CIS 320: Algorithms October 13, 2021

Lecture 11

Lecturer: Aaron Roth *Scribe: Aaron Roth*

Greedy Algorithms II: Minimum Lateness Scheduling

Last lecture we studied two related "scheduling" problems that we could solve with greedy algorithms: unweighted interval scheduling, and the interval partitioning problem. Our correctness argument followed from a simple inductive proof, arguing informally that the greedy solution always "stays ahead" of the optimal solution. In this lecture, we'll study a related variant of the scheduling problem that also has a very simple greedy solution, but whose analysis requires a more complex "exchange" argument.

The problem again corresponds to scheduling "jobs" on a single resource (computer jobs on a machine, classes in a classroom, conventions in an event space, etc.) But this time there is some flexibility: jobs i have a duration t_i, but not a specific start or end time. Instead, each job i has a *deadline* d_i, and if it is scheduled so that it ends after its deadline, it is considered to be *late*. How late it is depends on how much after its deadline it ends. We want to find a schedule that minimizes the maximum *lateness* across jobs.

Definition 1 (Minimum Lateness Scheduling) *An instance I of the Minimum Lateness Scheduling problem is given by $I = \{(t_i, d_i)\}_{i=1}^n$, a set of n jobs, each associated with a* duration $t_i \in \mathbb{R}_{\geq 0}$ *and a* deadline $d_i \in \mathbb{R}_{\geq 0}$.

Given an instance I, a solution S *consists of a collection of disjoint start and finish times for each job: $S = \{(s(i), f(i))\}_{i=1}^n$ such that $f(i) = s(i) + t_i$. The* lateness *of a job is $l_i = f(i) - d_i$ if $f(i) > d(i)$, and $l_i = 0$ otherwise. The maximum lateness of a solution $L(S) = \max_i l_i$ is the maximum lateness of any job. The goal is to find a solution that* minimizes *the maximum lateness.*

We again need to think about what a "greedy" algorithm should be greedy with respect to. There are a couple of natural choices that do not work:

1. Scheduling jobs in order of increasing length t_i: the idea is to get short jobs out of the way quickly. This ignores the deadlines entirely, so seems unlikely to work. Indeed, consider the instance $I = \{(1, 100), (10, 10)\}$. The short job has a very permissive deadline, so the optimal schedule is $\{(10, 11), (0, 10))\}$, which has a maximum lateness of $L = 0$. The algorithm that orders greedily by duration would instead schedule $\{(0, 1), (1, 11)\}$, which has a maximum lateness of $L = 1$, and is suboptimal.

2. So maybe we should pay attention to how tight the deadlines are for each job. Define the *slack* time for a job i to be $d_i - t_i$ — the amount of time we have until the moment that we are guaranteed to miss the deadline even if we start the job immediately. Another natural idea is to greedily order jobs in order of increasing slack time. But this doesn't work either; consider the instance $I = \{(1, 2), (10, 10)\}$. Ordering by slack time will schedule the 2nd job first, resulting in the solution $S = \{(10, 11), (0, 10)\}$ which has an overall lateness of $L = 11 - 2 = 9$. But scheduling the first job first $S = \{(0, 1), (1, 11)\}$ is better, with an overall lateness of $L = 11 - 10 = 1$.

But here is an idea for a greedy algorithm that will actually work: Schedule jobs in increasing order of their deadlines d_i! ("earliest deadline first"). To think about this algorithm, let us without loss of generality rename the jobs in this sorted order so that $d_1 \leq d_2 \leq \ldots \leq d_n$. Our (incredibly simple) algorithm will be as follows:

Algorithm 1 DeadlineOrderScheduler(I)

Sort the jobs by deadline, and rename them so that $d_1 \leq d_2 \leq \ldots \leq d_n$.
Let $f = 0$.
for $i = 1$ to n **do**
  Assign job i to the interval $s(i) = f$, $f(i) = f + t_i$.
  Let $f = f + t_i$.
end for
Return the solution $S = \{(s(i), f(i))\}_{i=1}^{n}$.

