Lecture 04/01/24

Optimization Problems

↳ NP-optimization problems

Set of instances

For every instance there is an associated (finite) set of "solutions" for the instance,

cost assigned to every solution

Number of solutions for a given instance is finite but may be very large (in terms of the size of instance)

Optimization problem: Find a solution with minimum/maximum cost

NP: Given a possible $sol^n$ it can be verified in polynomial time if it's actually a valid $sol^n$, cost can be computed in poly time.

Trivial: Enumerate over $sol^n$, so v. much decidable problems.

But, bottleneck = time.

The decision problem of deciding whether $\exists$ $sol^n$ with cost $\leq k$ for an instance $q$ and number $k$ is NP-complete

Finding optimal solution is NP-hard. $\Leftarrow$

NP-complete problems are not equivalent w.r.t. approx. Some problems are v. hard to approximate

① $n$ objects. $i^{th}$ object size $s_i$

Two bags with capacity $C$ each.

Pack maximum # objects in the two bags.

NP-hard opt problem.

$\geq$

Partition problem. If $\sum s_i = 2C$ then pack all objects iff $\exists S$ s.t. $\sum_{i \in S} s_i = C$ i.e. partitioning into two equal subsets

ALG: Order objects so that $s_1 \leq s_2 \leq \cdots \leq s_n$

At $i^{th}$ step put object $i$ in bag 1 if fits, o/w put in bag 2 o/w discard all remaining objects.

$\longrightarrow$ Claim: gives atleast OPT $-$ 1

2 2 3 3

| 5 | 5 |
|---|---|
| 2 2 | 3 |

So, clearly not optimal.

Approximation algorithm :

A polytime algorithm that outputs a solution for each instance along with a guarantee or bound on how good the solution is.

$A(I)$ = cost of solution given by algorithm for instance $I$

$opt(I)$ = opt " — " $I$

Bound on $|A(I) - opt(I)|$ ← Absolute error

Bounded ideally by a "constant".

$\dfrac{|A(I) - Opt(I)|}{opt(I)}$ ← Relative error

---

This algorithm satisfies $A(I) \geq opt(I) - 1$ for all instances.

Upper bound on opt : max $k$ s.t. $\sum_{i=1}^{k} s_i \leq 2C$

take objects greedily.

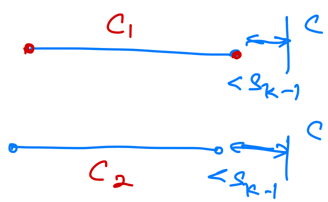
instead of using the $o(I)$ which is unknown bound error by comparing to upper bound for max, lower bound for min problem

i.e. $\sum_{i=1}^{k+1} s_i > 2C$

$\Rightarrow$ $OPT \leq k$.

Claim : $A(I) \geq k-1$

Pf : Suppose $s_{k-1}$ was not included




$c_1 + s_{k-1} > C$

$c_2 + s_{k-1} > C$

$\Rightarrow$ $c_1 + c_2 + 2s_{k-1} > 2C$

$c_1 + c_2 + s_{k-1} + s_k \geq c_1 + c_2 + 2s_{k-1} > 2C$

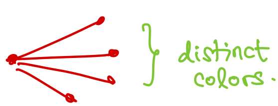
This contradicts the fact that $\sum_{i=1}^{k} s_k \leq 2C$.

---

Edge-coloring

Instance is a graph and a solution is a coloring of edges s.t. edges with common endpoints have different colors. Cost = no. of colors used to minimize.

Max degree $= \Delta$




$Opt(I) \geq \Delta$

Vizing thm : It is always possible to color such a graph with $\Delta+1$ colors.

Deciding if opt $= \Delta$ or $\Delta+1$ is NP-complete.

Given a graph, soln is a spanning tree. Cost is the maximum degree of a vertex $\geq$ Hamiltonian path. (NP-hard)

Non-trivial alg (end of course).

↳ Always find a spanning tree with max degree one more than optimal.

## Logistics

1. The design of approximation algorithms : Shmoys & Williamson (primary reference)
2. Approximation algorithms : Vazirani (problems + more examples)

## Evaluation

Assignments + Midsem + Endsem
(Problems from books)

---

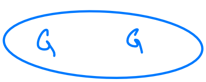
## Vertex Cover problem :

$G \leftarrow$ find smallest subset of vertices covering all edges.

Claim: There cannot be an algorithm with $A(I) \leq opt(I) + 1$ for all $I$ unless $P = NP$.

Show if such an algorithm exists, we can find optimum in polynomial time.

give input = two copies



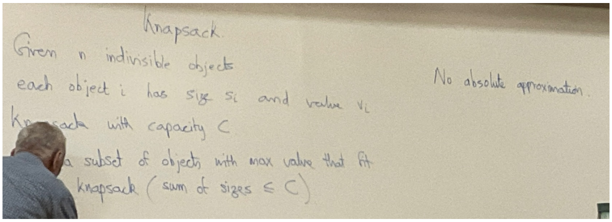

Idea:
Construct instances where next suboptimum solution is far away from optimum.
Approx $\Rightarrow$ exact.

Can repeat this constant # times, to refute any additive approx.

$A(I) \leq O(I) + \frac{n}{2}$ ✓ (using max-match)

Question not studied well for additive.

$A(I) \leq O(I) + \frac{n}{3}$ ?




(Amplification of gaps)

8/1/24

If $\exists$ k - constant approx

$\exists$ subset with val $\geq V$

$(I, V) \longrightarrow I'$ (mult by k+1)

yes $\longrightarrow$ OPT $\geq (k+1)V$

no $\longrightarrow$ OPT $\leq (k+1)(V-1) = (k+1)V - (k+1)$

Run approx algo on $I'$

$\Rightarrow$ Ans to $I$ is yes iff the sol$^n$ obtained has value $\geq (k+1)V - k$

(yes, no instance have gap $\geq k$).

In general for problems with weights, absolute approximation is usually refuted via scaling arguements.

## Special case of Integer linear program

$x_i \rightarrow$ takes value 0 or 1

indicates whether ith object is included or not

$$\sum_{i=1}^{n} s_i x_i \leq C$$

$$\max \sum_{i=1}^{n} v_i x_i$$

$$x_i \in \{0, 1\}$$

Any ILP reduces to 0 - 1 integer lin. program. $\longleftarrow$ variables restricted to be 0, 1 lin constraints and objective.

{indicators using bits}

LP relaxation (pretty bad. High density high size objects)

$0 \leq x_i \leq 1$

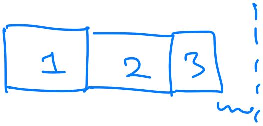
Allow variables to take fractional values between 0 and 1

Optimal value of LP will be an upper bound (maximization) on the optimal integral solution.

Optimal fractional obtained by a greedy algorithm. Sort objects such $\frac{v_1}{s_1} \geq \frac{v_2}{s_2} \geq \dots \geq \frac{v_n}{s_n}$ and at $i^{th}$ step as large a fraction of $i^{th}$ object as possible.

Challenge: Come up with ILP so that integrality gap is less.

Modification: Remove objects with $s_i \geq C$.




exclude.

| | | |
|---|---|---|
| $v$ | $2$ | $C$ |
| $s$ | $1$ | $C$ |

Cap = C

Greedy: value = 2

opt = C.

$$A > \text{opt} - v_i$$

(fractional object can contribute atmost $v_i$)

$$\text{Opt} \leq A + v_i \qquad (v_i >> A)$$

Issue!

Fix?

Take another solution as object with maximum value! (out of $s_i \leq C$)

Output = larger of these two solutions.

