the leaves $j = 0$. Now we label a node σ on level j of T_k with $\lambda(\sigma) = 2^{k+j}$. This way, the leaves get label 2^k, their parents get label 2^{k+1} up to the root with label $2^{k+\log n}$.

Now we traverse the tree T_0 in post-order and map its nodes onto the slots in our schedule. In this traversal order we descend left first and map a parent node right after all his children. For each node σ with label $\lambda(\sigma)$ we create a meta-slot of $\lambda(\sigma)$ many neighboring slots. We proceed to map T_1 starting from the first free slot after T_0 and so on.

Observation 6. *The mapping of tree T_k requires $2^k n\log(n)$ slots in the schedule and starts at slot $(2^k-1)n\log(n)+1$.*

This follows simply from the fact that every level j in tree T_k contains $\frac{n}{2^j}$ nodes and each node takes 2^{j+k} slots.

Lemma 7. *The γ-th meta slot with label λ ends no later than slot* $2\log(n)\gamma\lambda$ *in the schedule.*

Proof. The first occurrence of label λ is in tree T_k with $k = 0$ if $\lambda \leq n$ and $k = \log(\lambda) - \log(n)$ otherwise. Therefore tree $T_{k'}$ with $k' = 0$ if $\lambda \leq n$ and $k' = \log(\lambda) + \log(\gamma) - \log(n)$ otherwise is the first tree that contains at least γ meta-slots with label λ.

Now we make a case distinction, if $k' = 0$, then label λ is used on level $j = \log(\lambda)$. In the post-order traversal when reaching the γ-th node with label λ, the subtrees of previous nodes of label λ have been traversed plus the subtree of the current node plus at most a $\frac{\gamma}{n/2^j}$-fraction of all nodes on higher levels. Each subtree takes $\log(\lambda)\lambda$ many slots. There are $\left\lfloor \frac{\gamma}{n/2^j} \frac{n}{2^{-j}} \right\rfloor \leq \gamma 2^{j-j'}$ nodes on higher levels $j' = \log(\lambda) + 1, \ldots, \log n$, each taking $2^{j'}$ slots. So the γ-th node with label λ ends on slot $\gamma \log(\lambda)\lambda + (\log n - \log(\lambda))\gamma 2^j = \log(n)\gamma\lambda$.

In the other case, if $k' \neq 0$ it follows from Observation 6 that tree $T_{k'}$ starts at slot $(2^{k'} - 1)n\log(n) + 1 \leq \lambda + \gamma n \log(n) + 1 = 2^{\log(\gamma)+\log(\lambda)-\log(n)} n \log(n) + 1 = \gamma\lambda\log(n) + 1$. Furthermore label λ is used on level $j = \log(\lambda) - k'$ in tree $T_{k'}$ and the leaves in tree $T_{k'}$ take $2^{k'}$ slots each. Thus the total number of slots used through the post-order traversal on $T_{k'}$ is bounded by$\gamma(\log(\lambda) - k') \cdot 2^{\log(\lambda)-k'} \cdot 2^{k'} + (\log n - (\log(\lambda) - k'))\gamma\lambda = \log(n)\gamma\lambda$. So, the γth slot of label λ ends no later than $2\log(n)\gamma\lambda$. □

Algorithm 2. Log-Algorithm for Jobs with different processing times

```
for each round $r$ with incoming job $i$ having processing time $l_i$ do
    choose $b$ such that $2^{b-1} < l_i \leq 2^b$;
    Let $J_b := \{\text{jobs } j \text{ with } l_j \in (2^{b-1}, 2^b]\}$;
    Let $\Sigma_b := \{\text{meta-slots } \sigma | \lambda(\sigma) = 2^b\}$;
    $s^{(b)}$ = output from uniform algorithm with job $i$ on known $J_b$;
    schedule $i$ on $s_i$-th meta-slot in $\Sigma_b$;
```

Proof (of Theorem 5). We run one instance of an α-competitive algorithm for jobs with uniform processing time for every processing time class as a subroutine. These subroutines give for every job i a $\gamma = s_i^{(b)}$ and a $\lambda = 2^b$ such that $2^{b-1} < l_i \leq 2^b$. Now meta-slot γ of the subroutine would end in slot $\gamma\lambda$ if no other labels were interwoven. Therefore the labeling stretches the α-competitive schedule $s^{(b)}$ by an additional factor of $\frac{2\log(n)\gamma\lambda}{\gamma\lambda} = 2\log(n)$. □

4 Lower Bound for Fully Worst-Case Input

As mentioned before, we motivate the use of the random order model by giving a lower bound when performing classical worst-case analysis in the general setting with different job processing times and weights. There, an adversary is allowed to construct the instance, i.e. to determine the jobs' weights, and then also to present the jobs in any preferred order. This, obviously, is more powerful, but we show that it does not allow designing algorithms that achieve a reasonable competitive ratio. As we use a randomized instance, we can extend our results to even hold for randomized algorithms by applying Yao's principle.

