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

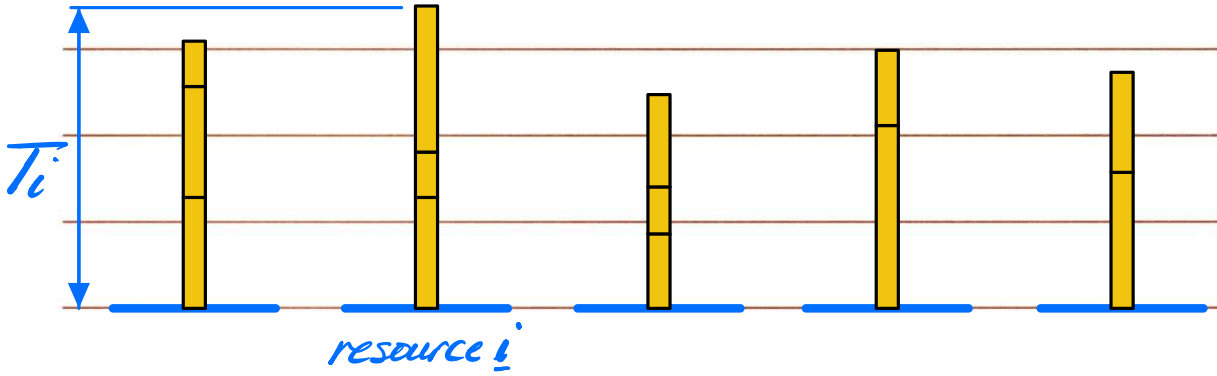
Objective:

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

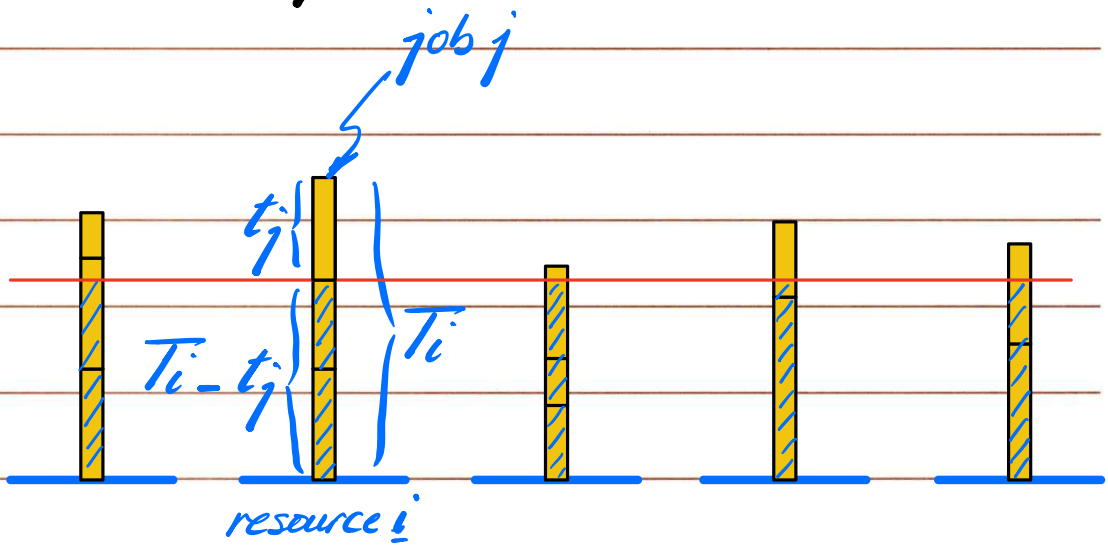


Greedy Balancing

- 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.



Load on resources i consists of $t_j + (T_i - t_j)$.