Still .... $\text{Opt} < A + v_i$

$$A \geq v_i$$

$$\Rightarrow A > \frac{1}{2} \times \text{opt} \qquad \left(\frac{1}{2}\text{-approximation algorithm}\right)$$

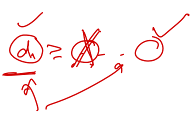
An algorithm for a maximization problem is an $\alpha$-approx. algorithm if for every instance $I$,

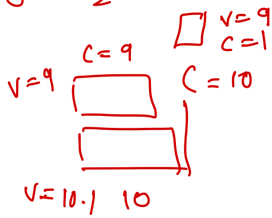
$$A(I) \geq \alpha \cdot \text{opt}(I). \quad \text{where } \alpha \leq 1$$

For a minimization problem,

$$A(I) \leq \alpha \cdot \text{opt}(I) \quad \text{where } \alpha \geq 1$$

Instance where Alg gives $\frac{1}{2}$ the optimum integer soln






| | | | |
|---|---|---|---|
| $V+8$ | $V+8$ | $2V$ | $2V$ |
| $\frac{C}{4}+\varepsilon$ | $\frac{C}{4}+\varepsilon$ | $\frac{C}{2}-\varepsilon$ | $\frac{C}{2}-\varepsilon$ |

$$\begin{cases} \text{Opt} \sim 4V \\ A \sim 2V \end{cases}$$

Further improvements?

$$\text{Opt} \le A + \underbrace{v_i}_{\text{ensure this has small value.}}$$

1. Take all possible subsets of size atmost k which are feasible & select these objects.
   "cardinality"

2. For remaining objects remove objects with value $\geq$ min value of k objects (wlog), or
   capacity $\geq$ rem. cap.

If opt has $\le k$ objects, we get exact opt.

If opt has $\geq k+1$ objects, we get maximum value elements (first k) and then greedy is applied.

Let $v_i$ be missed. Then $\text{opt}|_k \cup \underbrace{v_i}_{\text{value} \le \frac{\text{opt}}{k+1}}$ is feasible.

As

$$\text{opt} \le A + v_i \leftarrow \text{LP bound}$$

fractional object in greedy solution.

$$\Rightarrow A \geq \text{opt}\left(\frac{k}{k+1}\right).$$

**Lecture** 9/1/24

**Polynomial-time approximation scheme**

It is a family of approximation algorithms parametrized by an error parameter $\varepsilon > 0$, such that algorithm runs in polynomial time for any fixed $\varepsilon > 0$ and has error bounded by $\varepsilon \cdot \text{opt}$.

Minimization: $A(\varepsilon)(I) \le (1+\varepsilon)\, \text{opt}(I)$ for any instance $I$

Maximization: $A(\varepsilon)(I) \geq (1-\varepsilon)\, \text{Opt}(I)$

$$\text{opt} \le A + v_i \leftarrow \text{LP bound}$$

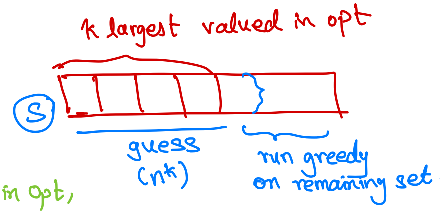
fractional object in greedy solution.

To get better approximations, ensure that $v_i$ is "small" compared to opt.

## PTAS

Fix a parameter $k \geq 1$

For all subsets $S$ of size atmost $k$ that are feasible (sum of sizes $\leq C$)




1) Select all objects in $S$ ← If $\leq k$ objects in opt, we get optimum.
2) Remove all objects whose value > the min value of $S$ ← when $S$ = k largest valued objects in optimum, this step doesn't change opt.
3) For remaining objects fill using greedy

Every remaining object has value $\leq \frac{opt}{k+1}$.

$\Downarrow$

greedy solution > opt − value of fractional obj.

$\geq opt \cdot \frac{k}{k+1}$

Output will be the best solution over all choices of $S$

This gives a solution with value $\geq \frac{k}{k+1}$ opt

Running time : $O(k\, n^{k+1}) \longrightarrow$ polynomial time for any fixed $k$.

$$\Big(\underbrace{\frac{C}{k+1}, \ldots\ldots, \frac{C}{k+1}}_{k+1 \text{ objects}}, \boxed{\varepsilon}\Big)$$

← tight example for approximation ratio.

Value $\Big(\frac{C}{k+1}, \ldots, \frac{C}{k+1}, \boxed{2\varepsilon}\Big)$

opt will not include this object. but greedy alg. always does!

| Alg | opt |
|---|---|
| $\frac{k}{k+1}C + 2\varepsilon$ | $C$ |

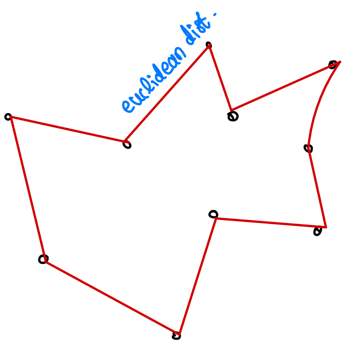
## Euclidean TSP

[Sanjeev Arora]

↓

Gave a PTAS for euclidean TSP.

[to be done]




Simple polygon with minimum perimeter with given points as corners

For arbitrary graph, no PTAS is known (1.4-approx is best)
But no proof that PTAS doesn't exist for metric TSP.

Fully Polynomial time approximation scheme (FPTAS) PTAS $\supseteq$ FPTAS.

$A(\varepsilon)$ should have running time which is polynomial in both $n$ and $1/\varepsilon$.

Our earlier alg was $O(\frac{1}{\varepsilon} n^{1/\varepsilon + 1})$ ← not an FPTAS.

$S_1 \quad S_2 \quad S_3 \quad \dots \quad S_n$

$v_1 \quad v_2 \quad \dots\dots \quad v_n$

If all values are 1, we have a simple greedy algorithm.

If all values are bounded by some constant $B$ and $B$ is a small number.

max possible value is $B \cdot n$

Use dynamic programming

For each value $0 \le v \le Bn$ find smallest size subset whose value is $\ge V$.

$S(i, V)$ = smallest size of objects that are subset of first $i$ objects with value $\ge V$.

$$S(1, v) = \begin{cases} 0 & v = 0 \\ S_1 & \text{if } v > 0. \end{cases}$$

$$S(i+1, v) = \min\{ \underbrace{S(i,v)}_{\text{exclude } i+1},\ \underbrace{S_{i+1} + S(i, v - v_{i+1})}_{\text{include } i+1} \}$$

Running time $= O(Bn^2)$

$S_1 \quad S_2 \quad S_3 \quad \dots \quad S_n$

$v_1 \quad v_2 \quad \dots\dots \quad v_n$

} Reduce values of these objects by scaling them with an appropriate factor

$$v_i' = \left\lfloor \frac{n v_i}{\varepsilon v_{max}} \right\rfloor$$

} $B$ is upper bounded by polynomial in $n$.

max value of an object.

$$= \left\lfloor \frac{v_i}{\left(\frac{\varepsilon v_{max}}{n}\right)} \right\rfloor$$

number of these units.

Question: How bad is the approximation?

$$v'_{i\,max} = \frac{n}{\varepsilon}$$

For this instance, $B = \frac{n}{\varepsilon}$. We can find optimum for this in $O\left(\frac{n^3}{\varepsilon}\right)$

By taking floor, amount lost from value of each object (in modified instance) is atmost $\frac{\varepsilon V_{max}}{n}$

Hence, amount lost in total $\leq n \times \frac{\varepsilon V_{max}}{n}$

$$= \varepsilon V_{max} \leq \varepsilon \, Opt.$$

$$Opt(I') \geq Opt(I) - \varepsilon \cdot V_{max}.$$

modified instance

$$\geq (1-\varepsilon)\, Opt(I)$$

---

Lecture 11/1/24

## Set cover

Given a set $U = \{e_1, e_2, \ldots, e_n\}$ called the universe and collection $C$ of subsets of $U$, $C = \{S_1, S_2, \ldots, S_m\}$. $S_i \subseteq U$ and each $S_i$ has weight $w_i$

$\bigcup_{i=1}^{m} S_i = U$. every element in universe is contained in some set in $C$.

