CS420+500: Advanced Algorithm Design and Analysis

Lectures: May 17, 2017

Prof. Will Evans *Scribe: Siang Lim*

In this lecture we:

- Minimum vertex cover and matching vertex cover;
- List scheduling approximation [Graham, 1966];

Suggested Reading: Jeff Erickson's notes on Approximation Algorithms

1 Minimum Vertex Cover

1.1 Definition

Given an undirected graph $G = (V, E)$, find the smallest set of vertices $S \subseteq V$ such that all edges in G have at least one endpoint in S.

We discussed the difference between a decision problem and an optimization problem:

- Decision Problem: "Do we have a vertex cover of size k?". Answer: yes/no.
- Optimization Problem: "What is the smallest vertex cover?"

1.2 Greedy Algorithm

1. Include in S the vertex, v with the highest degree of connected edges.
2. Remove the vertex v from G and all its incident edges. Repeat until G is empty.

This greedy algorithm guarantees that the size of the greedy vertex cover is no more than $(\log n)\times$ the optimal vertex cover (`OPT VC`) for all graphs.

$$\texttt{GREEDY VC} \leq \log n \cdot \texttt{OPT VC}$$

However, the approximation factor grows as n grows. Next, we will show a better approximation algorithm that seems at first to be worse than `GREEDY VC`. The key question is: For any given graph, how do we know what the optimal vertex cover is without solving for it? Remark: "think about what is making `OPT VC` large". We will demonstrate this concept in the next section.

2 Matching Vertex Cover [MVC]

2.1 Algorithm

1. $S = \emptyset$
2. Pick any edge $(u, v) \in G$
3. Remove (u, v) from G and all edges that have either u or v as an endpoint.
4. Put both u and v in the set S.
5. Repeat Step 2 and pick edges until graph is empty.

2.2 Proof of MVC

We don't know how big `OPT VC(G)` is, but we can find a lower bound on its size.

Claim MVC is a 2-approximation for VC.

Proof Size of the optimal vertex cover of $|\texttt{OPT VC}(G)| \geq$ number of edges picked by MVC.

1. Edges picked by MVC form a matching, no vertex covers more than one edge in a matching.
2. The number of vertices picked by MVC is 2 times the number of edges picked
3. $|\texttt{MVC}(G)| \leq 2|\texttt{OPT VC}(G)|$

This shows that MVC is a 2-approximation for VC, which is better than a $\log n$-approximation using `GREEDY VC`.

3 List Scheduling Approximation [Graham, 1966]

Given n jobs and m identical machines. Job i must execute uninterruptedly for P_i time units, and each machine can only work on one job at a time.

Find A schedule of jobs that minimizes the overall completion time, the time when the last machine finishes its job.

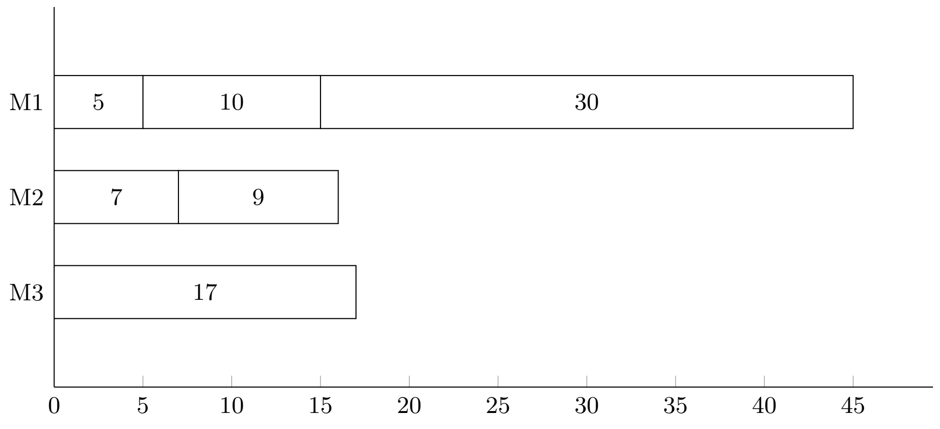
3.1 Greedy Algorithm

Algorithm Whenever a machine becomes idle, assign the next job to it. Repeat.

