CS 3510 Algorithms Last updated: September 26, 2023

Lecture 10: Greedy Scheduling

Lecturer: Sahil Singla *Scribe(s): Joseph Gulian, Allen Chang*

1 Scheduling Problem

In the scheduling problem you have n jobs, of sizes $s_1, s_2, \ldots, s_n > 0$, and weights $\omega_1, \omega_2, \ldots \omega_n > 0$. The goal is to find a permutation σ to minimize the total completion time, where the total completion time is

$$\sum_{i=1}^{n} \left[\left(\sum_{j:\sigma(j)<\sigma(i)} s_j \right) + s_i \right] \omega_i$$

.

This problem has a lot of use in many fields like serving requests as a server, or running a shop, or processor scheduling. Intuitively, we want to weight the "cost" or objective value of each job based on the time it takes to complete each job that occurs before said job (innermost parentheses), the size of the current job (s_i), weighted by the weight of the current job (ω_i). We want to sum over each element in the permutation, which will give us the total objective value of such a permutation. As an example, say you have three jobs $s_1, s_2, and s_3$, and you order them s_3, s_1, s_2. The total completion time will be $s_3 \cdot \omega_3 + (s_3 + s_1) \cdot \omega_1 + (s_3 + s_1 + s_2) \cdot \omega_2$.

2 Greedy Scheduling

In this problem obviously there's some tradeoff between weight and time; for instance if you have a very important job and you wait long to complete it, it will do damage to your objective. Because of this the "greedy" approach is a little weird here because we have two things we're try to optimize. Then let's consider both extremes and optimize each.

Imagine all jobs have the same size, then the optimal approach will be to serve in the order of weights in decreasing order; it will cost more for us to wait on higher-weighted jobs than it will for lower-weighted jobs. Alternatively, imagine we have all jobs with equal weight. In this case, it is better to serve shorter jobs first; the longer jobs don't have to wait as much.

For instance, let's try to look at a simpler version of the problem where there are only two jobs with $s_1 = 0, s_2 = 10$, and both have same weights. ($\omega_1 = \omega_2 = 1$). There are only two permutations: $\omega = (1, 2)$ implies that the cost is $0 \times 1 + (0 + 10) \times 1 = 10$, while $\omega = (2, 1)$ implies that the cost is $10 \times 1 + 10 \times 1 = 20$. Thus the former is the better permutation.

Now, let's consider extremes. What if a job has a large weight, but also takes a long time? On the other end, what if we have a short job with no weight?

To find an optimal strategy, the plan is to assign a value for each job, and serve jobs simply in order of decreasing value. As a reminder for our results before, jobs with larger weight should have larger value, whereas jobs with lower size should have larger value.

2.1 Attempt 1

Let's say we define a value $\omega_i - s_i$, and then we go in decreasing order. This makes sense because we want to order jobs more if the weight is higher, but we want to order jobs less if they take longer time.

We have a proposal that this algorithm will fail with $\omega_1 = 7, s_1 = 5, \omega_2 = 1, s_2 = 0$. We will find $i_1 = 2, i_2 = 1$, so we will order them i_1, i_2 by the approach above. However if we do the math to get the total completion time of both jobs we get $(1, 2) : 7 \times 5 + 1 \times 5 = 40, (2, 1) : 0 \times 1 + 7 \times 5 = 35$. Intuitively, we can see that the value function is nonlinear, and the function proposed by attempt 1 fails immediately with extreme values of w_i, s_i. Although this is a greedy algorithm, it fails.

We also notice that if time starts running faster and so every job takes half the time or if time starts running slow, the ordering shouldn't change.

2.2 Attempt 2

Let us now define the value $\frac{\omega_i}{s_i}$. It turns out that this value function is optimal, but we would like to prove this fact. Now, if the times double, we will still have the same orderings.

We will do a proof by contradiction that this works, and we will do it much like proofs for graphs. Then consider the optimal permutation of jobs, we will show that if it violates the greedy rule, we can further improve it.

Assume there exists some jobs i, j in the optimal order where i is done before j but the greedy ordering suggests the opposite (i.e. $\frac{\omega_i}{s_i} < \frac{\omega_j}{s_j}$). You may want to swap the two jobs and show that it will result in a shorter total completion time; however this becomes hard to argue because there are effects on jobs in the movement.

Now, observe a nice trick: we can assume without loss of generality that $j = i + 1$; that is, j is completed immediately after i. You may wonder why this is allowed to be assumed; imagine the greedy rule was violated on an arbitrary pair (i', j'), this would imply that there is location immediately next to each other where $i \geq i'$ and $j \leq j'$ which violates the greedy rule and $j = i + 1$. If we consider all possible adjacent i, js, then the inequality

$$\frac{w_{k'_i}}{s_{k'_i}} < \frac{w_{k_{i'+1}}}{s_{k_{i'+1}}} < \cdots < \frac{w_{k'_j}}{s_{k'_j}}$$

must be false for some inequality in the expression (otherwise the original assumption does not hold!).

Now we make the observation that if we swap the two jobs immediately next to each other this will result in a smaller total completion time. Any job done before i or after j will not change, so the only jobs that change are jobs i, j, meaning the total change in the objective will be only with respect to the times of those two jobs.

Let's find the time to do the jobs before the swap, minus the time to do the jobs after the swap:

$$= (t + s_i)w_i + (t + s_i + s_j)w_j - (t + s_j)w_j - (t + s_i + s_j)w_i$$

. Now we'll be able to cancel various values out until we get

$$= s_i w_j - s_j w_i$$

which becomes

$$= s_i s_j \left(\frac{w_i}{s_j} - \frac{w_i}{s_i} \right)$$

and we know that $\frac{w_i}{s_j} - \frac{w_i}{s_i}$ is greater than zero, which makes the old time strictly longer/worse than the new time. Thus, we have a contradiction (swapping makes it faster), it must be that the greedy algorithm holds, and we are done.