Solution: A subset $C'$ of $C$ such that union of sets in $C'$ is also $U$.

$C' \rightarrow$ set cover. the sets in $C'$ cover $U$.

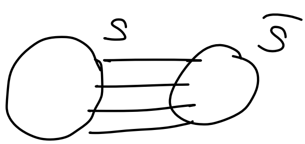
Cost = the sum of weights.

choose min cost sol$^n$ (NP-hard. reduce from vertex cover)

Min. wt spanning tree ← special case (PTIME solvable)

Connected subgraph with minimum weight

·Need to cover all cuts in a graph




atleast one included in spanning tree.

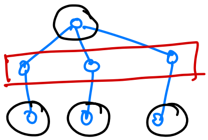
Approximation Algorithm

1. Try greedy algorithm (doesn't work exactly, but great for approximation)

1. Pick the set for which the ratio of wt / no. of new elements covered is as small as possible

2. Repeat this till all elements are covered

For VC problem. $1/\#\text{edges left}$ i.e. pick max degree and do repeatedly



opt

chosen.

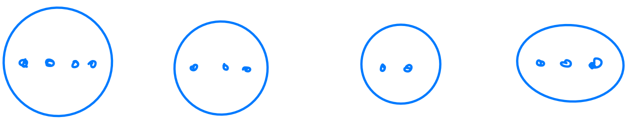
Hence, not an optimum algorithm clearly.

How bad is this algorithm? [Analysis]

Whenever a set is picked in algorithm assign a cost of $w(S_i)/\#$ new elements covered

To each of the new elements covered.

An element will be assigned cost only once when it gets covered in algorithm.



OPT

Adding cost of all elements $\geq$ total cost and sum over all sets

{ equality if sets are disjoint }

For any set $S$ in $C$, let cost of $S$ = sum of costs of elements $S$

$A(I)$ = sum of costs of all elements

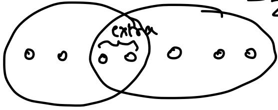
For any optimal solution $C'$

sum of costs of sets in $C'$ $\geq$ sum of costs of all elements

$= A(I)$

Idea: Cost isn't much more than the weight

To get a bound, we bound the cost of any set in $C$ in terms of weight



$S_1$

$S_2$

extra

$S_1$ : cost = wt, basically, sum of extra parts is not too much

Let $S$ be any set in $\mathcal{C}$ with $|S| = k$

$\overset{w}{\{x\}}$ ratio $= \frac{w}{1}$. This is smallest or something else covers $X$ with a smaller ratio.

i.e. Cost $\leq w$.

$$\text{cost}(S) \leq w(S)\Big(\underbrace{1 + \tfrac{1}{2} + \tfrac{1}{3} + \cdots + \tfrac{1}{k}}_{\approx \ln(k) \leq \ln(n)}\Big)$$

why?

Divide into groups based on when it got covered

$$\{\underbrace{\quad\quad}_{k_1} \,\vdots\, \underbrace{\quad\quad}_{k_2} \,\vdots\, \cdots \,\vdots\, \underbrace{\quad}_{k_r}\}$$

cost $\leq \frac{w}{k}$ | remaining $= k - k_1$, cost $\leq \frac{w}{k-k_1}$ | $\cdots$ | in the $r^{th}$ iteration. cost $\leq \frac{w}{k_r}$

First iteration cost $\leq k_1 \frac{w(S)}{k}$

$$\leq w(S)\left(\frac{1}{k} + \cdots + \frac{1}{k-k_1+1}\right)$$

$2^{nd}$ $\leq k_2 \frac{w(S)}{k-k_1}$

$$\leq w(S)\left(\frac{1}{k-k_1} + \cdots + \frac{1}{k-k_1-k_2+1}\right)$$

$\cdots\cdots$

Total cost of all elements $\leq w(S)\Big(\underbrace{1 + \tfrac{1}{2} + \cdots + \tfrac{1}{k}}_{H_k = k^{th} \text{ Harmonic number} = \ln(k)}\Big)$

$$\sum_{S \in \mathcal{C}'} w(S)\, H_k \geq \text{sum of costs of sets in } \mathcal{C}' \geq A(I)$$

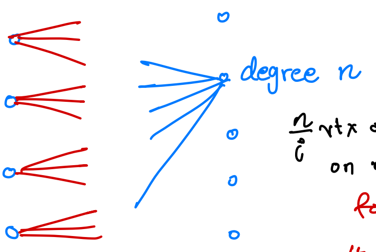
If every set has size atmost $k$, $H_i \leq \ln(k)$

$$\Big[\sum_{S \in \mathcal{C}'} w(S)\Big] H_k \geq A(I)$$

$$\Rightarrow \boxed{A(I) \leq OPT \times H_k} \longleftarrow \text{In general not a constant factor approx}$$

$k \leq n$ so $H_n$ - approximate algorithm

Idea: only RHS vertices are chosen!




$n$ vtx of degree $\leq n$ $\approx n^2$ edges

$\frac{n}{i}$ vtx of degree $i$ on right hand side for $i = 1$ to $n$. $\Big\}$ $\approx k$ such groups $\Rightarrow kn$ edges removed.

$$\#\text{ vertices in RHS} = \sum_{i=1}^{n} \frac{n}{i} = n H_n.$$

At each step there is a max degree vertex in RHS !

(So, analysis is tight.)

**Note**

If we have better than $\log n$-approx for set cover we get an quasi-polynomial ($n^{\log n}$) algorithm for any NP-complete problem {not believed to be true}

15/1/24

## Vertex Cover

$U$ = {set of edges in an undirected graph}

each vertex has a weight.

$C = \{ S_i \mid$ corresponding to vertex $v_i$ contains all edges with $v_i$ as an endpoint $\}$

Previous greedy alg. gives $\log \Delta$ approx.

→ every element in $U$ contained in exactly 2 sets

More general case, every element is contained in atmost $f$ sets, for some constant $f$

ILP formulation

$x_i \in \{0,1\}$ for each set $S_i$

$x_i = 1$ iff $S_i$ included in set-cover

For every element $e$ in $U$, $|U| = n$, $|C| = m$

$$\sum_{j,\, e \in S_j} x_j \geq 1 \qquad \xrightarrow[\text{relaxation}]{\text{LP}} \qquad 0 \leq x_i, \quad \sum_{j,\, e \in S_j} x_j \geq 1$$

$$\min \sum_{i=1}^{m} w_i x_i \qquad\qquad \min \sum_{i=1}^{m} w_i x_i$$

drop $x_i \leq 1$ without changing optimum

Suppose we have an optimal solution for the LP (polynomial time)

→ convert fractional values to integers in a suitable way whose cost is not too far from LP optimal [Rounding]

set cover instance where every element is in atmost $f$ sets (For vertex cover, there are exactly two)

$$x_{i_1} + x_{i_2} + \cdots + x_{i_f} \geq 1$$

Every $x_i \geq \frac{1}{f}$, round up to 1, others round down to zero.