Benefit Can be an online algorithm because it does not perform sorting and does not need to know future inputs.

Improvements A better solution would be to sort the input jobs in descending order before applying the greedy algorithm.

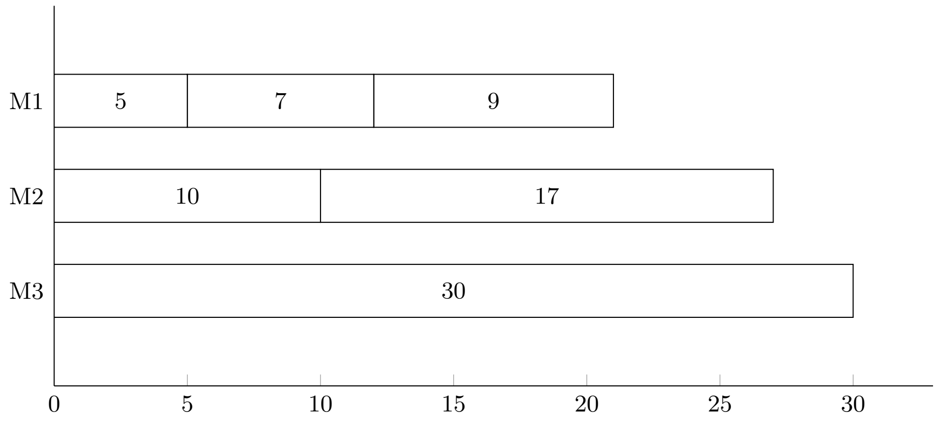
Given an example input of $P_i = [5, 7, 17, 10, 9, 30]$ with three machines M_1, M_2, M_3. The greedy algorithm gives:



The total time needed is 45 minutes with the greedy algorithm.

3.2 Optimal Solution

The optimal solution takes 30 minutes, as shown below:



3.3 Proof

Claim The greedy algorithm is a $\left(2 - \frac{1}{m}\right)$-approximation for list scheduling, slightly better than a 2-approximation.

$$|G(P_1, P_2, \ldots, P_n)| \leq \left(2 - \frac{1}{m}\right) \cdot |\texttt{OPT}(P_1, P_2, \ldots, P_m)|$$

Proof Remarks on proof strategy: (1) Look for some features in the optimal solution that we can use to show that the optimal solution is big, then (2) look for the best possible solution we can hope for. For the list scheduling problem, the best possible solution is that all machines are working at all times with the work divided evenly.

1. $\texttt{OPT} \geq P_i$, for all i because the optimal solution must be at least the size of the smallest job (this is the feature that shows that $\texttt{OPT}$ is big).
2. $\texttt{OPT} \geq \frac{\sum_i P_i}{m}$ because the best case scenario is that all jobs are divided evenly for all machines. (this also shows that $\texttt{OPT}$ is big)
3. Let job k be the last job to finish, and s_k be the start time of job k.
4. We know that P_k is less than $\texttt{OPT}$ from Item 1.
5. We know that all the machines have been working non-stop on jobs up until time s_k because whenever a machine becomes idle we assign a job to it and we've had jobs to assign to machines up until time s_k (in particular, job k has been waiting to be assigned up until time s_k). That means the total work required by jobs other than job k must be greater than (or equal to) the amount of work all m machines can do up to time s_k, which is $m \times s_k$.

   $$\sum_{i \neq k} P_i \geq S_k \times m$$

   Rearrange to get:

   $$S_k \leq \frac{\sum_{i \neq k} P_i}{m}$$

Combine both terms for the last job to complete our proof:

$$\begin{aligned} S_k + P_k &\leq \frac{1}{m} \sum_{i \neq k} P_i + P_k \\ &\leq \frac{1}{m} \sum_{i} P_i + \left(1 - \frac{1}{m}\right) P_k \\ &\leq \texttt{OPT} + \left(1 - \frac{1}{m}\right) \texttt{OPT} \\ &\leq \left(2 - \frac{1}{m}\right) \texttt{OPT} \end{aligned} \tag{1}$$