COS 330: Great Ideas in Theoretical Computer Science

Fall 2025

Precept 6

My kind of precept

Learn

In today's Learn section, we'll explore yet another problem and see how a simple greedy algorithm can provide a good approximation. The problem we'll study is known as *Load Balancing*, a classic problem in scheduling and resource allocation.

Input:

- n jobs with processing times $p_1, p_2, \ldots, p_n$
- m identical machines

Goal: Assign each job to exactly one machine to minimize the *makespan* (the maximum total processing time on any machine).

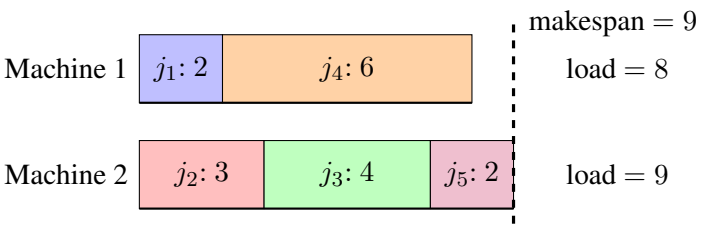
Let's formalize the above. An assignment is a function $A : \{1, \ldots, n\} \to \{1, \ldots, m\}$ where $A(j) = i$ means job j is assigned to machine i. The load on machine i is:

$$L_i = \sum_{j:A(j)=i} p_j$$

The makespan is $\max_{i=1}^{m} L_i$. We want to find an assignment that minimizes the makespan.

Let's consider an example. Suppose we have 5 jobs with processing times $p_1 = 2, p_2 = 3, p_3 = 4, p_4 = 6, p_5 = 2$ and 2 machines. Here is one possible assignment:

- Machine 1 gets jobs 1, 4 (load = 8), Machine 2 gets jobs 2, 3, 5 (load = 9), and so makespan = 9.



Question: Show that the makespan of the above instance is at least 9 without trying all possible assignments. (Try it yourself before looking at the solution).

Answer: The total processing time of all jobs is $2+3+4+6+2 = 17$. Since there are 2 machines, the average load per machine is $17/2 = 8.5$. Thus, at least one machine must have load at least 9 (since loads are integral). Therefore, the makespan is at least 9.

As has been the goal throughout this week, we want to find an efficient algorithm for this problem, but alas Load Balancing is NP-hard (does it remind you of any other problems we've seen in the past?) So we turn to approximation algorithms. Here is possibly the simplest natural greedy algorithm for this problem:

Greedy Load Balancing Algorithm:

1. Initially, all machines have load 0
2. For each job $j = 1, 2, \ldots, n$ (in any order):
    - Assign job j to the machine with the current minimum load
    - Update that machine's load

Intuitively, this algorithm tries to keep the loads balanced by always placing the next job on the least loaded machine, thus delaying the growth of the makespan as much as possible.

Running the greedy algorithm on the previous example (jobs in order 1, 2, 3, 4, 5):

- Job 1 ($p_1 = 2$): Both machines have load 0, assign to Machine 1. Loads: [2, 0]
- Job 2 ($p_2 = 3$): Machine 2 has minimum load, assign to Machine 2. Loads: [2, 3]
- Job 3 ($p_3 = 4$): Machine 1 has minimum load, assign to Machine 1. Loads: [6, 3]
- Job 4 ($p_4 = 6$): Machine 2 has minimum load, assign to Machine 2. Loads: [6, 9]
- Job 5 ($p_5 = 2$): Machine 1 has minimum load, assign to Machine 1. Loads: [8, 9]
- Makespan = 9 (optimal in this case!)

For the above example, the greedy algorithm produced an optimal solution. But how good is it in general? Let's analyze its performance.

Theorem 1. The greedy algorithm is a 2-approximation for Load Balancing.

Proof. Let A denote the makespan produced by the greedy algorithm, and let OPT denote the optimal makespan.

Recall that our strategy for proving approximation guarantees is to compare the solution produced by the algorithm to a lower bound on the optimal solution. We are going to use two different lower bounds on the optimal makespan.

Lower bound 1: The optimal makespan must be at least the average load:

$$\text{OPT} \geq \frac{1}{m} \sum_{\ell=1}^{n} p_\ell$$

This is the same argument we used in the example above, but now generalized to any instance. Note that the total work is $\sum_{\ell=1}^{n} p_\ell$, and it must be distributed among m machines, so at least one machine must have load at least the average.

Lower bound 2: Since the makespan must be at least the size of the largest job, we have:

$$\text{OPT} \geq \max_{1 \leq j \leq n} p_j$$

Observation: Suppose that machine i is the machine whose load is exactly the makespan, i.e., load $L_i = A$. In other words, machine i is the machine with the highest load in the greedy solution. Let job j be the last job assigned to machine i.