Theorem 8. *For every randomized online algorithm weights can be chosen in a way such that* $\mathbf{E}\left[\mathrm{ALG}\right] \geq \frac{n}{8} \cdot \mathrm{OPT}$*, even if all jobs have equal processing times.*

Proof. We show this claim by using Yao's principle. We will devise a randomized instance such that for any deterministic algorithm $\mathbf{E}\left[\frac{\mathrm{ALG}}{\mathrm{OPT}}\right] \geq \frac{n}{8}$. To this end, let T be drawn uniformly from $\{1, \ldots, n\}$. We define the weights of a job arriving in round t to be $w_t = M^t$ if $t \leq T$ and $w_t = 0$ otherwise, with $M > n$. First, we compute the cost incurred by an optimal algorithm. According to the construction described above, T many jobs with non-zero weights $M, M^2, \ldots, M^T$ are given. It is obviously the best solution to put the heaviest job first, and then proceed with the jobs decreasing in their weight. Formally, job j is assigned slot $T-j+1$. This results in cost $\mathrm{OPT} = \sum_{j=1}^{T} M^j \left(T - j + 1\right) = \sum_{j=0}^{T-1} M^{T-j}(j+1) \leq 2 \cdot M^T$.

Now we focus on the cost of a deterministic online algorithm ALG. Until (including) round T, the behavior of this algorithm is independent of T. Its behavior after this point is irrelevant for the resulting cost. Therefore, we can express the algorithm's choices as an injective function $\sigma\colon [n] \to \mathbb{N}$, meaning that the ith job is scheduled to slot $\sigma(i)$.

As the function is injective, there is a set $S \subseteq [n]$ of size $\left\lceil \frac{n}{2} \right\rceil$ such that $\sigma(i) \geq \left\lfloor \frac{n}{2} \right\rfloor + 1 \geq \frac{n}{2}$ for all $i \in S$. Note that if $T \in S$, then $\mathrm{ALG} \geq \frac{n}{2} \cdot M^T \geq \frac{n}{2} \cdot \frac{\mathrm{OPT}}{2}$, so $\frac{\mathrm{ALG}}{\mathrm{OPT}} \geq \frac{n}{4}$. As $T \in S$ happens with probability at least $\frac{1}{2}$, we get $\mathbf{E}\left[\frac{\mathrm{ALG}}{\mathrm{OPT}}\right] \geq \frac{n}{8}$. □

References

1. Afrati, F.N., Bampis, E., Chekuri, C., Karger, D.R., Kenyon, C., Khanna, S., Milis, I., Queyranne, M., Skutella, M., Stein, C., Sviridenko, M.: Approximation schemes for minimizing average weighted completion time with release dates. In: Proc. 40th Symp. Foundations of Computer Science (FOCS), pp. 32–44 (1999)
2. Agrawal, S., Wang, Z., Ye, Y.: A dynamic near-optimal algorithm for online linear programming. Operations Research 62(4), 876–890 (2014)
3. Ajtai, M., Megiddo, N., Waarts, O.: Improved algorithms and analysis for secretary problems and generalizations. SIAM J. Discrete Math. 14(1), 1–27 (2001)
4. Babaioff, M., Immorlica, N., Kleinberg, R.: Matroids, secretary problems, and online mechanisms. In: Proc. 18th Symp. Discr. Algorithms (SODA), pp. 434–443 (2007)
5. Begen, M.A., Queyranne, M.: Appointment scheduling with discrete random durations. Math. Oper. Res. 36(2), 240–257 (2011)
6. Buchbinder, N., Naor, J.: Online primal-dual algorithms for covering and packing. Math. Oper. Res. 34(2), 270–286 (2009)