Gives a valid solution, and cost is multiplied by atmost $f$.

$$x_i' \leq f x_i$$

$$\sum w_i x_i' \leq f \text{ LP-opt} \leq f \cdot \text{Opt}$$

So, this is an $f$ approximation. [Iterated rounding is also an idea here]

Use a lower bound on the cost of the optimal solution and find an integer solution whose cost is not much more than the bound.

k times

Lower bound $\leq$ Opt fractional sol$^n$ $\leq$ Opt Integer $\leq$ Algorithmic integer solution

$\uparrow$

Using dual of the LP

k times

## Dual of an LP

Multiply each inequality in original LP by a dual variable corresponding to inequality and add all the resulting inequalities to get a bound on the cost of the LP

$y_e \leftarrow$ corresponding to each inequality

$$y_e \geq 0$$

$$y_e \sum_{j, e \in S_j} x_j \geq y_e$$

Add up all of these.

$$\sum_{j=1}^{m} \Big( \underbrace{\sum_{e \in S_j} y_e}_{\leq w_j} \Big) x_j \geq \sum_e y_e$$

Ensure that $y_e$ satisfy $\sum_{e \in S_j} y_e \leq w_j$

since $x_j \geq 0$,

$$\sum_{j=1}^{m} w_j x_j \geq \sum_e y_e$$

**Dual**

assign non-negative values to elements in $U$

→ For a set $S_j$ in $C$, sum of values of elements in $S_j$ is atmost the weight of the set

→ Sum of values of all elements is a lower bound on the optimal fractional solution

Note : Analysis of Algorithm for set cover gave costs to elements, actually a dual solution

## Primal Dual Algorithms

Use dual variables to compute a bound on the fractional LP solution and try to construct an integer solution whose cost is not much more than the dual cost

While there exists an uncovered element $e$

→ increase dual value of variable till the inequality for some set containing it becomes tight.

→ Include all such sets in the set cover.

Repeat till all elements have been covered

Initially $y_e = 0$ for all elements, the set cover to be empty

$e \uparrow y_e$ until some dual inequality becomes tight for some set

(Happen for sets that contain $e$ and have minimum weight amongst them)

Compare cost of integer solution to the dual solution.

$$\text{Cost of solution obtained} = \sum_{\substack{\text{for all} \\ S \text{ in solution}}} w(S) \overset{\substack{\text{dual constraint}\\ \text{tight}}}{=} \sum_{\substack{S \text{ in} \\ \text{solution}}} \left( \sum_{e \in S} y_e \right) \leq f \underbrace{\sum_e y_e}_{\text{cost of dual}}$$

A particular $y_e$ appears atmost $f$ times

Gives an $f$-approximation

<u>Idea</u>: Bound # sets with $y_e \neq 0$

E.g. Dijkstra, Kruskal } $y_e \neq 0$ then there is only one set containing that $e$ (gives exact solution)

## Shortest Path

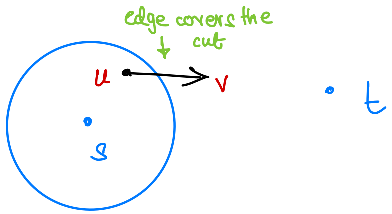
<u>Reverse deletion</u>

↓ Set cover obtained by greedy algorithm may contain redundant sets

In reverse order in which the sets were selected, delete a set if it does not cover any element not covered by the others.

Directed graph with +ve integer weights to edges and two vertices $s, t$. Find min wt path from $s$ to $t$.

An s-t cut is a subset of vertices that includes s but not t.



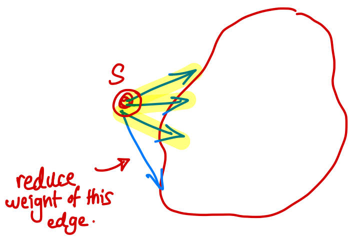

Elements of universe are s-t cuts, sets correspond to edges and an edge $(u,v)$ covers all cuts s.t. $u \in S$, $v \notin S$

Assuming +ve weights, min wt. set cover $\Leftrightarrow$ min-wt. path

Dual variables $\equiv$ cuts

inequalities for edges.

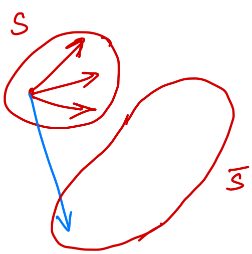
$$\sum_{e \text{ covers } S} y_S \le w_e$$




Increase the dual value for cut $\{s\}$ till it becomes equal to min weight edge leaving s.

Include all min weight edges leaving s.

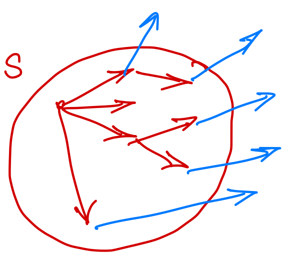
Reduce the remaining weight of these edges by min wt edge leaving s.




Increase the dual for this cut

$S = \{s\} \cup \{$all vertices nearest to s $\}$ ← update step in dijkstra's algorithm

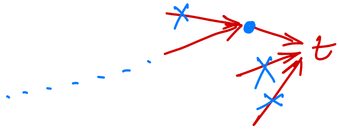
Increase in dual variable will be equal to difference in distances between vertices at level 2 and level 1.




and, repeat the procedure.

Continue till you get an s-t path

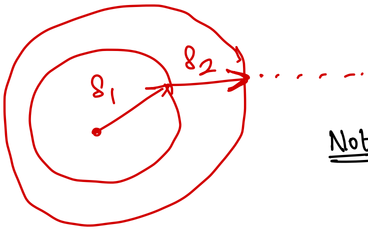
Applying reverse deletion, delete all redundant edges, till you are left with a single path.




Now, every cut with a non-zero value (dual) is covered by exactly one edge in the path.

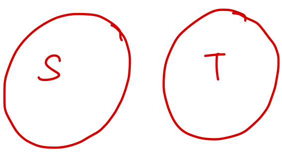
Weight of the path = total increase in dual variables

$\Rightarrow$ optimal fractional solution $\Rightarrow$ optimal integral solution.




<u>Note</u>: A similar analysis works for the min-cost flow primal-dual algorithm.

## Min-wt spanning tree




For any partition of vertices into two parts $S, T$ $\exists$ edge joining a vertex in $S$ to a vertex in $T$.

<u>Exercise</u>: Find rooted spanning tree at $u$ in a directed graph (same algo works)

Elements to be covered = all such partitions $(2^{n-1})$

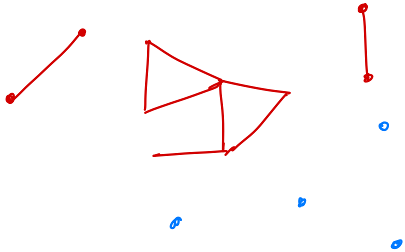

covers all partitions with $u, v$ in different parts. $2^{n-2}$ different such partitions

- Increase all dual values of uncovered elements by equal amount $\delta$ till some inequality becomes tight.
- Include that set

$$\sum_{e \text{ covers } S} y_S \le w_e$$

If everything inc. by $\delta$, increase for each edge is equal. So, tight first for min wt edge. Choose $\delta_1 = \frac{w_{min}}{2^{n-2}}$.

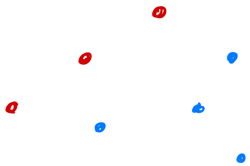
Inequality becomes tight for all min weight edges.



Already increased dual cost by $\delta_1 = \frac{w_{min}}{2^{n-2}}$ for all cuts.

Include all edges with min weight
Contract all connected components of that to a single vertex.

Consider this as a smaller graph



$n_1$ vertices

Effective weight = $w_{original}$ − dual cost already assigned

$= w_{original} - w_1$

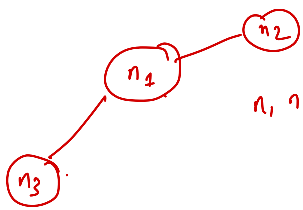
Pick minimum weight edges again.

(reverse deletion)
Finally, use a tree in each connected component ← gives a min weight ST.

So, Kruskal algorithm ≡ primal dual set cover algorithm.

Note : In min s-t path order of deletion doesn't matter, but here it does.

Counting # min wt spanning trees can be done using counting # ST



$n_1\ n_2\ n_3$

Every cut is covered by atleast two edges
↳ min wt 2-edge-connected subgraph

} NP-hard
(hamilton cycle is a min wt 2-edge-connected subgraph)

general problem.

