Lecture 11: Approximation Algorithms: Vertex Cover, Job Scheduling

In the last lecture, we talked about computational complexity and a formal approach to proving intractability. In particular, we talked about the classes $P$, $NP$ and $NP$-Complete problems.

**Question:** What do we do if the problem we wish to solve is in $NP$-complete?

**Answer:** There are 3 basic approaches:

- *Exploit special problem structure:* perhaps we do not need to solve the *general* case of the problem but rather a tractable special version;
- *Heuristics:* procedures that tend to give reasonable estimates but for which no proven guarantees exist;
- *Approximation algorithms:* procedures which are proven to give solutions within a factor of optimum.

In this class, we focus on the last approach.

**Definition:** An algorithm is a factor $\alpha$ approximation for a problem if and only if for every instance of the problem it can find a solution within factor $\alpha$ of the optimum solution.

Let us demonstrate this for the minimum vertex cover problem.

**Example:** *(Minimum Vertex Cover)* Given a graph $G(V,E)$, find a subset $S \subseteq V$ with minimum cardinality such that every edge in $E$ has at least one endpoint in $S$.

**Algorithm 1 2-Approximation for Vertex Cover**

1. **Find a maximal matching $M$ in $G$.**
2. **Output the endpoints of edges in $M$.**

**Claim 1** *The solution of the previous algorithm is feasible.*

**Proof:** We prove this by contradiction: suppose there exists an edge $e = \{v,u\}$ such that neither $u$ or $v$ is covered by the solution of our algorithm. Since $e$ does not share an endpoint with any of the vertices in $M$, $M \cup \{e\}$ is a larger matching, which contradicts with $M$ being a maximal matching.

**Lemma 1** *The cardinality of the solution of the previous algorithm is at most twice the optimum for all maximal matchings $M$.*

**Proof:** Edges of $M$ are independent, thus we need to take at least one vertex from every edge in $M$. This means that $|M| \leq OPT$. By the previous claim, $2|M|$ is a feasible solution, meaning that

$$|M| \leq OPT \leq 2|M|$$

As you can see, we did not compare the solution of our algorithm directly with the optimum solution. Instead, we compared it with the size of a maximal matching, which was a *lower bound* for the optimum solution. This is very typical for analyzing approximation algorithms. If the optimization problem is a
minimization, the solution of our algorithm may be bigger than optimum. For the analysis, we find a lower bound close to the optimum solution and compare our solution with that lower bound. For maximization problems, naturally, the algorithm may find a solution smaller than optimum and we compare the solution with an upper bound. Often, finding a proper lower or upper bound is as hard as finding the algorithm itself.

Our second example is on job scheduling.

**Example: (Job Scheduling Problem)**

Suppose we have $m$ identical machines and $n$ jobs. For each job $i$, we are given $t_i$, the time it takes to process it on one of the machines. Our goal is to assign the jobs to machines in a balanced way.

Let $A_j$ be the set of jobs assigned to machine $j$. Define $T_j = \sum_{i \in A_j} t_i$ to be the load of machine $j$. The *makespan* of an assignment is the maximum load on a machine (i.e. $\max_i T_i$). The goal of *load balancing* is to find an assignment of jobs to machines that minimizes the makespan.

A “greedy” approach, Algorithm 2 is to iteratively assign each job to the machine with the smallest load.

**Algorithm 2 Greedy**

\[
\forall j, \ A_j \leftarrow \emptyset, \ T_j \leftarrow 0 \\
\text{for } i = 1 \text{ to } n \text{ do} \\
\quad j \leftarrow \text{argmin}_k T_k \\
\quad A_j = A_j \cup \{i\} \\
\quad T_j = T_j + t_i \\
\text{end for}
\]

**Theorem 1 (Graham, 1966)** *Greedy scheduling is a 2-approximation for the minimum makespan problem.*

To prove this result, we need to find a few lower bounds for the *optimal solution* $T^*$.