7. Chekuri, C., Khanna, S.: A PTAS for minimizing weighted completion time on uniformly related machines. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) ICALP 2001. LNCS, vol. 2076, pp. 848–861. Springer, Heidelberg (2001)
8. Correa, J.R., Wagner, M.R.: Lp-based online scheduling: from single to parallel machines. Math. Program. 119(1), 109–136 (2009)
9. Devanur, N.R., Hayes, T.P.: The adwords problem: online keyword matching with budgeted bidders under random permutations. In: Proc. 10th Conf. Electr. Commerce (EC), pp. 71–78 (2009)
10. Devanur, N.R., Jain, K., Sivan, B., Wilkens, C.A.: Near optimal online algorithms and fast approximation algorithms for resource allocation problems. In: Proc. 12th Conf. Electr. Commerce (EC), pp. 29–38 (2011)
11. Feldman, J., Henzinger, M., Korula, N., Mirrokni, V.S., Stein, C.: Online stochastic packing applied to display ad allocation. In: de Berg, M., Meyer, U. (eds.) ESA 2010, Part I. LNCS, vol. 6346, pp. 182–194. Springer, Heidelberg (2010)
12. Feldman, M., Svensson, O., Zenklusen, R.: A simple O(log log(rank))-competitive algorithm for the matroid secretary problem. In: Proc. 26th Symp. Discr. Algorithms (SODA), pp. 1189–1201 (2015)
13. Fiat, A., Woeginger, G.J.: On-line scheduling on a single machine: Minimizing the total completion time. Acta Inf. 36(4), 287–293 (1999)
14. Göbel, O., Hoefer, M., Kesselheim, T., Schleiden, T., Vöcking, B.: Online independent set beyond the worst-case: Secretaries, prophets, and periods. In: Esparza, J., Fraigniaud, P., Husfeldt, T., Koutsoupias, E. (eds.) ICALP 2014, Part II. LNCS, vol. 8573, pp. 508–519. Springer, Heidelberg (2014)
15. Günther, E., Maurer, O., Megow, N., Wiese, A.: A new approach to online scheduling: Approximating the optimal competitive ratio. In: Proc. 24th Symp. Discr. Algorithms (SODA), pp. 118–128 (2013)
16. Kesselheim, T., Radke, K., Tönnis, A., Vöcking, B.: Primal beats dual on online packing LPs in the random-order model. In: Proc. 46th Symp. Theory of Computing (STOC), pp. 303–312 (2014)
17. Kleinberg, R., Weinberg, S.M.: Matroid prophet inequalities. In: Proc. 44th Symp. Theory of Computing (STOC), pp. 123–136 (2012)
18. Lachish, O.: O(log log rank) competitive ratio for the matroid secretary problem. In: Proc. 55th Symp. Foundations of Computer Science (FOCS), pp. 326–335 (2014)
19. Megow, N., Schulz, A.S.: On-line scheduling to minimize average completion time revisited. Oper. Res. Lett., 32(5):485–490 (2004)
20. Megow, N., Uetz, M., Vredeveld, T.: Models and algorithms for stochastic online scheduling. Math. Oper. Res., 31(3):513–525 (2006)
21. Megow, N., Vredeveld, T.: A tight 2-approximation for preemptive stochastic scheduling. Math. Oper. Res. 39(4), 1297–1310 (2014)
22. Mehta, A., Saberi, A., Vazirani, U.V., Vazirani, V.V.: Adwords and generalized online matching. J. ACM 54(5) (2007)
23. Meyerson, A.: Online facility location. In: Proc. 42nd Symp. Foundations of Computer Science (FOCS), pp. 426–431 (2001)
24. Meyerson, A.: The parking permit problem. In: Proc. 46th Symp. Foundations of Computer Science (FOCS), pp. 274–284 (2005)

25. Meyerson, A., Munagala, K., Plotkin, S.A.: Designing networks incrementally. In: Proc. 42nd Symp. Foundations of Computer Science (FOCS), pp. 406–415 (2001)
26. Molinaro, M., Ravi, R.: The geometry of online packing linear programs. Math. Oper. Res. 39(1), 46–59 (2014)
27. Schulz, A.S., Skutella, M.: The power of α-points in preemptive single machine scheduling. Journal of Scheduling 5, 121–133 (2002)
28. Sitters, R.: Competitive analysis of preemptive single-machine scheduling. Operations Research Letters 38(6), 585–588 (2010)