1 Minimum Makespan Revisited

Recall the minimum makespan problem. We have a set $J$ of jobs to schedule on a set $M$ of machines and we want to minimize the makespan, the maximum load on a machine. In the previous lecture, we assumed that each job has the same processing time on all machines. Here we consider a more general version of the problem in which job $j$ can have different processing time on different machines.

Let $t_{ij}$ be the time it takes machine $j$ to process job $i$. We want to minimize the makespan $T = \max_j \sum_i x_{ij} t_{ij}$ where the variable $x_{ij} \in \{0,1\}$ indicates whether job $i$ is assigned to machine $j$. We can write this as an integer program with constraints ensuring that each job is assigned to exactly one machine and that the load of each machine does not exceed $T$, the makespan.

\[
\begin{align*}
\text{minimize} & \quad T \\
\text{subject to} & \quad \sum_{j \in M} x_{ij} = 1 \quad \forall \ i \in J \\
& \quad \sum_{i \in J} x_{ij} t_{ij} \leq T \quad \forall \ j \in M \\
& \quad x_{ij} \geq 0 \quad \forall \ j \in M \ i \in J \\
\end{align*}
\]

First, one may try to relax variable $x_{ij}$ and let $x_{ij} \in [0,1]$, however, the solution of the corresponding LP may be too far from the solution of the IP. Thus, the solution of the LP does not serve as a tight lower bound for OPT. For instance, if we have only 1 job, $m$ machines, and $t_{1j} = m$, $j \in M$, then OPT= $m$, however, solution of LP is 1 by assigning $x_{1j} = \frac{1}{m}$.

The maximum ratio of the optimal IP and LP solutions is called the integrality gap. We need to define the relaxed problem in such a way that the integrality gap is small.

The idea is to ensure that if $t_{ij} > T$ we assign $x_{ij} = 0$. We do this by defining a series of feasibility LPs with a makespan parameter $T$ as follows. The idea and analysis follow [1].

\[
\begin{align*}
\sum_{j : t_{ij} \leq T} x_{ij} = 1 & \quad \forall \ i \in J \\
\sum_{i \in J} x_{ij} t_{ij} \leq T & \quad \forall \ j \in M \\
x_{ij} \geq 0 & \quad \forall j \in M \ i \in J \\
\end{align*}
\]

Using binary search we can obtain the smallest value of $T$, $T^*$, for which the above feasibility linear program (FLP) has a solution. It is easy to show that in such a solution we assign $x_{ij} = 0$ if $t_{ij} > T^*$.

Claim 1 FLP assigns at most $|J| + |M|$ jobs to the machines.

Proof: Recall that a solution $x$ is a vertex of the polyhedron formed by the constraints. Note that there are $2(|J| + |M|)$ total constraints At a vertex, $|J| + |M|$ linearly independent constraints must be satisfied; at
least \(|J| + |M|\) of them are of the form \(x_{ij} = 0\), therefore the number of non-zero \(x_{ij}\)’s is at most \(|M| + |J|\).

We now proceed to rounding the solution of FLP with a minimum value of \(T\) given by \(T^*\). Let \(G(J, M, E)\) be a bipartite graph defined on the set of jobs and machines where edge \((i, j)\) between job \(i\) and machine \(j\) has weight \(x_{ij}\).

The graph \(G\) has \(|M| + |J|\) vertices and by the above claim there are at most \(|M| + |J|\) edges. Thus if \(G\) were connected, we would need to remove one edge to obtain a tree; roughly speaking, graph \(G\) is “close” to a tree. Note that for each feasible solution, \(\sum_{j: t_{ij} \leq T^*} x_{ij} = 1, i \in J\), and \(\sum_{i \in J} t_{ij} x_{ij} \leq T^*, j \in M\).

We modify the weights \(x\) to obtain \(x'\) in such a way that the resulting graph \(G(J, M, E')\) is a tree and the constraints satisfied. In particular, we may increase the load by \(t_{ij} x_{ij} \leq T^*, j \in M\).

Suppose \(x'\) is given. Note that if node \(i \in J\) is a leaf of the tree with parent \(j \in M\), then \(x_{ij} = 1\), thus there is no \(i\) leaf with fractional weight. However, we can have fractional edge \((i, j)\) between leaf \(j \in M\) and its parent \(i \in J\). Suppose \(i\) has \(k\) leaves \(i_1, i_2, \ldots, i_k\), we choose one of the leaf machines uniformly at random and assign job \(i\) to it. Then we remove \(i\) from the graph. We repeat this procedure of assigning fractional load of a parent to one its children and removing the job from the graph until all the jobs are assigned.

**Claim 2** The above rounding procedure produces a factor 2 approximation.

**Proof:** Let \(OPT\) be the makespan of the optimal assignment and let \(T^*\) be the minimum value of \(T\) found using binary search on FLP. Then, \(T^* \leq OPT\) since the FLP is clearly feasible using the optimal assignment. \(x'\) is a feasible solution therefore load of each machine is at most \(T^*\). During the rounding procedure, we add the load of at most one job to each machine because a node \(j\) can only have one parent in \(G'\). Suppose machine \(j\) is a leaf of job \(i_p\). \(L'_j = \sum_{i \in J} x_{ij} t_{ij} \leq T^*\), and the load of job \(i_p\) is assigned to machine \(j\). Thus the new load of machine \(j\) is less than \(L'_j + t_{ij}\); since \(x_{ip} \neq 0\) we know that \(t_{ip} \leq T^*\), thus \(L'_j + t_{ip} \leq 2T^*\).

