## Lab 4-2 Greedy: Exchange Arguments

## The Problem: Scheduling to Minimize Lateness

Given a set of jobs specified by the length of time it will take to accomplish the jobs and a deadline for the job to be completed and a single resource to complete those jobs. For instance, jobi requires li units of time to complete and has a deadline of di. You goal is to create a schedule for the jobs, that is an order in which they should run on the resource that will minimize the maximum lateness of the jobs. Lateness is the maximum of {completion time of the job – the deadline for the job, 0}

To keep things simple we will assume that:

* Time begins at 0 all lengths and deadlines are integers
* A task may start at the same time another finishes and it is not considered overlapping the earlier task.
* We will disregard schedules that contain any gaps between jobs and assume that as soon as one job finishes the next job begins. This is not a real restriction since if a schedule did contain any gaps, removing them cannot increase the lateness of any job, and thus the resulting schedule would be at least as good as the original schedule.

Why can removing any gaps from a schedule not increase the lateness of any of the jobs? (one line answer!)

The difference remains the same regardless if we keep the gaps in or not.

Consider the following simple example: Three tasks:

* job1 l1 = 3 d1 = 4 job2 l2 = 1 d2 = 3 job3 l3 = 2 d3 = 5

Two possible schedules are:

* Schedule A: (1,2,3) that has maximum lateness= max{lateness of job1 , lateness of job2, lateness of job3} since under this schedule job1 completes at time 3, job2 completes at time 4, job3 completes at time 6 the maximum lateness for this schedule is max{ 3-4, 4-3, 6-5) = 1
* What is the maximum lateness of Schedule B: (3,2,1) ??

Max{2-5, 3-3, 6-4}= 2

Your goal for this part of the lab is to develop a greedy algorithm that always finds the schedule that   
**minimizes the maximum lateness for a set of jobs.** The jobs are described as above by their lengths and deadlines.

There are at least three greedy approaches to solving this problem. See if you can find three or more!

1. job1 l1 = 3 d1 = 4 job2 l2 = 3 d2 = 3 //same lengths

Max1,2{3-4, 6-3}= 3

Max2,1{3-3, 6-4}= 2 //earlier deadline first

1. job1 l1 = 3 d1 = 3 job2 l2 = 2 d2 = 3 //same deadline

Max1,2{3-3, 5-3}= 2

Max2,1{2-3, 5-3}= 2 //ordering of lengths produces same max lateness

1. job1 l1 = 2 d1 = 3 //d-l=1 job2 l2 = 1 d2 = 3 //d-l=2 //different d - l

Max1,2{2-3, 3-3}= -1

Max2,1{1-3, 3-3}= -2 //

Stop here and brainstorm different possible greedy approaches. For each look for a counterexample to rule it out as a candidate. While you are doing this, look for any insights into why it is not working

**DO NOT LOOK AT THE NEXT PAGE**

The greedy approaches you will evaluate are:

* Create a schedule by sorting the jobs in increasing order of their lengths, call this ***Shortest job first***
  + Find a **counterexample**: specify the jobs and two schedules that show that ***Shortest job first*** does not always result in finding the optimal schedule.
  + Notice that we are only using one of the parameters, the length and ignoring the deadlines. Perhaps this is the problem?
* Create a schedule by sorting the jobs in increasing order of their slack times, that is di-li, call this ***Shortest slack time first***
  + Find a **counterexample**: specify the jobs and two schedules that show that ***Shortest slack time first*** does not always result in finding the optimal schedule.
  + Notice that we are only using one of the parameters, the length and ignoring the deadlines. Perhaps this is the problem?
* Create a schedule by sorting the jobs in increasing order of their deadlines call this ***Earliest deadline first***
  + It is worth trying to find a counterexample to gain some insight into this as a possible solution. You will find you cannot find a counterexample.
  + Write pseudo-code that describes the algorithm.
  + What is the overall complexity of the full algorithm and why?
* 4. Prove that ***Earliest deadline first*** finds an optimal schedule using the following steps

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**You are to submit:**

1. **Scheduling to Minimize Lateness**
2. **A counter example for Shortest Job First.** It must consist of only 2 jobs specified by their duration and deadlines that shows that ***Shortest slack time first*** does not always result in finding the optimal schedule
   1. job1 l1 = 2 d1 = 2
   2. job2 l2 = 1 d2 = 4

Fill in the following table:

|  |  |  |  |
| --- | --- | --- | --- |
| Schedule | Lateness of first job in the schedule | Lateness of second job in the schedule | Maximum of the latenesses |
| ***Shortest Job First schedule goes here*** | 3-2= 1 | 1-4= -3 | 1 |
| Optimal ordering  (j1, j2) | 2-2= 0 | 3-4= -1 | -1 |

1. **A counter example for Shortest Slack Time First.** It must consist of only 2 jobs specified by their duration and deadlines, that is
   1. job1 l1 = 3 d1 = 3 //d-1 = 0
   2. job2 l2 = 1 d2 = 2 //d-1 = 1

