Approximation Algorithms

Josh Wilcox (jw14g24)

May 5, 2025

Table of Contents

The need for Approximation Algorithms

A note on NP hard problems

- Some algorithms can be defined as **NP-Hard**, meaning there is no hope for them having an efficient, polynomial, algorithm that solves them **exactly**
- All NP-Hard problems have complexity $\geq \mathcal{O}(2^n)$

"Solving" NP-Hard Problems

- Our best hope to get **a** solution to an NP-hard problem is to make an **Approximation**
  - This approximation must be good enough to be useful
- The way we find a good approximation is finding a solution that is **guaranteed** to be within a certain factor of the optimal solution

Defining Approximation Algorithms

An algorithm is an α-approximation if it returns a solution that is *provably* **within a factor** α of the optimal solution

- For all instances I of a problem that has optimal solution $OPT(I)$:

$$\text{Minimisation Algorithm A is an } \alpha\text{-approximation} \iff OPT(I) \leq A(I) \leq \alpha \cdot OPT(I), \quad \alpha > 1$$

$$\text{Maximisation Algorithm B is an } \alpha\text{-approximation} \iff \alpha \cdot OPT(I) \leq B(I) \leq OPT(I), \quad \alpha < 1$$

Load Balancing

Defining the Problem

- We have n tasks with programming times $t_1, \ldots, t_n$
- We want to distribute the tasks to run on m machines - while ensuring the load is as balanced as posible
- The load of a machine is the sum of processing times assigned to it

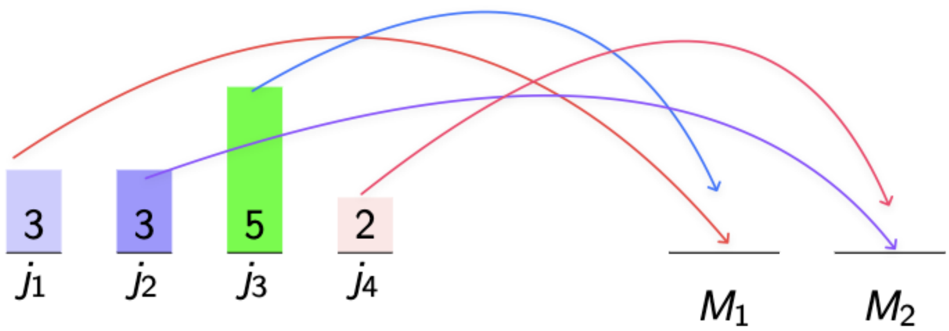
Minimisation Approach

- We want to minimise the **makespan**
  - The makespan is the total load in the set of the machine - the machine with the most load
  - If $L(M_i)$ is the load of machine M_i, the makespan of the problem is:

$$\max_{1 \leq i \leq m} L(M_i)$$

Our Greedy Solution

- Consider tasks in order and assign each task to the machine with the **smallest total load**



Subsections of Load Balancing

- Finding Alpha

Finding Alpha

Finding α

- We want to find α such that:
$$OPT(I) \leq G(I) \leq \alpha \cdot OPT(I) \quad \forall I$$
- This is done by finding a **lower bound** for $OPT(I)$ and seeing how the greedy algorithm compares in the worst case
  - The average load of each machine M is:
  $$\frac{1}{m}\sum_{i=1}^{m} L(M_i) = \frac{1}{m}\sum_{j=1}^{n} t_j$$
  - There will always exist a machine with M_i greater or equal to the average:
  $$\exists L(M_i) \geq \frac{1}{m}\sum_{j=1}^{n} t_j$$
  - Also, the largest tasks is assigned to some machine M_j
  $$\exists M_j \ s.t.\ L(M_j) \geq \max_{1\leq k\leq n} t_k$$
  - Therefore, the smallest the makespan could be is the average or the single largest load - whichever is bigger
  $$OPT(I) \geq max\left\{\frac{1}{m}\sum_{j=1}^{n} t_j, \max_{1\leq k\leq n} t_k\right\}$$
- Now, analyse the greedy algorithm
  - Let J_y be the last task assigned to Machine M_x with processing time t_y
  - Let L^* be the load of M_x **before** J_y is added
  $$G(I) = L^* + t_y$$
  - $G(I)$ will always be less than or equal to the average load of the machines + the longest task
    - This is because L^* has to have a load less than equal to the average!
    $$L^* + t_y \leq \frac{1}{m}\sum_{j=1}^{n} t_j + \max_{1\leq i\leq n} t_i$$
    - The sum of any two numbers is always $\geq 2\times$ the maximum of these two numbers
    - Take 4 and 5, $4 + 5 = 9$
    - $2 \cdot \max\{4, 5\} = 10$

$$\begin{aligned} G(I) &= L^* + t_y \\ &\leq \frac{1}{m}\sum_{j=1}^{n} t_j + \max_{1\leq i\leq n} t_i \\ &\leq 2 \cdot \max\left\{\frac{1}{m}\sum_{j=1}^{n} t_j, \max_{1\leq i\leq n} t_i\right\} \\ &\leq 2 \cdot OPT(I) \end{aligned}$$