And we know that

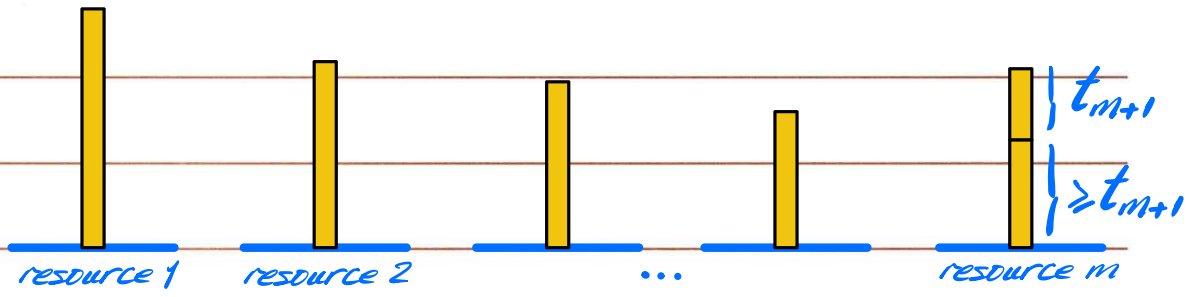
- $t_j \leq T^*$ since jobs cannot be split across resources.
- $(T_i - t_j) \leq T^*$ since at the time that job j was placed on resource i, all other resources must have had a load of at least $(T_i - t_j)$

So, $t_j + (T_i - t_j) \leq 2 \cdot T^*$

or $T_i \leq 2 \cdot T^*$

Improved Approximation to Greedy Balancing

Initially sort jobs in decreasing order of duration. Then use the same greedy balancing algorithm.



m resources

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

Otherwise, we will have at least two jobs on resource m.

$T^* \geq t_m + t_{m+1}$, since $t_{m+1} \leq t_m$, then

$T^* \geq 2 \cdot t_{m+1}$

Since jobs are sorted in decreasing order of duration, $t_j \leq t_{m+1}$, so

$T^* \geq 2 \cdot t_j$

or

$t_j \leq \frac{1}{2} T^*$

We still have $(T_i - t_j) \leq T^*$

and $t_j \leq \frac{1}{2} T^*$

$T_i \leq 1.5\, T^*$

This is a 1.5 approximation

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

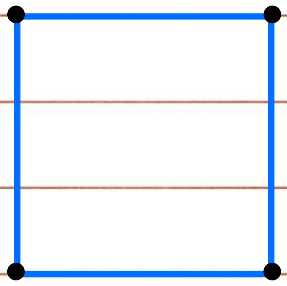
Why is this a 2-approximation?

At each step, we place both ends of edge e in S, but the optimal solution will need at least one of them. So we are within a factor of 2 of the opt. solution.

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? No!

Example:



A 2-approximation for this graph can have all 4 nodes in S. It complement will have a size of 0!

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? Yes!

A set cover solution of size k will produce a same size solution for the corresponding vertex cover problem.

And the opt. sol. size for set cover is equal to the opt. sol. size for the corresponding vertex cover problem.

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 ½ approximation to the Max_3SAT problem.

- Set everything to true

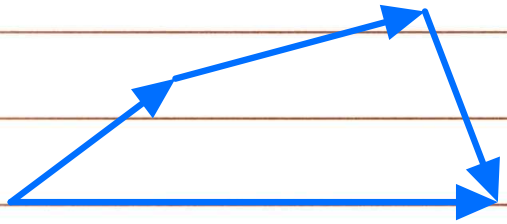
  If at least 50% of clauses evaluate to true, we are done

  Otherwise, set everything to false

More sophisticated approximation methods can get to within a factor of 8/9 of the opt. solution.

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

e.g.



Removing any one of the edges will result in a max. size DAG of size 3.

