On-line scheduling

Now we consider an on-line model in which time plays a role. Before any time t nothing is known what comes at time t or after. Decisions may be postponed at the price of time that is passing by, for which one pays in objective value. Consider the off-line scheduling problem $1 \mid r_j \mid \sum C_j$:

Total completion time scheduling with release dates
Given a single machine and n jobs, each having a processing time p_j and a release date r_j. Find a schedule of the jobs such that each job j is scheduled in an uninterrupted time interval of length p_j which does not start before r_j and such that the sum of the completion times of the jobs is minimised.

This problem is NP-hard, but there exists a PTAS. The version without release dates is easily seen to be solved by the *shortest processing time first* rule (SPT). (Just use a simple exchange argument.) Similarly easy it is to prove that the version in which preemption of the jobs is allowed, then the *shortest remaining processing time first* rule (SRPT) solves the problem.

We are interested in the online version. We assume that at time 0 we know nothing. Jobs are released over time and as soon as a job j is released we immediately get to learn its processing time. Before the release time we know nothing about the job, not even that it exists. Once we have a job and the machine is idle we may decide to start the job, and then we have to finish it uninterruptedly, or we may wait and see if other (smaller) jobs arrive soon, which we may like to give precedence. So we are allowed to postpone decisions. Only history is irrevocable.

Let us first think about how things can go wrong for an online algorithm due to this lack of information: i.e, let us try to find a lower bound on the *competitive* ratio for any algorithm.

Suppose at time 0 we see job 1 with $p_1 = 1$. Suppose an online algorithm start this job at time x. Then if $x \geq 1$, the adversary will not release any further job and he finishes at time 1, while the online algorithm does not finish before time 2, yielding a competitive ratio of at least 2.

So suppose that $x < 1$. Then the adversary releases at time $x + \epsilon$ an enormous set of n jobs with processing time very small. Think of these jobs as having processing time 0. Hence the adversary schedules the small jobs first adding $n(x + \epsilon)$ to the objective value and then the first job adding $x + \epsilon + 1$; a total of $(n + 1)(x + \epsilon) + 1$. Whereas the online algorithm has just started job 1 and has to wait until it completes before it can schedule the small jobs, yielding a total completion time of $(n + 1)(x + 1)$. For $\epsilon \to 0$ and $n \to \infty$, we obtain a ratio of $(x + 1)/x$, which is minimised by $x = 1$, and in that case is equal to 2.

Theorem 0.1. *No deterministic online algorithm for on-line minimising total completion time on a single machine can have a competitive ratio smaller than 2.*

In the literature several algorithms have been proposed that have a competitive ratio of 2. Let us study the most elegant among them. It is also the algorithm that can be used as the basis for a more sophisticated randomised algorithm, to which we come back later.

The algorithm just ignores the forbidding of preemption and uses the (online) SRPT-rule to build a schedule. This preemptive schedule is then used as the basis for building a non-preemptive schedule in the following way.

Algorithm: α-POINT.
Simulate on a single machine the (on-line by SRPT) connstruction of a preemptive schedule P. As soon as in this preemptive schedule an α-fraction of the processing time of a job has been completed, make it available for scheduling in the non-preemptive schedule by adding this job at the back of the list of jobs that became available before. Always schedule a job as soon as the machine completed all the previous jobs in the list of available jobs.

So just to repeat: as soon as a job in the preemptive schedule has seen an α-fraction of its processing time completed, it is made available for the non-preemptive schedule and is scheduled directly after the jobs for which an α-fraction of their processing time has been completed in the preemptive schedule before.

Our algorithm will not start the j-th job in the list of available jobs until the $j-1$-st is completed even if it may be profitable to switch them if at some time t they are both available and jod $j-1$ has longer processing time than job j. Our analysis does not require this algorithmic enhancement.

Let C_j^α be the completion time according to the α-point schedule and C_j^P according to the preemptive schedule.

Theorem 0.2. *Given a preemptive schedule P for $1 \mid r_j, pmtn \mid \sum C_j$, algorithm α-POINT yields in $O(n)$ time, a non-preemptive schedule in which, $\sum_j C_j^\alpha \leq (1 + (1/\alpha)) \sum_j C_j^P$.*

Proof. Index the jobs by the order of their α-points in the preemptive schedule P. We distinguish two situations.

