Approximation Algorithms

Approximation algorithms are efficient algorithms that find approximate solutions to optimization problems.

Load Balancing Problem

Input:

- m resources with equal processing power
- n jobs, where job j takes t_j time to process

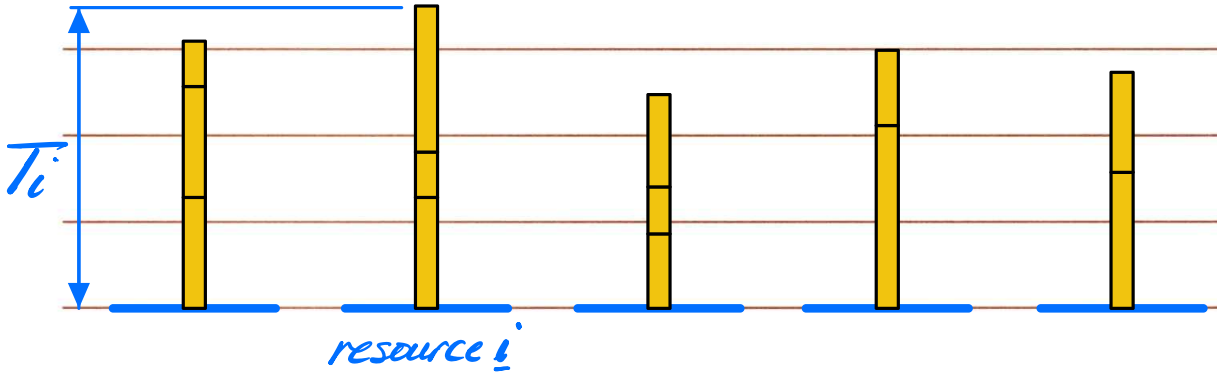
Objective:

Assignment of jobs to resources such that the maximum load on any machine is minimized.

Notation:

T_i : Load on machine/resource i

T^* : Value of the optimal solution

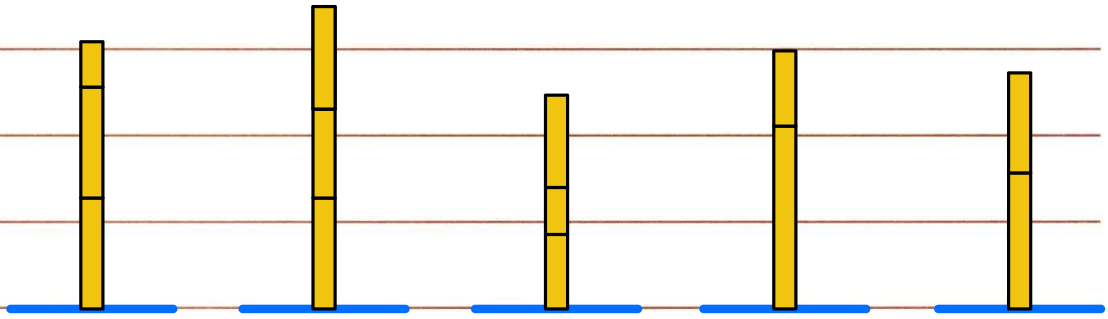


Greedy Balancing

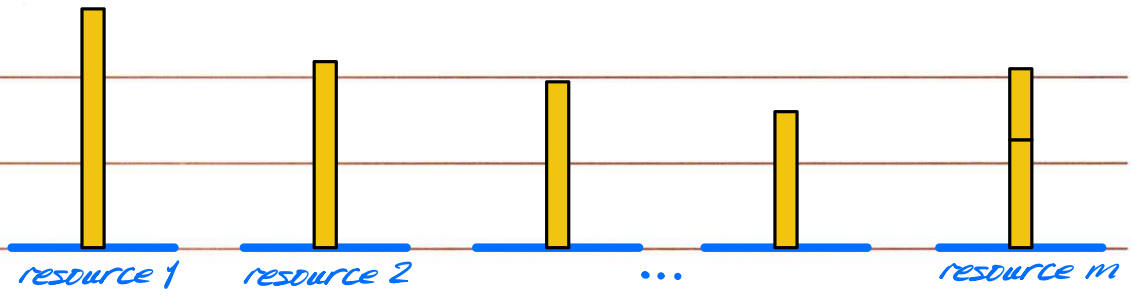
- Process requests in the order given
- Place the next request on resource with the smallest load.