Hence, the final makespan is at most \(2T^*\).

The remaining task is to show that we can covert \(x\) to \(x'\) such that the underlying graph become a tree. Suppose \(G = (J, M, E)\) is not a tree, thus it has cycle \(c = j_1, i_1, j_2, i_2, \ldots, j_r, i_r, j_1\); suppose we update \(x_{i_1,j_1}\) to \(x_{i_1,j_1} - \epsilon\), we proceed around cycle \(c\) and update the weight of edges in the following way:

Since the total weight of edges incident to \(i_1\) must add up to one, if we decrease \(x_{i_1,j_1}\) by \(\epsilon\), we need to increase \(x_{i_1,j_2}\) by \(\epsilon\), thus we update \(x_{i_1,j_2}\) to \(x_{i_1,j_2} + \epsilon\). Now to keep the load of machine \(j_2\) less than or equal to \(T^*\), we decrease \(x_{i_2,j_2}\) by \(\epsilon^2 = \frac{t_{i_2,j_2}}{t_{i_2,j_2}}\epsilon\), repeating this procedure, we modify the weights keeping the constraints satisfied. The only constraint that may become unsatisfied is the load of machine \(j_1\); we decrease the load of \(j_1\) by \(\epsilon\) via edge \((i_1, j_1)\) and at the end, we may increase the load by \(\frac{t_{i_1,j_2}}{t_{i_1,j_2}} \frac{t_{i_2,j_2}}{t_{i_2,j_2}} \ldots \frac{t_{i_{r-1},j_r}}{t_{i_{r-1},j_r}}\epsilon\). If \(\frac{t_{i_1,j_2}}{t_{i_1,j_2}} \frac{t_{i_2,j_2}}{t_{i_2,j_2}} \ldots \frac{t_{i_{r-1},j_r}}{t_{i_{r-1},j_r}} > 1\) we use the simple observation that if we start from \((i_r, j_1)\) and go around the cycle in that direction, then we would need to increase the weight of \((i_1, j_1)\) by \(\left(\frac{t_{i_1,j_2}}{t_{i_1,j_2}} \frac{t_{i_2,j_2}}{t_{i_2,j_2}} \ldots \frac{t_{i_{r-1},j_r}}{t_{i_{r-1},j_r}}\right)^{-1} < 1\) thus the total load of \(i_1\) would become less that \(T^*\).

Using the above scheme, we are able to decrease the weight of \((i_1, j_1)\) by \(\epsilon\) keeping the solution feasible. Repeating this reduction, we can make the weight of one of the edges zero, thus we can remove cycle \(c\). We obtain \(x'\) by breaking all the cycles.

### 2 Traveling Salesman Problem

One of the most famous NP-hard problems is the Traveling Salesman Problem, in which we want to find an optimal route for some “traveling salesman” to hit all cities exactly once. More formally, the problem is the
following: given a set of points $V$ and a cost function $c : V \times V \mapsto \mathbb{R}^*$, find a tour that visits every vertex exactly once and has the smallest total cost.

Clearly this problem is NP-hard, as putting all edge weights 1 is equivalent to the Hamiltonian Tour problem. This also shows there is no approximation for the general TSP, since we have no way of knowing whether there is a tour of cost $n$ or $\infty$. We can, however, restrict the TSP problem to make it more interesting by requiring the edge weights to be metric.

**Definition:** (Metric cost functions)

$$
\begin{align*}
    c(u, u) &= 0 & \forall u \\
    c(u, v) &= c(v, u) & \forall u, v \\
    c(u, v) + c(v, w) &\geq c(u, w) & \forall u, v, w
\end{align*}
$$

A 2-approximation for metric TSP (Alg. 2) comes from the following idea. Find a minimum spanning tree in the graph $T$. Clearly, the cost of $T$ is less than any tour, since removing an edge from a tour gives a spanning tree. If we double the edges of $T$ to make a new graph $T'$, we can find an Eulerian circuit in $T'$ which has total cost $2c(T)$. We can then transform this circuit into a Hamiltonian Tour by “shortcutting” the edges; the tour will be defined by the order in which they are first visited in the Eulerian circuit. Because of the metric property, “shortcutting” cannot increase the path length.

(Metric TSP 2-approximation via MST shortcutting)

Find a minimum spanning tree $T \in G$.

Double all the edges in $T$ and obtain an Eulerian circuit.

“Shortcut” the edges in the Eulerian circuit to form a Hamiltonian Tour $P$.

Output $P$.

**Claim 3** MST Shortcut is a 2-approximation algorithm for metric TSP.

**Proof:**

$$
TSP \geq MST = \frac{1}{2} \text{Eulerian Tour} \\
\geq \frac{1}{2} \text{Eulerian Tour with shortcuts}
$$

We can improve this algorithm by noticing that we do not actually need to double the edges of the MST. We want to create some Eulerian tour. So we just need to match all the odd degree vertices in the MST (of which there must be an even number - proving this is a good exercise). Then we find an optimal matching on this set for an Eulerian tour and shortcut as before. We can prove that this matching is no greater than $1/2$ OPT, so we can get a $3/2$ approximation.

For many years this had not been improved though people conjectured the approximation factor should be $4/3$ because that is the integrality gap of the problem. More recently, a breakthrough was shown that the approximation factor is in fact $3/2 - \epsilon$ for some very small $\epsilon$ [2]. Then that bound quickly improved to close to $4/3$ since.
References
