ONLINE ALGORITHM

Felix Céard-Falkenberg

October 6, 2025

1 Related Work

Calinescu et al. (2024) show how to prove competitive ratios for online scheduling problems/algorithms.

The Round Robin Algorithm (Kleinrock, 1964) is a popular choice for online scheduling problems, especially in the context of preemtive scheduling. While many different problems have been explored, such as the case where the jobs length is unknown, even at release time, the Round Robin algorithm is still a good strategy for an online algorithm. For example, allowing for preemption and weighting each job by a reward, Kim and Chwa (2003) introduce a 2-competitive algorithm based on the Round Robin Algorithm.

Note that in the case where no information is known about the job, the any online algorithm has a competitive ratio of at least $\Omega\left(\sqrt[3]{n}\right)$ (Motwani et al., 1994).

Recently, Jäger et al. (2025) examined the case where pausing a job forces the job to be restarted from the beginning, which they argue is a under-explored problem and is inspired by how computer processes are handled. While their problem is poorly related to ours, they use a "Weighted Shortest Elapsed Time First" algorithm, which they prove to be 2-competitive in the non-clairvoyant setting.

Interestingly, Epstein and Levin (2016) show that allowing for preemption in a schedule is at most $\frac{e}{e-1}$ better than the best schedule without preemption.

For the case of weighted jobs with release time but without deadlines, and where preemption is allowed, Sitters (2010) provide an algorithm with a competitive ration of 1.57. Note that their algorithm is very simple and does not make use of Round Robin.

Some online algorithm make use of Linear Programming to find good schedules. For example, Keyvanshokooh et al. (2021) use a primal-dual approach to find a good schedule in the case where the jobs have a release time, a deadline and a processing time.

1.1 Urgency Scores

Let us define the urgency of a job i at time step t as $u_i(t)$.

There exists a laaarge number of urgency scores in the literature, let us list a few of them here such that we can compare them and chose a good score for our algorithm. Note that they all simply define $u_i(t)$ differently.

- **Earliest Due Date** (EDD)

$$u_i(t) = d_i - t \tag{1}$$

  or even without t as

$$u_i(t) = d_i \tag{2}$$

  Note that this score minimizes the max lateness (Jackson's rule).
- **Remaining Spare Time** (RST)

$$u_i(t) = d_i - p_i - t \tag{3}$$

  This score is at the core of LLF (Least Laxity First), which is a very standard score in real-time systems.

- **Critical Ratio**

$$u_i(t) = \frac{d_i - t}{p_i} \tag{4}$$

- **Weighted Shortest Processing Time**

$$u_i(t) = \frac{p_i}{w_i} \tag{5}$$

  Which favors short jobs with high weights.

- **Apparent Tardiness Cost** (ATC)

$$u_i(t) = \frac{w_i}{p_i} \cdot \exp\left(-\frac{\max\{0, d_i - t - p_i\}}{k \cdot \overline{p}}\right) \tag{6}$$

  where $\overline{p}$ is the average processing time of all pending jobs and k is a so-called *lookahead parameter*, which is used to balance how much $u_i(t)$ reacts to the slack. In general, $k \in \{2, 3, 4\}$.

  This is very popular metric in the tardiness literature. ATC has been introduced by Vepsalainen and Morton (1987).

  Note that while we assume a setup cost of zero, if this is not the case, ATC has been extended to include setup costs (Lee et al., 1997).

- **ATC + Adaptive** k

  Idea: let k change over time depending on how many jobs are pending/late.
  Example:

$$k(t) = k_0 + \left(1 + \gamma \frac{\#\text{pending}}{|N|}\right) \tag{7}$$

  If system is busy, $k \rightarrow$ higher, which means that the metric lowers its sensitivity to aggressively chase deadlines.

- **Weighted Slack**

$$u_i(t) = \frac{d_i - t - p_i}{w_i} \tag{8}$$

I'm not sure whether related work change p_i whenever a job is processed, but I think decreasing p_i to refer to the remaining processing time implicitly. This would make jobs that are nearly completed much more urgent, and therefore more likely to be scheduled.

2 Own Algorithm

Idea: use some metric to estimate the future jobs

- Running average of all incoming jobs costs. Use some sort of metric like average sum of reward + penalty (without delay) to estimate whether it is worth to delay a job or completely drop it.

Use offline algorithm to compute a relaxed solution in polynomial time.

My algorithm makes use of the following ideas:

- **Round Robin**

  Because so much research has been done in Round Robin, I think that we can re-use a lot of the previous results. Moreover, the Round Robin allows to have a very fair division of all jobs in the schedule.

  Note that the Round Robin algorithm will make the schedule change jobs often, which might nor be the most beautiful algorithm, but I hope quite good.

- **Importance of each job**

  Each job has a different importance, be it w_i or $\hat{w}_i$. Hence, incorporating this weight is crutial. Note that a greedy score such as $\frac{w_i}{p_i}$ could find greedy solution like in the LP relaxation of the knapsack problem on the slides in the lecture.

- **Urgency of each job**
  Because each job can be delayed by a certain amount (we assume that we can delay a job only once we schedule it, if this is not allowed, the problem becomes muuch more complicated, probably even unbounded), we need to define a metric to measure how urgent we need to schedule a job on the specific time step.
  There exists a large number of different metrics to measure the urgency of a job, I provide a list in Section 1.1.

2.1 Job Importance

Let us now define a metric that we can use to measure the importance of a job. As we allow for preemption, we define the imporance scoer for each time step t of a job.