**Network Design** [ref : vazirani]

For some pairs of vertices ∃ atleast some # paths b/w those pairs in min wt subgraph

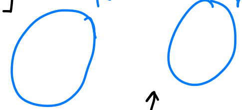
2-approx. algorithm is known, by primal-dual algorithm.

# Integrality Gap

18/1/24

Complete graph will all edges of weight 1.

$$\sum_{e \in cut} x_e \geq 1$$ [set cover formulation]




max $\frac{1}{k(n-k)}$

$\frac{1}{n-1}$

assign $\frac{1}{n-1}$ to each edge, gives

a valid fractional solⁿ with cost $= \frac{n}{2}$ } ← worst case.

$$\frac{n(n-1)}{2(n-1)}$$

Integer optimal $= n-1$

So, we can say Kruskal algorithm is 2-approximate using this LP

↓

Integrality gap of an LP relaxation of an ILP problem

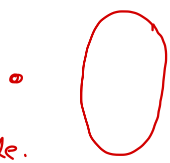
max ratio of $\frac{\text{optimal integral sol}^n}{\text{optimal fractional sol}^n} = r$

We cannot prove an approximation ratio better than $r$, if the optimal fractional solution is used as a bound.

$\#$ cuts $= 2^{n-1} - 1$ ← exclude empty set

Every edge covers exactly $2^{n-2}$ cuts. Assign $\frac{1}{2^{n-2}}$ to every cut then all (dual) inequalities become tight $\Rightarrow$ dual cost $= \frac{2^{n-1}-1}{2^{n-2}} \approx 2$ which is much less than dual optimal $= \frac{n}{2}$. Hence, those algorithms = Kruskal but optimality is yet to be shown using better LPs.

Isolating cuts: one vertex one side, all others on other side.



← There are $n$ such cuts, increase dual value for each of these.

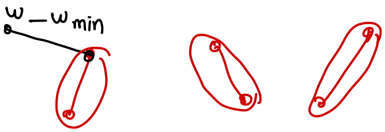
← belongs to 2 cuts.

Increase value of each isolating cut ← $\frac{w_{min}}{2}$

Include all min weight edges.

* Dual cost increases by $n\, w_{min}/2$.

Finally tree has only $n-1$ edges by reverse deletion




Reduce the weights of all other edges by $w_{min}$, contract connected components, repeat

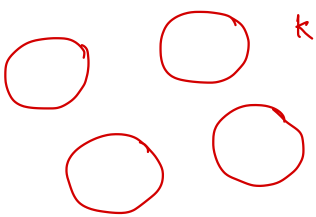
Dual cost of tree $= \underbrace{(n-1)}_{\text{edges in tree}} w_{min}$ for 1st step.

Opt : $(n-1) w_{min}$

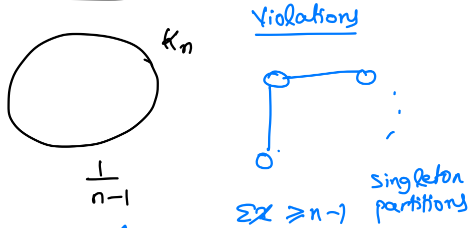
Excess cost = +ve.

Dual cost : $\frac{n w_{min}}{2}$ } 2 approx in each step.

Idea : Add more constraints to LP which eliminate fractional bound. [Generalises isolating cuts]




Example

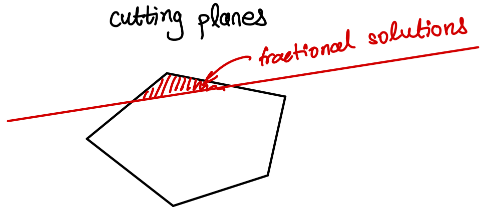



For any partition of vertices into k parts,

$\sum x_e$ for e with end points in different parts $\geq k-1$

this fractional solution is ruled out

cutting planes




With these constraints, the optimal fractional solution = optimal integral solution and shows optimality of Kruskal algorithm.

1. Consider partition with all singleton vertices and increase dual variable for that $\left(\forall e \in E, \sum_{\text{partition} \ni e} y_{\text{partition}} \leq w_e\right)$

Every edge crosses this partition, so increase this by min weight edge.

All min wt edge tight.

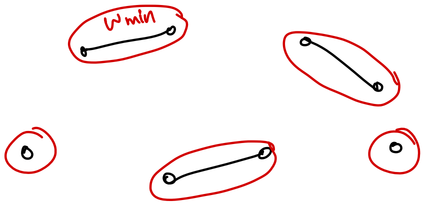
$\uparrow y_{\text{parti.}} \left( \sum_{e \in \text{partition}} x_e \geq n-1 \right)$ Dual cost $=$ RHS $\times\ \delta$,

so increases dual cost by $\underbrace{(n-1)}$ $w_{min}$

counting $w_{min}$ for every edge in ST.

$w_{min}$ contracted, weights of all other edges reduced by $w_{min}$

Easy arguement by induction, dual cost $=$ weight of minimum ST.




Remaining alg gives cost of ST for remaining components, add $w_{min}$ edges to get MWST for larger graph.

Note : This is not possible for 2-edge connected subgraph problem

$$\left.\begin{array}{l} \sum_{e \in cut} x_e \geq 2 \\ 0 \leq x_e \leq 1 \end{array}\right\}$$

Has an integrality gap of 2. $\leftarrow$ 2-approx algorithm

Problem is NP-hard

(Note : $\sum x_e \geq 2(k-1)$ is actually 2 disjoint ST problem, has an efficient algorithm using similar techniques)

Exercise : 2-edge connected by deleting edges until 2-edge connected is 2-approx

But better ratios known.

In general if constraints for each pair, set cover formulation is best known.

Feedback vertex Set $\leftarrow$ NP-hard {$\geq$ Vertex Cover}

Undirected Graph with weights assigned to vertices.

min wt subset of vertices whose removal destroys all cycles

i.e. every cycle must contain a vertex from that subset.

$x_v \to \{0,1\}$

$U$ = {cycles in the graph}

Every cycle $C$ $\quad \sum_{v \in C} x_v \geq 1$

$S_i \to$ corresponds to a vertex $v_i$

it covers all cycles $\ni v_i$

$\min \sum w_v x_v$

Integrality Gap $= \Omega(\log n)$ where $n = \#$ vertices.

Construct an example where optimal integer sol$^n$ $\uparrow$, fraction sol$^n$ $\downarrow$

For all $g \geq 3$ there exists a cubic (3-regular) graph with $2^g$ vertices and no cycles has length less than $g$. ($g$ = girth of the graph = length of smallest cycle)

# To do: show construction

$x_v \leftarrow \frac{1}{g}$ $\forall v$. is a feasible solution, cost $= \frac{2^g}{g}$

Optimal integral sol$^n$ contains $\frac{2^g}{4}$ vertices atleast $\leftarrow$ counting arguement.

FV set of size $k$, then $n-k$ vertices don't have a cycle

$\Rightarrow$ $n-k-1$ edges left atmost

$$\text{edges deleted} = n \times \frac{3}{2} - (n-k-1) \leq 3\underbrace{k}_{\#\text{vtx removed}}.$$

$$\Rightarrow \frac{n}{2} + 1 \leq 2k$$

$$\Rightarrow \boxed{k \geq \frac{n}{4}}$$

Hence integrality gap $\geq \frac{2^g/4}{2^g/g} = \frac{g}{4} = \Omega(g)$

which is $\Omega(\log n)$, $n = \#$vertices.

