Week 6: Greedy

Agenda:

- Job Scheduling on Multi-processors
- Activity Selection

Reading: 371-384

Example 2: Job Scheduling on multi-processors

- Suppose we have a set T of n jobs
- Job i has start time s_i and finish time f_i
- each job can be performed on a machine, but each machine can do one job at a time
- Jobs i and j are said to conflict if their periods overlap, i.e. $s_i < f_j \leq f_i$ or $s_j < f_i \leq f_j$.
- Goal: schedule the jobs on minimum number of machines so that no two conflicting jobs are scheduled on the same machine.
- Greedy idea: Consider the jobs as arriving by their start time. Each jobs that arrives is supposed to be assigned to a machine.

  If you can assign it to a previously used machine, do so. Else allocate a new machine.
- This implicitly suggests the order in which we should consider the jobs: order in non-decreasing order of start time. So

$$s_1 \leq s_2 \leq \ldots \leq s_n$$

Job Scheduling on multi-processors (cont'd)

- Here is the pseudocode:

```
Procedure Multi-Job-Schedule ($T$)

$m \leftarrow 1$
While $T \neq \emptyset$ do
    extract job $i$ with smallest $s_i$ from $T$
    If $i$ has no conflict with the last job on machine $j$ for
                some $1 \leq j \leq m$ then
        schedule $i$ on machine $j$
    else
        $m \leftarrow m + 1$
        Schedule $i$ on machine $m$
```

- Time:
  - $O(n \log n)$ to sort the jobs or build a heap and n calls to extract-max (again total of $O(n \log n)$).
  - How to find a free machine?
  - Just need to keep the finish time of the last job on each machine.
  - Build a PQ (min-heap) with this key; if the smallest value is $\leq s_i$ then schedule job i on that machine. Else take a new machine.
  - Each time a new job is added update the PQ (with increase-key method)
  - So each iteration takes $O(\log n) \longrightarrow$
  - total will be $O(n \log n)$

Correctness of the algorithm:

- All jobs on a machine are non-conflicting by the way we schedule them
- Need to prove that we use minimum number of machines to complete the proof of correctness.
- By way of contradiction, suppose the optimal solution uses k machines and the greedy uses $k' > k$ machines
- Consider the first time that the greedy algorithm schedules some job i on machine $k + 1$.
- So at this point of time job i is conflicting with all the jobs currently on machines $1, \ldots, k$.
- That is, all these jobs start before s_i (because we ordered them based on s_i) and they all finish after s_i.
- Thus, all these k jobs, plus job i, are conflicting $\longrightarrow$ no two of them can be scheduled on one machine
- So optimal needs at least $k+1$ machines too, a contradiction.

Example 3: Activity Selection

- Again, suppose that we have n jobs, each with a start time s_i and finish time f_i; $0 \leq s_i < f_i$.

- We have only one machine to use

- Two jobs i and j are said to conflict (or are incompatible) if the two intervals $[s_i, f_i)$ and $[s_j, f_j)$ overlap.

- Goal: schedule a largest subset of non-conflicting jobs on this machine

- 1st attempt: as in the previous example sort the jobs based on start time, i.e. $s_1 \leq s_2 \leq \ldots \leq s_n$. Schedule a job if we can.

  – Bad example: $s_1 = 1$, $s_2 = 2$, $s_3 = 3$
    $f_1 = 4$, $f_2 = 3$, $f_3 = 4$
    The greedy can only schedule job 1 but the optimal is to schedule 2 and 3.

- 2nd attempt: Try to schedule shorter jobs first, i.e. sort jobs based on $f_i - s_i$ in non-decreasing order and then Schedule a job if we can.

  – Bad example: $s_1 = 3$, $s_2 = 1$, $s_4 = 4$
    $f_1 = 5$, $f_2 = 4$, $f_3 = 8$
    The greedy can only schedule job 1 but the optimal is to schedule 2 and 3.

- 3rd attempt: Sort the jobs based in non-decreasing order of finish time, i.e. $f_1 \leq f_2 \leq \ldots \leq f_n$ and then Schedule a job if we can.