- Therefore $\alpha = 2$

Vertex Cover

Defining the vertex cover problem

- A vertex cover is a set $C \subseteq V$ where every edge in E touches at least on node in C
- We want to find a **minimum** vertex cover that has the smallest possible number of vertices

The Greedy Algorithm

Algorithm Modified Approximation Algorithm for Minimum Vertex Cover

function VC($G = (V, E)$)
  $C \leftarrow \emptyset$ ▷ Initialize empty vertex cover set
  $E_c \leftarrow E$ ▷ Initialize working set of edges
  while $E_c \neq \emptyset$ **do** ▷ Continue while there are uncovered edges
    select edge $(u, v) \in E_c$ ▷ Pick any remaining edge
    $C \leftarrow C \cup \{u, v\}$ ▷ Add both endpoints to vertex cover
    remove all edges incident to u and v from E_c ▷ Remove covered edges
  end while
  return C ▷ Return the vertex cover
end function

- Start at a node u ,and select an edge incident to it with destination node v
- Remove all the edges touching either of these two nodes u or v
- Continue until there are no edges left

Lower Bound for the Optimal Problem

- Let E^* be the set of **dijoint edges** in the graph
  - Disjoint edges are edges that do not share any vertices whatsoever - either u or v
- Let $OPT(H)$ be the size of the minimal vertex cover for instance H

$$OPT(H) \geq |E^*|$$

- At least one vertex will be needed for each disjoint edge to be covered in all instances

Finding Alpha

- In the original pseudocode $|C|$ is twice the number of processed edges, we know:

$$|C| \leq 2 \cdot OPT(G)$$

- $\alpha = 2$

Weighted Vertex Cover

Defining the problem

- Similar to vertex cover, but each vertex v has a weight w(v)
- Goal: Find a vertex cover C with minimum total weight
  - Instead of minimizing $|C|$, we minimize $\sum_{v \in C} w(v)$

Linear Programming Optimal Solution

- Introduce decision variable x_v for each vertex $v \in V$
  - $x_v = 1$ would mean v is in the vertex cover and 0 otherwise
- Formulate a LP problem such that you minimize the weight of the vertices in the vertex cover
- Objective function:

$$z = \sum_{v \in V} w(v) \cdot x_v$$

- Constraints:

$$x_u + x_v \geq 1 \quad \forall (u,v) \in E$$
$$x_v \in \{0,1\}$$

Linear Programming Approximation

- We can approximate the LP problem by **relaxing the constraints** $x_v \in \{0,1\}$ to $0 \leq \overline{x_v} \leq 1$
- So the new LP **approximation** of the original problem can be as follows:

$$\begin{aligned} \text{minimize } \overline{z} &= \sum_{v \in V} \overline{x_v} \cdot w(v) & \\ \text{subject to } \overline{x_u} + \overline{x_v} &\geq 1 & \forall (u,v) \in E \\ \overline{x_v} &\leq 1 & \forall v \in V \\ \overline{x_v} &\geq 0 & \forall v \in V \end{aligned}$$

Why This Works

- After solving the relaxed LP, we can get a vertex cover by choosing vertices with $\overline{x_v} \geq \frac{1}{2}$
- For any edge (u,v), at least one of $\overline{x_u}$ or $\overline{x_v}$ must be $\geq \frac{1}{2}$ as $\overline{x_u} + \overline{x_v} \geq 1$
- Therefore all edges are covered by definition
- This may not necessarily be optimal but always guarantees a vertex cover

Finding Alpha

Finding α

- Let $\overline{z^*}$ be the optimal value of the LP relaxation
- Let z^* be the optimal value of the original problem
- Because the LP relaxation removes integral constraints:

$$\overline{z^*} \leq z^*$$

- This means the LP relaxation solution is a lower bound on the optimal value

Proving 2-Approximation

- Let $\overline{x_v^*}$ be the solution to the LP relaxation
- Let x_v^* be the solution obtained by rounding
- We can show:

$$\begin{aligned}
\sum_{v \in C} w(v) &\leq \sum_{v \in C} 2 \cdot \overline{x_v^*} \cdot w(v) && \text{(since } C = \{v : \overline{x_v} \geq 0.5\}\text{)} \\
&\leq \sum_{v \in V} 2 \cdot \overline{x_v^*} \cdot w(v) && \text{(since } \overline{x_v} \geq 0 \text{ for all } v \in V\text{)} \\
&= 2 \cdot \overline{z^*} && \text{(by definition of } \overline{z^*}\text{) (the objective function)} \\
&\leq 2 \cdot z^* && \text{(since } \overline{z^*} \text{ is a lower bound)}
\end{aligned}$$

- Therefore, $\alpha = 2$