In the first one, when j's α-point is reached (when job j becomes available for the non-preemptive online schedule, the machine on which we are building the online (non-preemptive) schedule is idle. In that case we start the job immediately and finish latest αp_j after C_j^P. Hence $C_j^\alpha \leq (1+\alpha)C_j^P \leq (1 + (1/\alpha))C_j^P$,

since $\alpha \leq 1$.

In the second situation at j's α-point the machine on which we are building the online (non-preemptive) schedule is processing some earlier available job. But then the last time the machine was idle was at the α-point of some earlier available job. Call this time t. And hence,

$$C_j^\alpha \leq t + \sum_{k=1}^{j} p_k. \tag{1}$$

Clearly $C_j^P \geq t$. We also know that $C_j^P \geq \alpha \sum_{k=1}^{j} p_k$, since the α-fractions of jobs $1, \ldots, j$ must run before time C_j^P. Plugging these last two inequalities into (1) yields the proof. □

Clearly, the best value for α in the range is $\alpha = 1$, yielding a ratio of 2.

To see that for each α the bound in the theorem is asymptotically tight, consider the following class of instances. At time 0, we release a job with processing time 1. At time $\alpha - \epsilon$ we release a job with processing time ϵ and at time $\alpha + \epsilon$ we release n jobs of processing time 0.

The optimal solution processes the first job last and the others in the order given, yielding a total completion time of $\alpha + n(\alpha + \epsilon) + \alpha + \epsilon + 1$.

The α-point algorithm will first schedule job 2 then job 1 and then the n jobs of length 0, yielding a total completion time of $\alpha + (\alpha + 1) + n(\alpha + 1)$. For $n \to \infty$ and $\epsilon \to 0$ the ratio tend to $1 + (1/\alpha)$.

This suggest that we can fool the adversary by not revealing the α we choose. This is exactly the motivation of the randomized algorithm, which plays against an oblivious adversary. Thus the adversary knows the distribution with which the randomized α-point algorithm is playing but does not see the realisation. We wish to study the ratio: $E[\sum C_j^\alpha / \sum C_j^{OPT}]$.

Algorithm RANDOMIZED α-POINT: draw $\alpha \in (0, 1]$ according to probability density function f and apply α-POINT.

To analyze the competitive ratio of this algorithm, let us do a slightly more precise analysis of what happens with the deterministic α-point algorithm. Take the completion time of job i in the preemptive schedule C_i^P. Let J_i^β the set of jobs that by time C_i^P have processed exactly a β-fraction of their processing time. Let $S_i^\beta = \sum_{j \in J_i^\beta} p_j$. Let T_i be the total idle time before C_i^P. It is easy to see that

Lemma 0.3. $C_i^P = T_i + \sum_{0 \leq \beta \leq 1} \beta S_i^\beta$

Let us split it out a bit more:

$$C_i^P = T_i + \sum_{\beta<\alpha} \beta S_i^\beta + \sum_{\beta\geq\alpha} \alpha S_i^\beta + \sum_{\beta\geq\alpha} (\beta-\alpha) S_i^\beta$$

Let $J^B = \{j \in \cup_{\beta\geq\alpha} J_i^\beta\}$. Clearly the jobs that run before job i in the α-point schedule belong to J^B.

Now we are changing the preemptive schedule. The following procedure is shown in a picture on the blackboard and can be read in the lecture notes. We move pieces of jobs in the preemptive schedule such that jobs in J^B get all their processing time that they had in the preemptive schedule between their α-point and C_i^P directly after their α-points (possibly pushing job-pieces to a later time in the schedule in case overlap would occur). Clearly C_i^P does not become larger, since we are only reshuffling pieces of jobs before C_i^P and don't insert idle time.

Let x_j be the fraction of job j finished by time C_i^P. Thus we have that for any job $j \in J^B$ a piece of size $(x_j - \alpha)p_j$ is schedules consecutively after j's α-point.