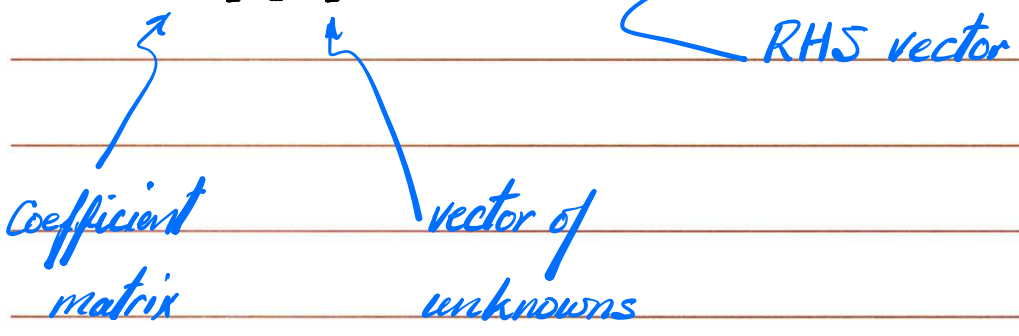
We are looking for a ½ approximation:

Choose an arbitrary ordering of nodes. If this does not give you a DAG of size at least $m/2$, then reverse the order.

Linear Programming

System of Linear Equations

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



Linear Programming (LP)

LP deals with inequalities/constraints

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

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

i.e. They must have positivity constraints

- Objective function is maximized

Ex. $x_1 - x_2 \geq 0$

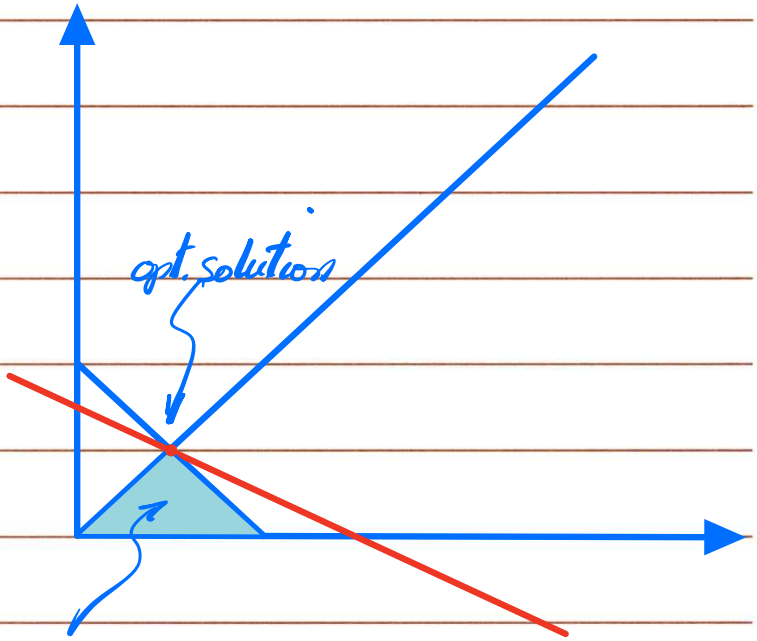
$x_1 \geq 0$

$x_2 \geq 0$

$x_1 + x_2 \leq 4$

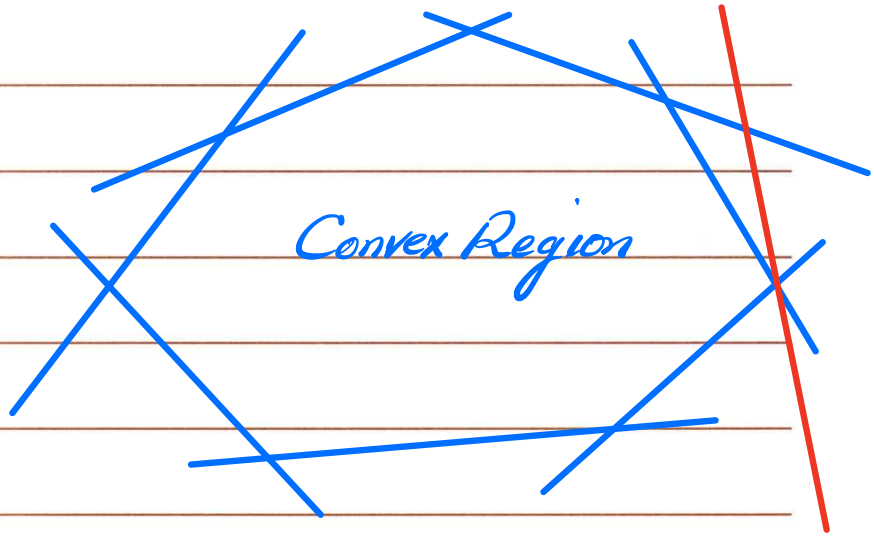
Objective Function:

Maximize $x_1 + 2x_2$

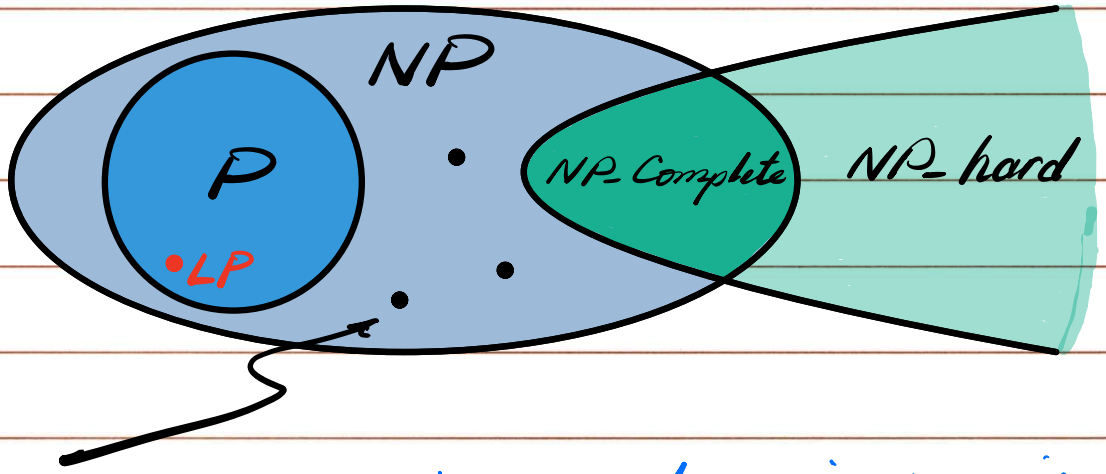


feasible region

Simplex Method



The intersection of a set of linear constraints is a convex region. And if the objective function is also linear, the opt. solution will always be at a vertex of that convex region.



LP was in this region (NP-intermediate) until 1979, when a polyn. time algorithm to solve LP was developed.

The Simplex method has an exponential time worst case performance but in practice normally runs in polynomial Time.

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 $W(S)$

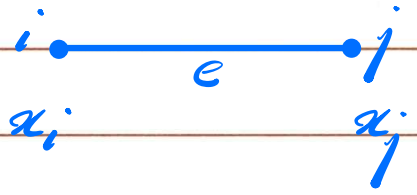
Decision variables:

for each $i \in V$, x_i is a decision variable

$$\begin{cases} x_i = 0 & i \notin S \\ x_i = 1 & i \in S \end{cases}$$

How do we make sure edge e is covered?

We must have: $x_i + x_j \geq 1$



So, our formulation will be:

Minimize $\sum w_i x_i$

Subject to:

$$x_i + x_j \geq 1 \quad \text{for } (i,j) \in E$$

$$x_i \in \{0,1\} \quad \text{for } i \in V$$

The above formulation represents an

Integer Linear Program

- Linear Programming
  Continuous variables
- Integer Programming
  Discrete variables
- Mixed Integer Programming
  Both Continuous & Discrete variables
- Nonlinear Programming
  Nonlinear constraints or objective function

To find an approximate solution using LP, we drop the requirement that $x_i \in \{0,1\}$ and solve the resulting LP (in polynomial time) to find $\{x_i^*\}$ with values between 0 & 1.