We begin with a couple observations that are so simple we won't bother with formal proofs. Lets say that a solution S has a *gap* if for two jobs i and j that are scheduled adjacently so that i is scheduled first, j is scheduled next, and there is no job in between, $f(i) < s(j)$. The time $s(j) - f(i)$ will be called *idle time*.

1. DeadlineOrderScheduler(I) produces a solution with no idle time. This is by inspection.

2. There is an optimal solution with no idle time — if there is idle time, we can compress the schedule and only decrease the maximum lateness.

The next concept that will be useful is that of an *inversion* — i.e. a violation of our greedy policy:

Definition 2 *Fix a solution S. We say that job i and job j constitute an* inversion *if $s(i) < s(j)$, but $d_j < d_i$.*

Our algorithm produces a schedule with no inversions, but note that if there are multiple jobs with the same deadline, there may be many schedules with no inversions, and we have not specified how our algorithm should select amongst them. But this is ok, because of the following lemma — any of them will do:

Lemma 3 *Any solution S with no inversions and no idle time has the same maximum lateness $L(S)$.*

Proof Two different schedules S, S' that have no inversions can differ only in how they order consecutive sequences of jobs that have the same deadline d. Consider any sequence of k such jobs with the same deadline d. No matter how we permute them, the last one finishes at the same time t and has the same lateness $\max(t - d, 0)$. ■

All that remains to show is that there is some optimal solution O that has no inversions and no idle time. The proof will proceed by an "exchange" or "local surgery" argument. We will start with an optimal solution O. If it has no inversions, we'll be done. Otherwise, we'll modify O to produce another solution O' that remains optimal, *and* has one fewer inversion. If we can do this, we're almost done, because we can simply repeat this operation until we have a solution with no inversions.

Lemma 4 *There exists an optimal schedule that has no inversions and no idle time.*

Proof Start with an optimal schedule O with no idle time. Suppose O has an inversion: a pair of jobs a, b such that $s(a) < s(b)$ but $d_b < d_a$. Then there must be a pair of adjacent jobs in the schedule i, j that also form an inversion. To see this, consider starting at job a, and walking through the scheduled jobs in order until we get to job b. Because $d_b < d_a$, at some point on our walk, we must pass a pair of adjacent jobs i, j in which the deadline decreases: $s(i) < s(j)$ but $d_j < d_i$. This pair (i, j) form an adjacent inversion.

The idea will be to consider the schedule O' that results from swapping the order of job i and job j — i.e. setting $s'(j) = s(i)$, $f'(j) = s'(j) + t_j$, $s'(i) = f'(j)$, and $f'(i) = s'(i) + t_i = s(i) + t_j + t_i$ Observe that after this reordering, we have that $f(j) = s(i) + t_i + t_j = f'(i)$ — i.e. the finishing time of job j under

the reordering is the same as the finishing time of job i before the reordering. This has only improved the maximum lateness of jobs i and j, because by assumption, $d_j < d_i$, so $f'(i) - d_i < f(j) - d_j$. Since i and j were adjacent, it did not effect the lateness of any job scheduled before i or after j. Thus the maximum lateness $L(O') \leq L(O)$, which is what we wanted.

Thus we have a method to produce from any schedule O with inversions, a schedule O' that is only improved, but has one fewer inversions. Since no schedule can have more than $\binom{n}{2}$ inversions, repeating this operation starting from an optimal schedule yields an optimal schedule with no inversions. ■

This is enough for us to conclude that our algorithm is optimal!

Theorem 5 *On any instance I, DeadlineOrderScheduler(I) returns an optimal solution S.*

Proof By construction, our algorithm returns a solution with no inversions and no idle time. By Lemma 4, there exists some such solution with no inversions and no idle time which has optimal maximum Lateness. But by Lemma 3, all such solutions have the same maximum lateness, and hence all must be optimal. ■