Since the algorithm assigns a job to the least loaded machine, it follows that all machines have load at least $A - p_j$, otherwise job j would have been assigned to a machine with load less than $A - p_j$. Therefore, $p_\ell \geq A - p_j$ for all machines $\ell = 1, 2, \ldots, m$. Summing over all m

$$\sum_{\ell=1}^{n} p_\ell \geq m \cdot (A - p_j)$$

Rearranging gives:

$$A - p_j \leq \frac{1}{m} \sum_{\ell=1}^{n} p_\ell$$

Combining these two lower bounds and the observation:

$$\begin{aligned} A &= (A - p_j) + p_j \\ &\leq \frac{1}{m} \sum_{\ell=1}^{n} p_\ell + p_j && \text{(from observation above)} \\ &\leq \text{OPT} + \text{OPT} && \text{(using both lower bounds)} \\ &= 2 \cdot \text{OPT} \end{aligned}$$

Therefore, the greedy algorithm produces a solution with makespan at most $2 \cdot \text{OPT}$, which means it is a 2-approximation algorithm. □

The intuition behind the proof is that the greedy algorithm tries to keep the loads balanced, but if the loads are perfectly balanced, then a new job of size exactly equal to the current makespan could double the makespan. However, we could have rearranged the jobs to avoid this doubling.

Exercise: This analysis is tight. There exist instances where the greedy algorithm produces a solution with makespan exactly $2 \cdot \text{OPT} - \epsilon$ for arbitrarily small $\epsilon > 0$. Can you construct such an instance?

Practice

Problem 1

In class, we saw how rounding the Vertex Cover LP relaxation gives a 2-approximation algorithm. As we discussed, the rounding procedure might change the objective value. So one could ask: how

far can the LP relaxation be from the integral optimum? In other words, can we expect a smarter rounding procedure to give a better approximation ratio than 2?

Recall the Vertex Cover problem: given an undirected graph $G = (V, E)$, find a minimum-size subset $S \subseteq V$ such that every edge has at least one endpoint in S.

The Integer Program (IP) formulation is:

$$\begin{aligned} \text{Minimize:} \quad & \sum_{v \in V} x_v \\ \text{Subject to:} \quad & x_u + x_v \geq 1 \quad \forall (u, v) \in E \\ & x_v \in \{0, 1\} \quad \forall v \in V \end{aligned}$$

The LP relaxation replaces the integrality constraint with $0 \leq x_v \leq 1$ for all $v \in V$.

(a) Consider the triangle graph K_3 with vertices $\{a, b, c\}$ and edges $\{(a, b), (b, c), (a, c)\}$. Find the optimal integral solution (i.e., the size of the minimum vertex cover).

(b) Find an optimal fractional solution to the LP relaxation for K_3. What is its objective value?

The integrality gap of a graph G is the ratio $\frac{\text{OPT}_{\text{IP}(G)}}{\text{OPT}_{\text{LP}(G)}}$. One can think of the integrality gap as a measure of how much worse the integral solution can be compared to the fractional solution, i.e., the question we posed at the start of this problem.

(c) What is the integrality gap for the K_3 instance?

(d) Show that the integrality gap of K_n (the complete graph on n vertices) is $\frac{2(n-1)}{n}$.

So we see that the integrality gap approaches 2 as n grows large. This means that no rounding scheme based solely on this LP relaxation can achieve an approximation ratio better than 2 for Vertex Cover.

Problem 2

In lecture, we saw a 2-approximation algorithm for the unweighted Vertex Cover problem using LP rounding. In this problem, you'll extend the analysis to the weighted version.

Weighted Vertex Cover: Given an undirected graph $G = (V, E)$ where each vertex $v \in V$ has a weight $w_v > 0$, find a vertex cover $S \subseteq V$ that minimizes $\sum_{v \in S} w_v$.

The LP relaxation for Weighted Vertex Cover is:

$$\begin{aligned} \text{Minimize:} \quad & \sum_{v \in V} w_v \cdot x_v \\ \text{Subject to:} \quad & x_u + x_v \geq 1 \quad \forall (u, v) \in E \\ & 0 \leq x_v \leq 1 \quad \forall v \in V \end{aligned}$$

The rounding algorithm is the same as in lecture:

1. Solve the LP relaxation to get x_v^* for all $v \in V$
2. Let $S = \{v \in V : x_v^* \geq 1/2\}$
3. Return S as the vertex cover

(a) Prove that S is a valid vertex cover.

(b) Prove that the total weight of S is at most $2 \cdot \text{OPT}_{\text{LP}}$, where OPT_{LP} is the optimal value of the LP relaxation.

(c) Conclude that this algorithm is a 2-approximation for Weighted Vertex Cover.