Note: You can get an LP with integrality gap of 2 using additional constraints

Remark: Small integrality gap formulation doesn't guarantee an algorithm with that approx ratio, but it does turn out that way for a lot of cases

Exercise: Construct algorithm attaining $\Omega(\log n)$ approximation algorithm

Lecture 23/1/24

$U = \{e_1, \dots, e_n\}$

each $e_i$ has weight $w_i$

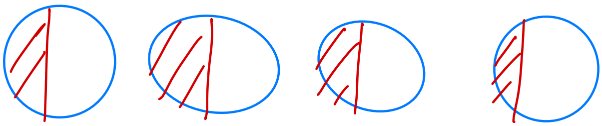
$C = \{S_1, S_2, \dots S_m\}$, $S_i \subseteq U$

Maximum coverage (NP-hard)

Find $k$ subsets in $C$ s.t. sum of weights of elements covered is maximized

Greedy algo:
choose the set s.t. sum of weights of newly covered elements is as large as possible. Repeat till $k$ sets are selected

$S_{opt}$ = optimal collection with weight $W_{opt}$



($k$ sets)

$\exists$ atleast one set wt $\geq \frac{W_{opt}}{k}$ {first step of greedy}

$(i+1)^{th}$ step of greedy. Let $w_i$ = wt of elements already covered in greedy

There must exist a set s.t. weight of uncovered elements in that set

$$\geq \frac{W_{opt} - w_i}{k}$$

$$\Rightarrow \quad w_{i+1} \geq w_i + \frac{W_{opt} - w_i}{k}$$

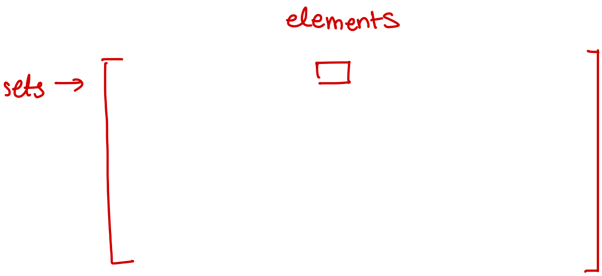
$$W_{opt} - w_{i+1} \leq W_{opt} - w_i - \left(\frac{W_{opt} - w_i}{k}\right)$$

$$\Rightarrow \quad W_{opt} - w_{i+1} \leq \underbrace{\left(1 - \frac{1}{k}\right)}_{\text{gap at } i\text{th step reduced by } 1-\frac{1}{k}} (W_{opt} - w_i)$$

$$\Rightarrow \quad W_{opt} - w_k \leq W_{opt}\left(1 - \frac{1}{k}\right)^k$$

$$w_k \geq W_{opt}\left(1 - \left(1 - \frac{1}{k}\right)^k\right) \geq W_{opt}\left(1 - \frac{1}{e}\right)$$

## Float optimization problem




$w_{ij}$ indicates how well the set $S_i$ covers element $e_j$

Choose a set $S$ of $k$ rows such that

$$\sum_{\substack{\text{all columns} \\ \text{of } j}} \max_{i \in S} \{w_{ij}\} \text{ is maximized}$$

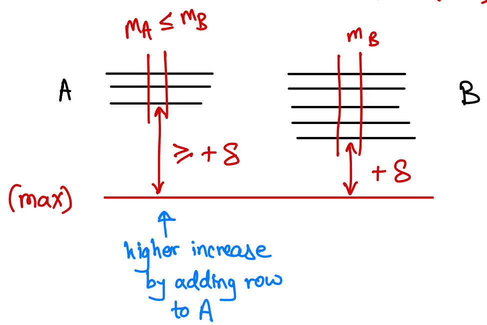
Previous problem had $w_{ij} = 1$ iff $j \in S_i$ else $w_{ij} = 0$, so is a special case of this.

At $i^{th}$ step choose a row that increases the objective function as much as possible.

Exact same arguement as before works, we get $1 - \frac{1}{e}$ approx algorithm using a greedy approach.

$$f(S) = \sum_j \max_{i \in S} \{w_{ij}\}$$

1. $f(S)$ is monotone : if $S_1 \subseteq S_2$, $f(S_1) \le f(S_2)$
2. Submodular : If $A \subseteq B$ and $x \notin B$ then $f(A \cup \{x\}) - f(A)$
$\ge f(B \cup \{x\}) - f(B)$ [decreasing marginal return]




$W_{opt}$ is the optimal value, $w_i$ is the current value $\exists$ a row such that it increases the value by $\frac{W_{opt} - w_i}{k}$, holds for any monotone, submodular $f^n$

* No approx algo known for non monotone submodular $f$

$Opt = \{v_1, v_2, \ldots, v_k\}$

$w_i \leftarrow S_i$
↓
adding all rows in opt to $S_i$ gives weight $\ge W_{opt}$ (monotone)

This gives an increase of atleast $W_{opt} - w_i$

$S \quad S \cup v_1 \quad S \cup v_1 \cup v_2 \quad \cdots \quad S \cup v_1 \cup \cdots \cup v_k$

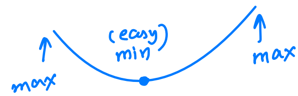
$w_i \longrightarrow \longrightarrow \quad \longrightarrow \; \geq w_{opt}$

$k$ steps.

one of these steps must have increased $f^n$ by $\dfrac{w_{opt} - w_i}{k}$

$S \cup v_1 \cup \cdots \cup v_j \xrightarrow[\geq w]{v_{j+1}}$ then $S \xrightarrow[\geq w]{v_{j+1}}$ by submodularity

Note: For an arbitrary submodular function, better than $\left(1 - \frac{1}{e}\right)$ approx is NP-hard.
Minimizing submodular $f^n$ has a PTIME algorithm, but maximization is hard.



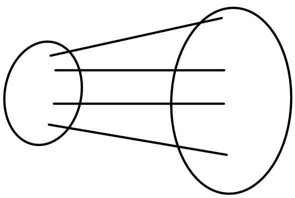

in $n$ dim with many vertices.

**Cut-function** [Max-cut]

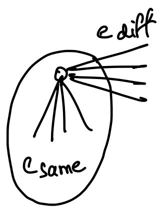
$f$: defined on subset of vertices

$f(S) = \#$ edges with one end point in $S$, other not in $S$

not monotone $f(\phi) = f(V) = 0$
but is submodular.
non-negative.



$\frac{1}{2}$-approx.




if $e_{diff} < e_{same}$,
switch vertex to the other side, keeping all other vertices the same.

atleast $\frac{1}{2}$ the edges are in the cut

Cardinality : size of set is at most $k$

each element has a cost, find subset with cost $\leq C$ having maximum $f$ value
[Knapsack constraint] where $f$ is monotone, submodular.

also has a constant factor approx.

Just greedy doesn't work, need to compare two different solutions.
(Analogous to knapsack)

Lecture 25/1/24

## K-centres

Metric space : Finite set of points with distance $d(u,v)$ specified for each pair of points

(1) $d(u,v) \geq 0$

(2) $d(u,v) = d(v,u)$

(3) $d(u,v) + d(v,w) \geq d(u,w)$

$\Leftrightarrow$ Complete graph with weights $d_{ij}$ assigned to edges

Choose k points (centres) such that maximum distance of a point from set of centres is minimized

$$d(u,S) = \min_{v \in S} d(u,v)$$

radius is the minimum value.

Find $S$ s.t. $|S| \leq k$ , $\max\limits_{v \in V} \min\limits_{u \in S} d(u,v)$ is minimized

k rows s.t. max entry in each column is minimized

$n \times n$

Dominating set

$S \subseteq V$ s.t. every vertex not in $S$ is adjacent to some vertex in the subset

Existence of Dominating set of size k is NP-complete.

Now, $d_{ij} = 1$ if $e(i,j)$ otherwise $d_{ij} = 2$

If we have better than 2-approximation, dominating set would be solved. ("gap" = 2)

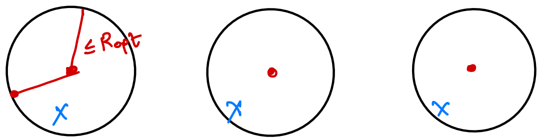
Dominating set of size k iff $\exists$ k-centres such that distance is 1.