We wish to have the following properties in our metric:

(1) Jobs that have to be completed soon need to have a high importance compared to other. If a job has its deadline in the near future, this part of the equation should be higher than

To this end, we adapt the ATC score to fit our needs. First, let us recall the definition of the ATC score:

$$u_i(t) = \frac{w_i}{p_i} \cdot \exp\left(-\frac{\max\{0, d_i - t - p_i\}}{k \cdot \overline{p}}\right) \tag{9}$$

which can be separated into three parts, namely

$$u_i(t) = W \cdot P \cdot E \tag{10}$$

where W, P, and E denote various factors of the ATC score.

- W represents the weight of the job.
  Ideally, we want W to be high if the job is important, i.e., has a high weight, and low otherwise. In the formal definition of ATC, we have $W = w_i$ which perfectly illustrate this property.
- P represents the processing time of the job.
  Ideally, we want P to be high whenever the job has a low precessing time, as we can complete more shorter jobs in the same amount of time as in a longer job, and therefore expect to have a higher reward.
  In Equation 10, we have $P = \frac{1}{p_i}$, which perfectly illustrate this property.
- E. In the formal definition of ATC, E represents the remaining time of the job to the deadline, which is even devided by $k \cdot \overline{p}$ to introduce some slackness into the metric.
  In Equation 10, we have $E = \exp\left(-\frac{1}{k \cdot \overline{p}} \cdot \max\{0, d_i - t - p_i\}\right)$. While the idea of this metric is to be low while there is a lot of time left to its deadline, and high whenever the deadline approaches — or even has been reached — we believe that this is the job of P. Hence, we will adapt E to fit our needs.

In the ATC score, the main issue is that E and P have the same intention, and therefore biases the schedule to only schedule the shortest jobs near their deadline, but with a quadratic increast/decrease factor. We argue that this is bad and therefore remove E from the equation.

Moreover, in our problem definition, we have a function $f_i(t)$ that denotes the cost of a job being late at time t. We have to include this function into our metric in order to achieve a competitive online schedule that can make use of push-back.

We therefore decide to replace E with a formula that compute the impact of a job, i.e. how much later can we schedule a job before it becomes not worth it anymore. For that, we assume that the algorithm's window is complete and that no new jobs will arrive. While this is obviously not the case in practice, this assumption would allow us to compute the impact of delaying a job, and hence include the delay of a job in our metric. Note that $E = f_i(t - d_i)$ is a valid choice for E. However, it only considers the next step and not the future k steps.

Instead, using the assumption above, we can compute, for each job i in the current window, the impact of delaying it. We believe that computing $C(j) = \sum_{i=t}^{T} f_j(\max(i - d_i, 0)) \cdot \Gamma(i)_j$ where $\Gamma(i)_j$ denotes the number of pending jobs at time i, without counting the current job j. We can make use of the Theorem that shows the maximal viable delay of a job to restrict this sum. Finally, by normalizing $C(j)$ by either (1) the total sum of costs, i.e. $\hat{C} = \sum_{i=1}^{N} C(i)$, but this make the final score way too small, or (2) the maximal/minimal cost, i.e. $C' = \min_{i \in N} C(i)$.

Alternatively, we can compute

$$C(j) = \sum_{i=t}^{T} f_j(\max(i - d_j, 0)) \cdot \frac{\text{Number of pending jobs at time step } i}{\text{Urgency of pending jobs at time step } i} \tag{11}$$

While computing Equation 11 is extremely difficult, we can do it recursively from the latest time step T to the current time step t. Not only can we reuse a lot of the computations, but the Urgency part can be better estimated.

Idea: Use derivative to estimate urgency?

Idea: Clip E to be between 0 and 1?

Idea: we can denote the maximal viable delay of a job j as T_j.

References

Calinescu, G., Davies, S., Khuller, S., and Zhang, S. (2024). Online flexible busy time scheduling on heterogeneous machines. *arXiv preprint arXiv:2402.11109*.

Epstein, L. and Levin, A. (2016). The benefit of preemption for single machine scheduling so as to minimize total weighted completion time. *Operations Research Letters*, 44(6):772–774.

Jäger, S., Sagnol, G., Schmidt genannt Waldschmidt, D., and Warode, P. (2025). Competitive kill-and-restart and preemptive strategies for non-clairvoyant scheduling. *Mathematical Programming*, 210(1):457–509.

Keyvanshokooh, E., Shi, C., and Van Oyen, M. P. (2021). Online advance scheduling with overtime: A primal-dual approach. *Manufacturing & Service Operations Management*, 23(1):246–266.

Kim, J.-H. and Chwa, K.-Y. (2003). Non-clairvoyant scheduling for weighted flow time. *Information processing letters*, 87(1):31–37.

Kleinrock, L. (1964). Analysis of a time-shared processor. *Naval research logistics quarterly*, 11(1):59–73.

Lee, Y. H., Bhaskaran, K., and Pinedo, M. (1997). A heuristic to minimize the total weighted tardiness with sequence-dependent setups. *IIE transactions*, 29(1):45–52.

Motwani, R., Phillips, S., and Torng, E. (1994). Nonclairvoyant scheduling. *Theoretical computer science*, 130(1):17–47.

Sitters, R. (2010). Competitive analysis of preemptive single-machine scheduling. *Operations Research Letters*, 38(6):585–588.

Vepsalainen, A. P. and Morton, T. E. (1987). Priority rules for job shops with weighted tardiness costs. *Management science*, 33(8):1035–1047.