Correctness:

```
procedure Activity-Selection ($S$)

    Sort activities in $S$ s.t.  $f_1 \leq f_2 \leq \ldots \leq f_n$
    $A \leftarrow \emptyset$; $e \leftarrow 0$;
    for $i \leftarrow 1$ to $n$ do
        if $s_i \geq e$ then
            $A \leftarrow A \cup \{i\}$
            $e \leftarrow f_i$
    return $A$
```

- In this algorithm A is the set of jobs scheduled by our algorithm and e at any given time is the time at which the last scheduled job finishes (i.e. the earliest time we can schedule the next job).
- We prove that this 3rd attempt is correct.
- First note that, since we always schedule a new job if its start time s_i is larger than the finish time of the last scheduled job we get a set of compatible jobs at the end.
- To complete correctness, we have to show our algorithm obtains an optimum solution.
- Promising: we say a schedule after step i is promising if it can be extended to an optimal schedule using a subset of jobs in $\{i+1, \ldots, n\}$.
- We prove that after every step i, the partial solution we have is promising.

- Formally, let A_i and e_i denote the values of A and e after iteration i of the for-loop, respectively.

- Note that $A_0 = \emptyset$ and $e = 0$.

- We show that after each iteration i, the decisions we have made so far are all correct in the sense that there is an optimum solution which contains everything we have selected so far and everything that we have decided not to include (so far) does not belong to that optimum either:

  Lemma: For every $0 \leq i \leq n$, there is an optimal solution A_{opt} such that $A_i \subseteq A_{opt} \subseteq A_i \cup \{i+1, \ldots, n\}$.

- Note that if we prove this for all $0 \leq i \leq n$, and in particular for $i = n$ we have: $A_n \subseteq A_{opt} \subseteq A_n \cup \emptyset$ which implies $A_n = A_{opt}$, i.e. our solution is the same as some optimum solution; this is what we wanted.

- We use induction on i to prove the above lemma.

- Base: $i = 0$, clearly empty schedule can be extended to an optimal one from $\{1, \ldots, n\}$; Thus $\emptyset = A_0 \subseteq A_{opt} \subseteq \emptyset \cup \{1, \ldots, n\}$.

- Induction Step: Suppose we have a promising schedule after step $i \geq$, i.e. there is an optimum solution A_{opt} such that: $A_i \subseteq A_{opt} \subseteq A_i \cup \{i+1, \ldots, n\}$.

- We consider three cases

- **Case 1:** if $s_{i+1} < e_i$ so we cannot schedule job $i+1$ because it overlaps with one of the previously scheduled ones; i.e. $A_{i+1} = A_i$.

Since $A_i \subseteq A_{opt}$, A_{opt} has the job from A_i that is overlapping with $i+1$ as well, so A_{opt} cannot have $i+1$ either. Thus: $A_{i+1} = A_i \subseteq A_{opt} \subseteq A_{i+1} \cup \{i+2, \ldots, n\}$.

- **Case 2:** $s_{i+1} \geq e_i$; so $A_{i+1} = A_i \cup \{i+1\}$.

  Case 2A: $i+1 \in A_{opt}$ then $A_{i+1} \subseteq A_{opt} \subseteq A_{i+1} \cup \{i+2, \ldots, n\}$ and we are done.

  Case 2B: $i+1 \notin A_{opt}$;

  - Note that $s_{i+1} \geq e_i$ and so $i+1$ does not conflict with anything in A_i;
  - On the other hand we cannot add $i+1$ to A_{opt} (otherwise it would have not been an optimum solution); so there must be a job $j \in A_{opt}$ conflicting with $i+1$. Since $A_{opt} \subseteq A_i \cup \{i+1, \ldots, n\}$ and since $i+1$ does not conflict with anything in A_i: $j \geq i+2$.
  - Let $j \geq i+2$ be a job of A_{opt} with the earliest finish time that is conflicting with $i+1$.
  - All activities in $A_{opt} - A_i - \{j\}$ have start-time after f_j (or they will overlap with job j); so they do not overlap with job $i+1$ because $i+1$ finishes before f_j.
  - So $A'_{opt} = (A_{opt} - \{j\}) \cup \{i+1\}$ is feasible and has the same size as A_{opt}; i.e. it is another optimum solution and $A_{i+1} \subseteq A'_{opt} \subseteq A_{i+1} \cup \{i+2, \ldots, n\}$.