So, better than 2-approximation is NP-hard.

**Algorithm** (2-approx)

Initially pick an arbitrary vertex

At each step pick a vertex that is furthest away from currently selected till k vertices are selected




Optimal Radius is $R_{opt}$

Groups formed in Opt by associating point with nearest point in S.

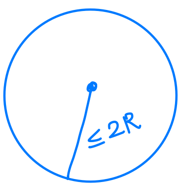
distance b/w any pair in the same group is atmost $2R_{opt}$.

If greedy picks one point from each cluster then distance of every point is atmost $2R_{opt}$ from greedy centres.

If not, it picks 2 points from same cluster. At the moment of picking $2^{nd}$ point, its dist $\leq 2R_{opt}$ hence, all other points are at distance $\leq 2R_{opt}$ from currently selected centres.

Guess the optimum value R

either show there is no solution with cost $< R$ or find a solution with cost atmost $2R$

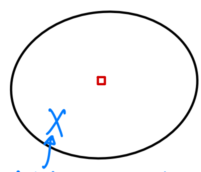



Pick arbitrary point,
remove all points
with distance $\leq 2R$

Repeat until all points are deleted

If $\exists$ a solution w/ cost $\leq R$ then atmost k points will be selected by this algorithm.

If more are selected $\Rightarrow$ no solution with cost $\leq R$



deletes the entire cluster!

opt

$$\Big[\,0\;,\;\big[\,l\;\cdots\;\overset{\downarrow}{R}\;\;\;\;u\,\big]\;\;\text{max distance}\,\Big]$$

{we need to find optimal radius}

There is no solution w/ cost $\leq l$ and $\exists$ a solution with cost $\leq 2u$.

Now, do binary search till length of interval becomes one.

$$\Rightarrow \quad R_{opt} \leq R^* \leq 2R_{opt}$$

## Weighted k-center

Each vertex has a cost for selecting

We consider only subsets of vertices s.t. cost $\leq$ some specified cost $C$

Sort all edges in non-decreasing order of distances

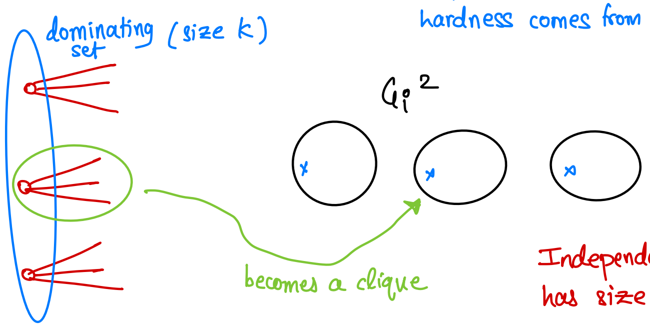
$d(e_1) \leq d(e_2) \leq \dots \leq d(e_{\binom{n}{2}})$ $\leftarrow$ bottleneck problems (e.g. min of max weight edge in ST)

k-center problem $\Leftrightarrow$ Finding smallest index $i$ s.t. the graph $G_i$ formed by the first $i$ edges has domination set of size $\leq k$

hardness comes from here.

$H_i = G_i^2$

Two vertices are adjacent iff $\exists$ a path of length $\leq 2$ between them.




Independent set in $G_i^2$ has size atmost $k$

If $G_i$ has a dominating set of size $k$, $G_i^2$ cannot have an independent set of size $>k$.

no vtx can be added to it. maximal indep set

Initially MIS has size $n$ → $n-1$

Find the smallest $i$ such that MIS in $G_i^2$ has size $\leq k$ → optimal radius is atleast cost($e_i$)

So, $G_{i-1}^2$ has MIS of size $> k$

($G_{i-1}$ doesn't have dominating set $\Rightarrow e_i$ must be there in radius)

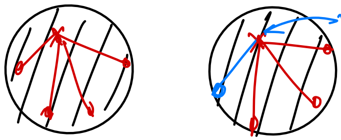
Cost($e_i$) is a lower bound on optimal solution.

We use the maximal ind set in $G_i^2$ to get a solution of cost atmost 2 cost($e_i$) $\leftarrow$ MIS is a dominating set $\Rightarrow \exists$ a path of length 2 in $G_i$, by $\Delta$ ineq $\leq 2$ cost($e_i$) of that edge.

Gives a dominating set in $G_i$ w/ distance $\leq 2R_{opt}$.

Subsets of cost $\leq W$

In $G_i^2$ find a maximal independent set, replace each vertex by its lowest weight neighbor if any (including itself) in $G_i$ → claim: Weight of modified set $\leq C$ if $\exists$ a dominating set with cost $\leq C$ in $G_i$



replaced by $\leq$ this.

So, after modification, weight $\leq W$

Now, $\exists$ a path of length $\leq 3 \cdot$ b/w centre, vtx.

$\Rightarrow$ 3-approximation. [Shmoys]

(but best bound for this algorithm) (not the best approx.)

Note : If $\exists$ a dominating set of size $\leq k$ in $G_i$ then all independent sets have size $\leq k$ in $G_i^2$

---

Lecture 29/1/24

## Travelling Salesman Problem (TSP)

Given a metric space, n points with distances, find a cycle passing through all the points having minimum total distance
[min. weight hamiltonian cycle]

A lower bound on this is given by minimum weight spanning tree

$$MWST \leq OPT$$

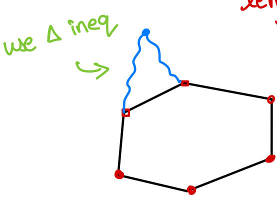
But, this turns out to be a 2-approximation i.e.

$$OPT \leq 2 \times MWST.$$

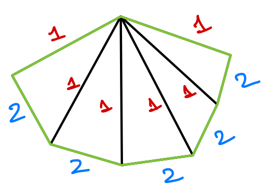
Proof : Take the min. wt. spanning tree and take vertices in order of of min wt spanning tree. Now, because of $\Delta$ ineq,

length of cycle $\leq$ length of pre-order traversal $= 2 \times MWST$

#ask: proof ?




Tightness




Opt-TSP $= 2 \times 5 + 2$

MWST $= 6$.

So, 2-approx is tight.

## Christofide's Algorithm

Add edges to the min wt. spanning tree to get an eulerian graph. Now, traverse the Eulerian graph to get a cycle by jumping over already visited edges.

1. Find min wt spanning tree
2. Look at the subset of vertices with odd degree (say $S$)
3. We need to add one edge to each vertex in $S$ to make the graph Eulerian.
   So, add a perfect matching in the graph formed by $S$ and add it to the tree.
   [We allow duplicate edges as well]
4. We find a min. wt. perfect matching in the graph formed by $S$ and add it to tree

This gives an Eulerian graph with weight = wt. of tree + wt. of matching
$\geq$ Cost of the hamiltonian cycle

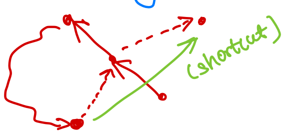
Now, wt. of perfect matching $\leq \frac{\text{Opt}}{2}$ (since Opt hamiltonian cycle can be split into two perfect matchings and choose the one with lower weight)

$\Rightarrow$ wt. of tree + wt of matching $\leq \text{Opt} + \frac{1}{2}\text{Opt} = \frac{3}{2}\text{Opt}$

$$\Rightarrow \boxed{A(I) \leq \frac{3}{2} \times \text{Opt}}$$

Hence, this gives a $\frac{3}{2}$-approximation for TSP.

Note: Finding an eulerian graph is enough since an eulerian tour can be converted to cycle using a skip of already visited edges/vertices.




Graphical Metric: (best approx. is 1.4)

Arbitrary undirected graph with unit weight edges,
$d_{ij}$ = weight of shortest path from $i$ to $j$

TSP for graphical metric is equivalent to a closed walk in the graph where each vertex is visited atleast once.

