Load Balancing Approximation

1 Load Balancing with a Greedy Algorithm

1.1 The Load Balancing Problem

LOAD BALANCING: Given jobs $j = 1, \ldots, m$ with processing time t_j and n machines. We want to partition $\{1, \ldots, m\}$ into $S_1, \ldots, S_n$ to minimize $\max_i \sum_{j \in S_i} t_j$ (i.e., the largest load on any machine). This problem is **NP**-hard even for $n = 2$ by a reduction from PARTITION.

Lower bounds on OPT:

(1). $\frac{1}{n} \sum_j t_j$ (if the loads are split perfectly evenly)

(2). $\max_j t_j$ (because somebody has to do the largest job)

1.2 Greedy Algorithm

In some order $j = \{1, \ldots, m\}$, give the tasks to the currently least loaded machine.

Let i be the machine most loaded at termination. At some point, a last job j^* was added to i. At that point, i was the least loaded among all machines. Since then, only more jobs could have been added to $i' \neq i$. Thus, at termination:

(3). $\sum_{j \in S_{i'}} t_j \geq (\sum_{j \in S_i} t_j) - t_{j^*}$

1.3 Approximation Bounds

$$\sum_{j \in S_i} \leq t_{j^*} + \sum_{j \in S_i, j \neq j^*} t_j$$

$$\leq^{(2)} OPT + \sum_{j \in S_i, j \neq j^*} t_j$$

$$\leq^{(3)} OPT + \frac{1}{n} \sum_{j \neq j^*} t_j$$

$$\leq^{(1)} OPT + OPT$$

$$= 2OPT$$

Thus the greedy algorithm is a 2-approximation. By processing jobs in order of non-decreasing t_j, we can improve the bound to a $\frac{3}{2}$-approximation.

2 Load Balancing with an ILP

2.1 Designing an ILP

We are also given a bipartite graph $\mathcal{G}$ on jobs $\times$ machines, and machine i can do job j only if there is an edge between them. In the ILP, if there exists an edge $e = (i, j) \in \mathcal{G}$, we have a variable $x_{ij} = t_j$ if the job j is on machine i, and is 0 otherwise. We also have L representing the maximum load on any machine. Our ILP is:

Minimize

$$L$$

Subject to

$$\begin{cases} \sum_i x_{ij} = t_j & \forall j \\ L \geq \sum_j x_{ij} & \forall i \\ x_{ij} \geq 0 & \forall i, j \\ x_{ij} \in \{0, t_j\} & \forall i, j \end{cases}$$

Notice that the integrality gap here is n when there is one job that can run on each of n machines. So, we **strengthen** the LP by adding an extra constraint $L \geq t_j \ \forall j$, which is (2) from earlier. By strengthening an LP, we add a constraint which doesn't change the ILP, but restricts the LP, so the integrality gap decreases. Our LP is:

Minimize

$$L$$

Subject to

$$\begin{cases} \sum_i x_{ij} = t_j & \forall j \\ L \geq \sum_j x_{ij} & \forall i \\ L \geq t_j & \forall j \\ x_{ij} \geq 0 & \forall i, j \end{cases}$$

2.2 Rounding the LP

Our goal is to round $\vec{x}$. Recall that the variable values x_{ij} define a bipartite graph $\mathcal{G}$ on jobs $\times$ machines with $e = (i, j)$ present iff $x_{ij} > 0$. We cannot just round small values down and large values up, because the LP solution might assign small fractions of a job j to each machine, leading to no large x_{ij} values (similar to the integrality gap example earlier). We'll show that by removing the cycles from this bipartite graph, we can easily find our approximate assignment.

Lemma 1. *Given a solution $\vec{x}$, we can compute in polynomial time an $\hat{x}$ with an acyclic bipartite graph and is an equally good solution to the LP.*

Proof. While $\mathcal{G}$ has a cycle, let $C = \{e_1, \ldots, e_{2k}\}$ be such a cycle (with even length because $\mathcal{G}$ is bipartite). Say $e_l = (i_l, j_l)$ and $\mathcal{E} = \min_{l=1,\ldots,2k} x_{i_l j_l}$. For all $2l - 1$, we increase $x_{i_{2l-1} j_{2l-1}}$ by $\mathcal{E}$, and for all $2l$, we decrease $x_{i_{2l} j_{2l}}$ by $\mathcal{E}$. In other words, we alterjnate increasing and decreasing around the cycle, which removes the edge with $x_{ij} = \mathcal{E}$. This leaves all LP constraints unchanged, so the new solution is feasible and uses one less edge. We repeat until there are no cycles left in $\mathcal{G}$. $\square$

Now, the bipartite graph for $\vec{x}$ is a forest. We root each tree at an arbitrary node (diagram in Kleinberg & Tardos page 641). From here, we can assign the leaf jobs to their parent machine, and each internal job to an arbitrary child machine (this is better than assigning them to parents, because then each machine gets at most one extra job).

2.3 Bounding the LP Solution

Consider one machine i which causes the maximum load for our solution. We assigned to i:

1. All t_j for leaf children j of i (total cost $\leq L$)
2. Possibly t_{j^*} for the parent j^* of i (total cost $\leq L$ by the extra constraint we added to the LP)

The LP solution assigned to i:

1. All t_j for leaf children j of i
2. Next to nothing for t_{j^*} for the parent j^* of i (can't be 0 because it was assigned, but can be really close)

Thus, the total cost for our solution of i is at most $2L \leq 2OPT$, so we have a 2-approximation.

2.4 Solving Load Balancing with Network Flow

Given L, deciding if it is possible to balance the load is a flow problem with fractional outcome. Then, we can binary search for the value of the minimum L. This is a more efficient way to solve the LP – we technically don't need to LP to solve load balancing, but it inspired our initial algorithm.