Then we move for each job in J^B all the $p_j - (x_j - \alpha)p_j$ remaining processing time (that is the processing time of j remaining to be done at time C_i^P plus the αp_j time done before j's α-point) directly after the piece of size $(x_j - \alpha)p_j$.

We then have a schedule in which all jobs that come before i in the α-point schedule are completed non-preemptively before job i is completed. The completion time of job i in this schedule is therefore an upper bound on its completion time in the α-point schedule. It has become bounded by:

$$C_i^P + \sum_{j\in J^B} p_j - (x_j - \alpha)p_j = C_i^P + \sum_{\beta\geq\alpha} (1 - \beta + \alpha) S_i^\beta$$

Inserting $C_i^P = T_i + \sum_{0\leq\beta\leq 1} \beta S_i^\beta$ from the lemma yields

$$C_i^\alpha \leq T_i + \sum_{\beta\geq\alpha} (1+\alpha) S_i^\beta + \sum_{\beta<\alpha} \beta S_i^\beta. \tag{2}$$

Based on this bound we will be able to give a bound on the expected completion time for each job i in the randomised α-point schedule.

Theorem 0.4. *Given that α is drawn according to density f over $[0,1]$, we have for each job i that $E[C_i^\alpha \leq (1+\delta)C_i^P]$, with*

$$\delta = \max \int_0^\beta \frac{1+\alpha-\beta}{\beta} f(\alpha) d\alpha.$$

The proof is really just taking (2) and taking integral over α of the right-hand side times $f(\alpha)$

$$E[C_i^\alpha] \leq \int_0^1 \left(T_i + \sum_{\beta\geq\alpha} (1+\alpha) S_i^\beta + \sum_{\beta<\alpha} \beta S_i^\beta \right) f(\alpha) d\alpha.$$

and then taking outside of the integral the terms that are independent of α; i.e., T_i and S_i^β. You can read it yourself in the lecture notes.

If we draw α according to density $f(\alpha) = \frac{e^\alpha}{e-1}$ then $\delta = \frac{1}{e-1}$. Thus, the following competitive ratio follows directly.

Theorem 0.5. RANDOMISED α-POINT *with* α *according to density* $f(\alpha) = \frac{e^\alpha}{e-1}$ *has competitive ratio* $\frac{e}{e-1} \approx 1.58$.

It can be shown that this is in fact the best ratio that can be obtained by any randomised algorithm on this on-line problem. For that I'll explain a very powerful tool to prove such lower bounds.

Explain Yao's minimax principle. A good explanation of this technique is found in the book R. Motwani & P. Raghavan, *Randomised Algorithms*, Cambridge University Press.

Applying Yao's minimax principle we specify a random instance of the problem and analyse what any algorithm could attain in expectation on this instance.

- At time 0 one job with processing time 1 arrives.
- With probability $1 - \frac{e-1}{n}$ no further jobs arrive.
- With probability $\frac{e-1}{n}$ one set of $n-1$ jobs with processing time 0 arrives at some time x, which is a random variable over the interval (0,1] having probability density function $f(x) = \frac{e}{e-1}\, e^{-x}$.

First we derive the expected optimal objective value $E[\sum C_j^{OPT}]$ on the random instance. Observe that if only the first job is released the optimal value is equal to 1. If at any time $x \leq 1 - \frac{1}{n}$ the set of $n-1$ jobs is also released then it is profitable to schedule the $n-1$ jobs before the first job, yielding an objective value of $nx+1$. If the $n-1$ jobs arrive between time $1-\frac{1}{n}$ and 1 it is better to process the first job first giving a sum of completion times equal to n. These observations lead to the following upper bound on the expected optimal solution value.

$$\begin{aligned}
E[\sum C_j^{OPT}] &\leq \left(1 - \frac{e-1}{n}\right) \cdot 1 + \frac{e-1}{n} \int_0^1 (nx+1) f(x) dx \\
&= 1 - \frac{e-1}{n} + \frac{e}{n} \int_0^1 e^{-x}(nx+1) dx \\
&= 1 - \frac{e-1}{n} + \frac{e}{n}(-ne^{-1} - ne^{-1} - e^{-1} + n + 1) \\
&= e - 1.
\end{aligned}$$