Claim: This algorithm will produce a 2-approximation

Proof: Let's say resource i ends up with the biggest load, and that job j is the last job placed on that resource.



Improved Approximation to Greedy Balancing



If we have no more than m jobs, our solution will obviously be optimal.

Vertex Cover Problem

- Find the smallest vertex cover set in G.

(optimization version)

Here is a 2-approximation algorithm to the vertex cover problem:

```
Start with S = Null
While S is not a vertex cover
    Select an edge e not covered by S
    Add both ends of e to S
Endwhile
```

Why is this a 2-approximation?

Question:

Since Independent Set $\leq_p$ Vertex Cover, Can we use our 2-approximation algorithm for vertex cover to find a $\frac{1}{2}$-approximation to independent set?

Theorem: Unless P=NP, there is no $\frac{1}{n^{1-\varepsilon}}$ approximation algorithm for the maximum independent set problem for any $\varepsilon > 0$ where n is the no. of nodes in the graph.

Question:

Since Vertex Cover $\leq_p$ Set Cover,

Can I use a 2-approximation algorithm for set cover to find a 2-approximation to vertex cover?

Max_3SAT Problem Statement:

Given a set of clauses of length 3, find a truth assignment that satisfies the largest no. of clauses.

Find a $\frac{1}{2}$ approximation to the Max_3SAT problem.

Consider the following problem:
Given a directed graph G, remove some edges to turn G into a DAG of maximum size.

We are looking for a 1/2 approximation:

Linear Programming

System of Linear Equations

$$[A][x] = [B]$$

Linear Programming (LP)

$$[A][x] \leq [B]$$

Objective function: $[c^T][x]$

Goal: Minimize the objective function subject to the given constraints

Standard form of Linear Programming

- All constraints are of the form $\leq$
- All variables are non-negative
- Objective function is maximized

Ex. $x_1 - x_2 \geq 0$

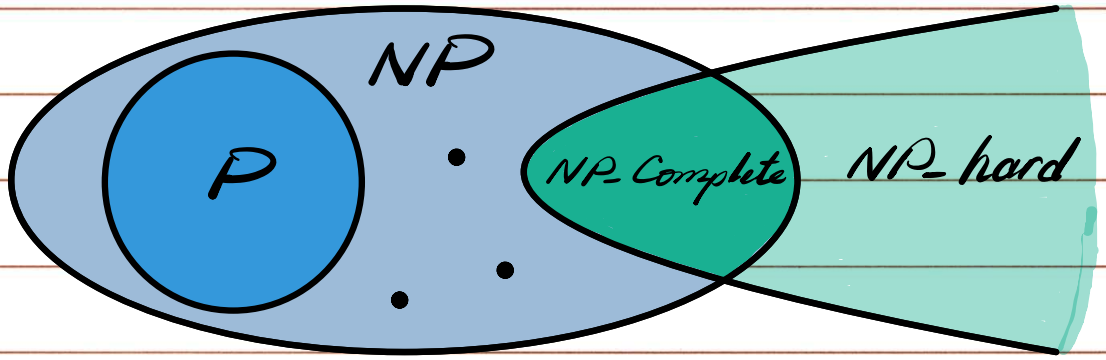
$x_1 \geq 0$

$x_2 \geq 0$

$x_1 + x_2 \leq 4$

Objective Function:

Maximize $x_1 + 2x_2$


NP
P
NP_Complete
NP_hard

Weighted Vertex Cover Problem

For $G=(V,E)$, $S \subseteq V$ is a vertex cover such that each edge has at least one end in S.
Also, $w_i \geq 0$ for each $i \in V$, so the total weight of the set $= W(S) = \sum_{i \in S} w_i$

Objective: Minimize $W(S)$

Decision variables:

How do we make sure edge e is covered?