We define the weight of the LP solution W_{LP} as follows:

$$W_{LP} = \sum w_i x_i^*$$

Interpreting the LP solution:

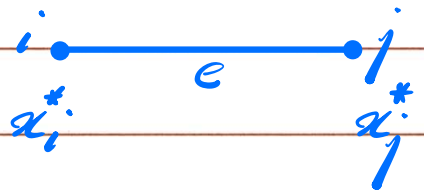
Obviously, if $x_i^* = 0 \Rightarrow i \notin S$, and
$x_i^* = 1 \Rightarrow i \in S$

But what about values between 0 and 1?

Say $S_{\geq 1/2} = \{ i \in V : x_i^* \geq 1/2 \}$

Claim: $S_{\geq 1/2}$ is a vertex cover set

We must have: $x_i^* + x_j^* \geq 1$



It is impossible for both x_i^* and x_j^* to be less than $1/2$. So $S_{\geq 1/2}$ will always cover every edge.

Claim: Our solution ($S_{\geq 1/2}$) is a 2-approximation

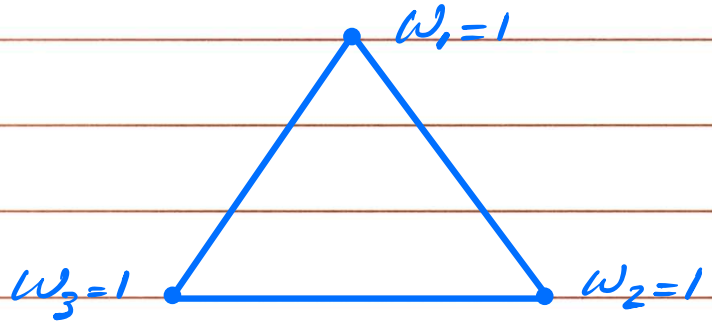
Assume S' is the optimal vertex cover set and $W(S')$ is the weight of the optimal vertex cover set.

$$\begin{cases} W_{LP} \leq W(S') \\ \text{and } W(S_{\geq 1/2}) \leq 2 * W_{LP} \end{cases}$$

At most we round up x_i^* from .5 to 1, i.e. increase by at most $2x$.

So $W(S_{\geq 1/2}) \leq 2 * W(S')$

Ex.



Solution to resulting LP:

$x_1^* = x_2^* = x_3^* = .5$

$w_{LP} = 1.5$

$w(S') = 2$

$w(S_{\geq 1/2}) = 3$ which is within $2 * w(S')$

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

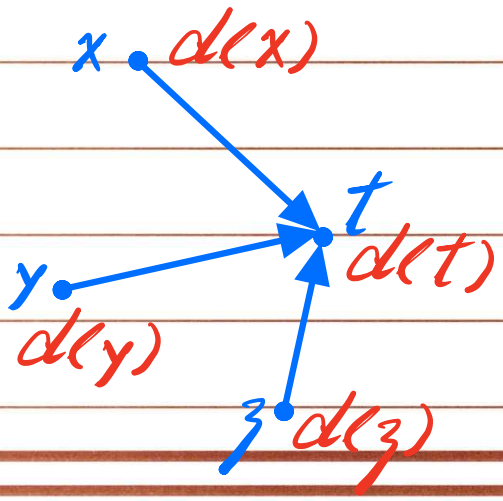
Note that equalities can be easily represented as inequalities

$A = B$ is equivalent to

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

Solving Shortest Path using LP

Our variables will represent shortest distance from s to v ($d(v)$) for each $v \in V$



To ensure that to get to node t we get the shortest distance, we will create the following constraints:

$$d(t) \leq d(x) + c_{xt}$$
$$d(t) \leq d(y) + c_{yt}$$
$$d(t) \leq d(z) + c_{zt}$$

So our LP formulation will be:

$$\begin{cases} d(v) \leq d(u) + w(u,v) \\ \qquad \text{for edge } (u,v) \in E \\ d(s) = 0 \end{cases}$$

Objective function: ~~Maximize~~ ~~Minimize~~ $d(t)$