Any deterministic on-line algorithm will have to start scheduling the first job at some point in time. Consider an algorithm that starts processing the first

job at time t, unless the set of $n-1$ jobs arrives before t. In the latter case it is obviously better for the algorithm to schedule the $n-1$ jobs first, before starting the (big) first job. Moreover, since the deterministic algorithm knows the distribution, he will start the first job immediately after having processed the other jobs since no further jobs will arrive. In this case a cost of $nx+1$ will be incurred. In case the set of $n-1$ jobs arrives after t then these jobs have to wait until the first job is finished producing an objective value of $(t+1)n$. Obviously, if the set of $n-1$ jobs does not arrive the only job will be completed at time $t+1$. We will denote the expected solution value of an on-line algorithm that does not start the first job before time t by $E[\sum C_j^{OL(t)}]$. Now,

$$\begin{aligned} E[\sum C_j^{OL(t)}] &= (1-\frac{e-1}{n})(t+1) + \frac{e-1}{n}\int_0^t \frac{e}{e-1}e^{-x}(nx+1)dx \\ &\quad + \frac{e-1}{n}\int_t^1 \frac{e}{e-1}e^{-x}(t+1)ndx \\ &= e - \frac{t(e-1)+e^{1-t}-1}{n} \\ &\geq e - \frac{e-1}{n}. \end{aligned}$$

This last inequality follows from minimizing the expected value with respect to t over the interval $[0,1]$. The minimum is obtained at either $t=0$ or $t=1$. Thus, for any $t \in [0,1]$,

$$\frac{E[\sum C_j^{OL(t)}]}{E[\sum C_j^{OPT}]} \geq \frac{e-\frac{e-1}{n}}{e-1}.$$

The ratio can be made arbitrarily close to $\frac{e}{e-1}$, by choosing n large enough. The observation that it is useless for any algorithm to start the first job after time 1 shows that the above ratio holds for the best possible deterministic algorithm on this random instance.

Some of you might object that we do not prove a lower bound on $E[\frac{\sum C_j^{OL(t)}}{\sum C_j^{OPT}}]$, but the following lemma shows that what we showed is in fact enough.

Lemma 0.6. *Given an on-line optimization problem, with possible input sequences $\mathcal{I}$, and possible algorithms $\mathcal{A}$, both possibly infinite, for any random sequence I_p, and any randomized algorithm A_q, we have*

$$\min_{A\in\mathcal{A}} \frac{E_{I_p}[Z^A(I_p)]}{E_{I_p}[Z^{OPT}(I_p)]} \leq \max_{I\in\mathcal{I}} \frac{E_{A_q}[Z^{A_q}(I)]}{Z^{OPT}(I)}$$

provided that the left hand side is bounded.

Proof. Suppose that there exists a randomized algorithm A_q with competitive ratio c. Then $\forall I \in \mathcal{I}$

$$cZ^{OPT}(I) \geq E_{A_q}[Z^{A_q}(I)].$$

Hence,

$$\begin{aligned} cE_{I_p}[Z^{OPT}(I_p)] &\geq E_{I_p}[E_{A_q}[Z^{A_q}(I_p)]] \\ &= E_{A_q}[E_{I_p}[Z^{A_q}(I_p)]] \\ &\geq \min_{A \in \mathcal{A}} E_{I_p}[Z^{A}(I_p)]. \end{aligned}$$

□

Exercises

Exercise 1. Consider the following scheduling problem with *set-up times*:

Given is a set of n jobs and a set of m machines. Each job has a processing time p_j. Jobs may be preempted and, even stronger, several machines can work on any job simultaneously. However, before each part of job j that is processed a fixed time s_j must be spend on preparing the machine for job j. You may assume that $s_j = s$ for all j. The objective is to minimize the total completion time.

For the online version of this problem, in which jobs have also (unknown) release dates, find a lower bound in the competitive ratio for deterministic online algorithms and design a deterministic algorithm and analyse its competitive ratio. If you like you may take two machines ($m = 2$). Do not worry if you do not find matching lower and upper bounds.