Fill in the following table:

|  |  |  |  |
| --- | --- | --- | --- |
| Schedule | Lateness of first job in the schedule | Lateness of second job in the schedule | Maximum of the latenesses |
| ***Shortest Slack Time schedule goes here*** | 3-3=0 | 4-2=2 | 2 |
| Optimal Ordering  (j2, j1) | 1-2=-1 | 4-3=1 | 1 |

1. A proof that ***Earliest deadline first*** finds an optimal schedule using the basically the same steps used in proof **Largest weight to length ratio first** produces an optimal schedule for the Minimize total weighted time to completion problem
   1. Any additional assumptions you need to make ( usually you wait to see what you need for the proof and add as needed as long as they do not change the generality of what you are trying to prove,
   2. Set up the notation for the optimal and greedy schedules
   3. Argue that if greedy is not an optimal schedule (does not minimize the maximum lateness) then two jobs in any optimal schedule must be ***out of order***.
   4. Swap the jobs and show that the **maximum lateness of the new schedule cannot be worst than the optimal schedule**. This is different (!!) in that in the TWTC problem we could show that objective function decreased and thus was better. Thus resulting in a contradiction
   5. So now life is difficult since we cannot just conclude we have a contradiction. To reach the conclusion, you need to continue swapping jobs until you actually reach the greedy solution and show that it has the same maximum lateness
   6. Hence the greedy solution is an optimal solution!!

Submit a Proof using the above steps. (Note it is possible to make assumptions about the jobs that would allow you to follow the proof used for TWTC more closely, but you are **not** allowed to do that here. The point it to expose you to a different style of proof that must be used in some situations.

Proof of correctness: Earliest deadline algorithm finds the minimum lateness of a set of jobs.

Assume that there is an ordering ∑\* that is the optimal solution but is different from ∑ the ordering given by the greedy algorithm.

If ∑\* is not ∑, it must have two consecutive jobs: i, j where j has the earlier deadline but i is before j in ∑\*. Swapping the two jobs does not affect the max lateness of the jobs before and after the two jobs i and j. Since we know that j has the earlier deadline, then di > dj. Let C be the completion time of the jobs before i. Then, before the swap our max lateness would look like: max{Lbefore i , (C + li ) – di , (C + li + lj) – dj , Lafter j } and the max lateness after the swap would look like: max{Lbefore j , (C + lj ) – dj , (C + lj + li) – di , Lafter i }. Let’s compare the max before the swap and after the swap without the lateness before i and after j since these are just constant values. Thus, we have max­before {(C + li ) – di , (C + li + lj) – dj } which would result in the max lateness being (C + li + lj) – dj­ since dj­­ < di. Now, we compare this max lateness value to that of maxafter {(C + lj ) – dj , (C + lj + li) – di}. First, (C + li + lj) – dj­ must be greater than (C + lj ) – dj , since their difference results in li which is greater than zero. Then, (C + li + lj) – dj is also greater than (C + lj + li) – di , since their difference results in di – dj which is greater than zero. Thus, since the maxbefore is greater than the possible maximums for maxafter, the new swapped schedule cannot be worse than the optimal schedule. Now we know that every time we swap two consecutive jobs out of order, the max lateness is lowered. These steps are repeatable for all two consecutive jobs that are out of order (di > dj). Therefore, we can continue swapping in ∑\*, until no more pairs are out of order. Then, we will get that ∑\* is equal to ∑ since both will be ordered from earliest deadline and therefore have the same max lateness that is minimized.

1. **Minimize Time to Completion:** Given n jobs j1 to jn with duration times d1 to dn , that need to be run on a single processor. What is the best way to schedule the jobs to minimize the time to completion. Time to completion here is defined to be the same as in the Total Weighted Time to Completion problem covered in the screencasts.

1. Specify the algorithm. Clearly describe how your algorithm would order the jobs.

Order the jobs in increasing order of duration times with shortest duration time being the first job in our ordered list. This should minimize the time to completion for the set of jobs.

Ex| job1: d1 = 3 job2 = d2 = 5

TTC for {1,2} = 3 + 8 = 11

TTC for {2,1} = 5 + 8 = 13

1. Prove this algorithm always finds an optimal solution.

Proof of correctness: Shortest duration algorithm finds the minimum time to completion of a set of jobs.

Assume that there is an ordering ∑\* that is the optimal solution but is different from ∑the ordering given by the greedy algorithm. If the greedy solution is not the optimal solution, the optimal solution must be out of order. If ∑\* is not ∑, it must have two consecutive jobs: i, j where di > dj and i is before j in ∑\* but i has the longer duration. Let C be the completion time of the jobs before i. Then, before swapping i and j, our CT would be: jobi = C + di and jobj­ = C + di + dj. Then after the swap the CT would be: jobi = C + di + dj and jobj­ = C + dj. Now if we take the difference between the TTC of the two jobs before and after the swap we get di - dj which is greater than 0. Thus, the TTC after the swap was smaller than the TTC before the swap. Therefore, ∑\* cannot be the optimal solution since it did not produce the set of jobs with a minimized time to completion. Thus, after the swap we obtain the solution given by the greedy solution, which is a job ordering or shortest duration. Therefore, the after ordering given by the greedy solution produces the minimum time to completion for a set of jobs.

1. Give the algorithm’s asymptotic complexity.

Sort in increasing duration: O(n log n) using quick sort