**Lemma 2** The optimal makespan $T^* \geq \max_i t_i$.

**Proof:** The most time consuming job must be assigned to some machine.

**Lemma 3** The optimal makespan $T^* \geq \frac{1}{m} \sum_{i}^n t_i$.

**Proof:** The maximum load must be larger than the average load.

**Lemma 4** The solution of the greedy makespan algorithm is at most

\[
\frac{1}{m} \sum_{i}^n t_i + \max_i t_i
\]

**Proof:** Consider machine $j$ with maximum load $T_j$. Let $i$ be the last job scheduled on machine $j$. When $i$ was scheduled, $j$ had the smallest load, so $j$ must have had smaller than the average load. We have,

\[
T_j = (T_j - t_j) + t_j \leq \frac{1}{m} \sum_{i}^n t_i + \max_i t_i .
\]

Therefore the total load on machine $j$ is less than the average load plus the largest job.
Combining these three lemmas implies Theorem 1. Is this analysis tight? The following example shows that it essentially is.

**Example:** Consider $m$ machines, with $m(m - 1)$ jobs of length 1 and one job of length $m$. The optimal solution is to assign the largest jobs to one machine, and $m$ of the small jobs to each of the remaining $m - 1$ machines, resulting in an optimal makespan of $m$. The greedy algorithm may assign the largest job last, at which point each machine has load $m - 1$, making a makespan of $2m - 1$.

Now, consider a slightly modified greedy approach. First, we sort the jobs so that $t_1 \geq t_2 \geq \cdots \geq t_n$. Then, assign them iteratively to the machines, using Algorithm 3.

**Algorithm 3** Greedy after sorting

<table>
<thead>
<tr>
<th>∀ $j$, $A_j \leftarrow \emptyset$, $T_j \leftarrow 0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sort the jobs so that $t_1 \geq t_2 \geq \cdots \geq t_n$</td>
</tr>
<tr>
<td>for $i = 1$ to $n$ do</td>
</tr>
<tr>
<td>$j \leftarrow \text{argmin}_k T_k$</td>
</tr>
<tr>
<td>$A_j = A_j \cup {i}$</td>
</tr>
<tr>
<td>$T_j = T_j + t_i$</td>
</tr>
<tr>
<td>end for</td>
</tr>
</tbody>
</table>

**Lemma 5** The approximation factor of the modified greedy algorithm is at most $3/2$.

**Proof:** If there are at most $m$ jobs, the scheduling is optimal since we put each job on its own machine. If there are more than $m$ jobs by the pigeonhole principle, at least one processor must get 2 of the first $m + 1$ jobs. Each of these jobs is at least as big as $t_{m+1}$. Thus, $T^* \geq 2t_{m+1}$.

Consider machine $j$ assigned maximum load $T$ where $j > m$ since otherwise we are done. Let $i$ be the last job assigned to $j$. Since the loads are sorted $t_i \leq t_{m+1} \leq T^*/2$ and as before

$$T = (T - t_i) + t_i \leq \frac{1}{m} \sum_{j=1}^{m} T_j + t_i \leq T^* + T^*/2 = \frac{3}{2} T^*$$

The above analysis is not tight. It is possible to show that the approximation factor of greedy after sorting is at most $4/3$. The proof of $4/3$ factor uses the same ideas as the above proof but it is more involved. We omit it in this lecture note but sketch the main idea: if the last job assigned to the machine with the highest load has index less than $2m$, then the algorithm has found the optimum solution. Otherwise, $T \leq \frac{1}{m} \sum_{j=1}^{m} T_j + t_{2m+1} \leq T^* + T^*/3 = \frac{4}{3} T^*$.

**Lemma 6** The approximation factor of the modified greedy algorithm is $4/3$.

Note that $4/3$ is essentially tight. Consider an instance with $m$ machines, $n = 2m + 1$ jobs, $2m$ jobs of length $m + 1, m + 2, \cdots, 2m - 1$ and one job of length $m$. 