CSC373 Summer '22
Tutorial 2 Solutions

May 26, 2022

Q1 Interval Scheduling on m Machines (Clue for A1Q3!)

Let us recall the interval scheduling problem from class. We are given n jobs, where each job $I_j = [s_j, f_j)$ is an interval. Two jobs are *compatible* if their intervals have empty intersection. In class, we saw how to schedule a maximum number of mutually compatible jobs on one machine: consider the jobs one-by-one in an increasing order of their finish time (EFT), and greedily pick the job being considered if it is compatible with the ones picked so far.

Now, suppose that we have m machines available instead of just one. A feasible schedule can be thought of as a mapping $\sigma : \{1, \ldots, n\} \to \{0, 1, \ldots, m\}$, where job I_j is assigned to machine k if $\sigma(j) = k > 0$, and unassigned if $\sigma(j) = 0$. Jobs scheduled on any single machine must still be mutually compatible. Subject to that, we want to maximize the number of jobs scheduled, i.e., $|\{j \in \{1, \ldots, n\} : \sigma(j) > 0\}|$.

(a) Consider the following Earliest Start Time (EST) algorithm.

Algorithm 1: m-ISP-EST

```
Sort the jobs in an increasing order of their start time so that s_1 ≤ ... ≤ s_n
for j = 1, ..., n do
    if there is a machine k such that all jobs assigned to it are compatible with I_j then
        Assign job I_j to any such machine k
    else
        Let job I_j remain unscheduled
    end
end
```

Consider the following attempt to prove optimality of this algorithm.

1. Consider any job I_j that the algorithm does not schedule.
2. At that point, each machine must have a job scheduled that conflicts with I_j.
3. As seen in class, all m conflicting jobs must have start time earlier than s_j (due to the EST order) and finish time after s_j (since they conflict with I_j).
4. Hence, there are at least $m + 1$ jobs that all contain time s_j in their intervals.
5. Since there are only m machines, even the optimal algorithm must drop at least one of them.
6. The optimal algorithm drops a job for every job dropped by our EST algorithm. Hence, our EST algorithm schedules the maximum number of jobs.

Which step(s) of the argument above are flawed, and why?

(Solution to (a)) The sixth step is flawed. We would need to show that for every job dropped by the EST algorithm, the optimal solution drops a *distinct* job. Otherwise, the optimal solution may drop one job that conflicts with two or more jobs dropped by the EST algorithm, thus leading to more jobs scheduled.

(b) Next, consider the Earliest Finish Time (EFT) algorithm.

Algorithm 2: m-ISP-EFT

```
Sort the jobs in an increasing order of their finish time so that f_1 ≤ ... ≤ f_n
for j = 1, ..., n do
    if there is a machine k such that all jobs assigned to it are compatible with I_j then
        Assign job I_j to any such machine k
    else
        Let job I_j remain unscheduled
    end
end
```

Prove that this algorithm does not always yield an optimal solution by producing a counterexample.

(Solution to (b)) Consider an instance with $m = 2$ and four jobs: $I_1 = [1, 3)$, $I_2 = [2, 4)$, $I_3 = [4, 5)$, and $I_4 = [3, 6)$. Note that these are sorted by EFT. The EFT algorithm schedules the first two jobs on two different machines. In the third round, it has a choice to schedule I_3 on either machine. If it ends up scheduling it on the same machine as I_1, then I_4 will have to be dropped. This would be suboptimal because scheduling all four jobs — I_2 and I_3 on one machine and I_1 and I_4 on the other — is feasible.

Q2 Cops and Robbers

You are given an array $A[1, \ldots, n]$ with the following specifications:

- Each element $A[i]$ is either a cop ('c') or a robber ('r').
- A cop can catch any robber who is within a distance of K units from the cop, but each cop can catch at most one robber.

Write an algorithm to find the maximum number of robbers that can be caught.

Solution to Q2

Algorithm 3: Cops-and-Robbers

```
for c = 1, ..., n do
    if A[c] is a cop then
        Find the smallest index r in [c − K, c + K] such that A[r] is a robber who is not
         currently assigned to any cop
        If such an index r exists, assign cop A[c] to catch robber A[r], else let cop A[c] be
         unassigned
    end
end
```

Correctness: Suppose for contradiction that this algorithm is not optimal. Let G denote the greedy assignment it produces. Let OPT be an optimal solution that matches G for as many iterations as possible. That is, for some k, OPT assigns the first k cops exactly the same way as G does, and there is no optimal solution which matches G on the first $k+1$ cops. Because G is not optimal, k is less than the number of cops.
We derive a contradiction by proving that there is an optimal solution OPT' which matches G on the first $k+1$ cops. Let c denote the index of the $k+1^{\text{th}}$ cop. We consider the following cases:

1. Suppose G leaves cop $A[c]$ unassigned, while OPT assigns cop $A[c]$ to robber $A[r]$. In this case, $r \in [c-K, c+K]$. However, since OPT matches G on the first k cops, $A[r]$ must not be assigned to the first k cops under G. This is a contradiction as the greedy algorithm would not have left cop $A[c]$ be unassigned in that case.

2. Suppose G assigns cop $A[c]$ to some robber $A[r]$, while OPT leaves cop $A[c]$ unassigned. If $A[r]$ is not assigned to any cop under OPT, then we have a contradiction as assigning $A[c]$ to $A[r]$ would increase the quality of the solution. Hence, $A[r]$ must be assigned to some other cop $A[c']$ under OPT. In this case, we can create OPT' by assigning $A[c]$ to $A[r]$ and leaving $A[c']$ unassigned.

3. Suppose G assigns cop $A[c]$ to some robber $A[r]$, while OPT assigns cop $A[c]$ to some other robber $A[r']$. If $A[r]$ is not assigned to any cop under OPT, then we can create OPT' by switching the assignment of $A[c]$ from $A[r']$ to $A[r]$. If $A[r]$ is assigned to some cop $A[c']$ under OPT, then we create OPT' by switching the assignments of $A[c]$ and $A[c']$, i.e., by assigning $A[c] \to A[r]$ and $A[c'] \to A[r']$. For this to work, we need to argue that $A[c'] \to A[r']$ is a valid assignment, i.e., that $r' \in [c'-K, c'+K]$.

Because OPT and G match on the assignment of the first k cops, we have $c' > c$. Further, because G greedily assigns cop $A[c]$ to the first feasible robber not assigned to any previous cops, we have $r' > r$. Now, because OPT assigns $A[c']$ to $A[r]$, we have $c' - K \leq r \leq r'$. On the other hand, because OPT also assigns $A[c]$ to $A[r']$, we also have $r' \leq c + K \leq c' + K$. This completes the proof of validity of $A[c'] \rightarrow A[r']$ assignment.

Running Time: The above algorithm clearly runs in time $O(nK)$ because for each of $O(n)$ cops, it searches for the first available feasible robber in $O(K)$ time. However, the running time can be reduced to $O(n)$ by keeping track of two indices c and r for the next available cop and robber, and increasing both when a match is made or the minimum when they are too far, as the algorithm below shows.

Algorithm 4: Cops-and-Robbers

```
Initialize c and r to the indices of the first cop and the first robber, respectively
while c and r are not null do
    if |c − r| ≤ K then
        Assign cop c to catch robber r
        Increment both c and r to the indices of the next cop and robber, respectively
    else if c < r then
        Increment c to the index of the next cop
    else
        Increment r to the index of the next robber
    end
end
```