Solving Max Flow using LP

Our variables will represent flows over edges

Objective function: Maximize $\sum_{e \text{ out of } s} f(e)$

Subject to:

$0 \leq f(e) \leq c_e$ for each edge $e \in E$

$\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$ for $v \in V$ except for s & t

Note that equalities can be easily represented as inequalities

$A = B$ is equivalent to

$$\begin{cases} A \leq B \\ B \leq A \end{cases}$$

Solving Shortest Path using LP

So our LP formulation will be:

1. The *bin packing* problem is as follows. You have an infinite supply of bins, each of which can hold M maximum weight. You also have n objects, each of which has a (possibly distinct) weight w_i (any given w_i is at most M). Our goal is to partition the objects into bins, such that no bin holds more than M total weight, and that we use as few bins as possible. This problem in general is **NP**-hard.

Give a 2-approximation to the *bin packing* problem. That is, give an algorithm that will compute a valid partitioning of objects into bins, such that no bin holds more than M weight, and our algorithm uses at most twice as many bins as the optimal solution. Prove that the approximation ratio of your algorithm is two.

2. Suppose you are given a set of positive integers A: $a_1\ a_2\ \ldots\ a_n$ and a positive integer B. A subset $S \subseteq A$ is called *feasible* if the sum of the numbers in S does not exceed B.

The sum of the numbers in S will be called the *total sum* of S. You would like to select a feasible subset S of A whose total sum is as large as possible.

Example: If A = {8, 2, 4} and B = 11 then the optimal solution is the subset S = {8, 2}.

Give a linear-time algorithm for this problem that finds a feasible set $S \subseteq A$ whose total sum is at least half as large as the maximum total sum of any feasible set $S' \subseteq A$. Prove that your algorithm achieves this guarantee.
You may assume that each $a_i \leq B$.

3. A cargo plane can carry a maximum weight of 100 tons and a maximum volume of 60 cubic meters. There are three materials to be transported, and the cargo company may choose to carry any amount of each, up to the maximum available limits given below.

- Material 1 has density 2 tons/cubic meter, maximum available amount 40 cubic meters, and revenue $1,000 per cubic meter.
- Material 2 has density 1 ton/cubic meter, maximum available amount 30 cubic meters, and revenue $1,200 per cubic meter.
- Material 3 has density 3 tons/cubic meter, maximum available amount 20 cubic meters, and revenue $12,000 per cubic meter.

Write a linear program that optimizes revenue within the constraints. You do not need to solve the linear program.

4. Recall the 0/1 knapsack problem:

Input: n items, where item i has profit p_i and weight w_i, and knapsack size is W.

Output: A subset of the items that maximizes profit such that the total weight of the set ≤ W.

You may also assume that all $p_i \geq 0$, and $0 < w_i \leq W$.

We have created the following greedy approximation algorithm for 0/1 knapsack:

Sort items according to decreasing p_i/w_i (breaking ties arbitrarily).

While we can add more items to the knapsack, pick items in this order.

Show that this approximation algorithm has no nonzero constant approximation ratio.

In other words, if the value of the optimal solution is P*, prove that there is no constant ρ ($1 > \rho > 0$), where we can guarantee our greedy algorithm to achieve an approximate solution with total profit ρ P*.

5. There are *n* people and *n* jobs. You are given a cost matrix, C, where C_{ij} represents the cost of assigning person i to do job j. Each applicant *i* has a subset of jobs S_i that he/she is interested in. Each job can only accept one applicant and an applicant can be appointed to only one job. Write an integer linear program to minimize the total cost of assigning all applicants. You can assume that a feasible assignment of people to jobs exists.

Solution