## Bottleneck TSP

- Find a hamiltonian cycle s.t. max weight of an edge in cycle is minimized
- Better than 2-approx not possible, else hamiltonian cycle can be reduced to this

Consider $K_n$ with $w_{ij} = \begin{cases} 1, & (i,j) \in E(G) \\ 2, & \text{otherwise.} \end{cases}$

Opt $= 1$ iff $\exists$ a hamiltonian cycle in $G$.
otherwise opt $= 2$.

### 2-approximation

$d(e_1) \le d(e_2) \le \dots \le d(e_m)$

$G_i$ = include first $i$ edges in graph.

Optimal at smallest $i$ for which $G_i$ has a hamiltonian cycle.

If $G_i$ has a hamiltonian cycle, it must be 2-vertex-connected [can't be disconnected by removing any one vertex]

2-connectedness can be checked by using DFS in linear time

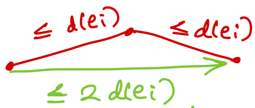
Now, find smallest $i$ for which $G_i$ is 2-connected

$\Rightarrow$ opt $\ge d(e_i)$ ← may or may not have a hamiltonian cycle. but previous $G_i$'s definitely don't.

Fleischner's Theorem : If $G$ is connected then $G^2$ has a hamiltonian cycle.

Hence, $G_i^2$ contains a hamiltonian cycle, and weight of any edge in $G_i^2$ $\le 2d(e_i)$ {$\Delta$ inequality} [wt of edge $(i,j)$ = length of shortest 2-hop]

Hamiltonian cycle in $G_i^2$ uses 2-hops in $G_i$




in original graph!

$\Rightarrow$ $\boxed{d(e_i) \le \text{Opt} \le 2d(e_i)}$

Hence, we get a 2-approximation for bottleneck TSP.

3-approximation

Stop as soon as $G_i$ is connected and for any tree $T$, $T^3$ contains a Hamiltonian cycle

**Steiner Tree**

Given a metric space and a subset $S$ of points, find the min. wt. tree that contains all points in $S$ (may / may not contain other points)

Minimum weight ST with vertices in $S$ is not optimal always.

Show that this gives a 2-approximation.

---

**Lecture** 31/1/24

**Scheduling**

- Minimizing lateness
- $n$ tasks, each task $i$ has a release time $r_i$ and execution time $t_i$, deadline
- One machine available, one task at a time, no task can be interrupted once started

Find a schedule (an order of executing tasks) to minimize max lateness over all tasks

Lateness = max (0, completion time − deadline)

- Deciding if $\exists$ a solution of lateness 0 is itself NP-complete

$\Rightarrow$ We cannot hope to get an approximation algorithm, since any such algorithm would have to output a solution of lateness 0.

So, we assume that each $r_i \geq 0$ and $d_i < 0$ (Ensures that objective function $> 0$)

**2-approximation**

Algorithm : Whenever the machine is free and a task is available, start executing any available task

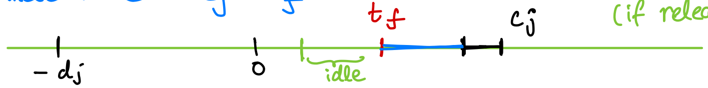
Consider task $j$ with maximum lateness [in the algorithm]. If it finishes at $c_j$, then lateness $= c_j + d_j$ (deadline is $-d_j$).

Let $t_f$ be the last time before $c_j$ when machine was idle for sometime just before $t_f$

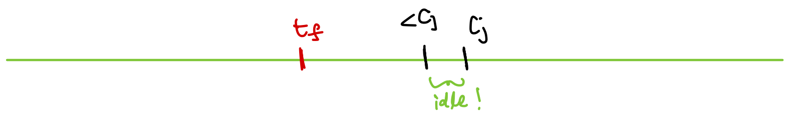
Let $S$ be the set of tasks executed in the time from $t_f$ to $c_j$

$\Rightarrow$ All tasks in $S$ have release time $\geq t_f$, and the total execution time of these tasks $= c_j - t_f$

(if release before $t_f$ could have started the task earlier!)




$\Rightarrow$ One of these tasks has to finish at time $\geq c_j$ in any scheduling




$\Rightarrow$ Since its deadline $\leq 0$, lateness in any schedule $\geq c_j$

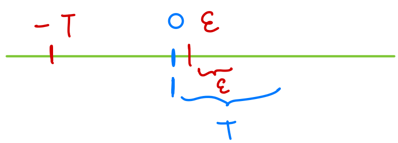
Also, in any schedule, the max lateness $\geq d_j$ (since release $\geq 0 \Rightarrow$ lateness$_j \geq d_j$)

$$\Rightarrow \quad A(I) = c_j + d_j \leq \text{opt} + \text{opt} = 2 \times \text{Opt}$$ (gives a 2-approx algorithm)

Tight example

| | | |
|---|---|---|
| $r_i$ | 0 | $\varepsilon$ |
| $d_i$ | 0 | $-T$ |
| $T_i$ | $T$ | $\varepsilon$ |

$\Rightarrow$ max lateness $= 2T + \varepsilon$




## Another version of Scheduling

$r_i \rightarrow$ Pre-processing time

$t_i$

$d_i \rightarrow$ Post processing time

$\rightarrow$ Can happen in parallel

Completion time = Time at which execution is completed + delivery time

Minimize max completion time

- Executing any available task gives a 2-approximation in this case as well

If the $j^{th}$ job has max completion time

$\Rightarrow$ In any schedule, some job finishes after $c_j$ and Opt $\geq d_j$     #ask : why ?

$\Rightarrow \quad A(I) \leq 2 \times \text{opt}(I)$

- $m$ identical machines
- $n$ tasks with $i^{th}$ task having execution task $t_i$

↳ Assign tasks to the machines to minimize the makespan

Makespan = max completion time of a machine

This is NP-hard even for 2 machines, since if makespan = $\frac{\sum t_i}{2}$ then it would be same as dividing set into a set into 2 with equal sums.

- $T_{opt} \geq T_{max} = \max(t_i)$
- $T_{opt} \geq \frac{\sum t_i}{m}$

} simple lower bounds.

## 2-approximation

- Select the tasks in any order. At $i^{th}$ step, assign task $t_i$ to the machine which currently has the least load [current finish time is min]
- If $t_j$ is the task that finishes last, then the corresponding machine had the least load before $t_j$ was assigned

$\Rightarrow$ All machines were executing until atleast $T - t_j$

$$\Rightarrow \quad T_{opt} \geq \frac{\sum t_i}{m} \geq \frac{m(T-t_j)+t_j}{m} = T - \left(\frac{m-1}{m}\right) t_j$$

$(\because T_{opt} \geq t_j)$

$$\geq T - \frac{m-1}{m} T_{opt}$$

$$\Rightarrow \quad \boxed{T_{alg} \leq \frac{2m-1}{m} T_{opt}}$$

## $\frac{4}{3}$-approx algorithm

[LPT : longest processing time first]

- Order the tasks in non-increasing order of execution time

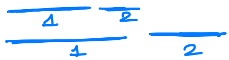
$$t_1 \geq t_2 \geq \cdots \geq t_n$$

Let $t_j$ be the last task to finish. We can assume that $t_j = t_n$, since otherwise we can delete all tasks after $t_j$, which does not change $T_{alg}$ and does not increase $T_{opt}$.

Case 1: $t_n > \frac{T_{opt}}{3}$

then, every task $> T_{opt}/3 \Rightarrow$ optimal schedule has atmost 2 tasks per machine. In such cases, LPT gives the optimal solution

Sort times for each machine in desc. order for opt solution.
$\Rightarrow$ LPT schedules first m largest times, followed by rest in next tasks in reverse order.




can swap to reduce makespan

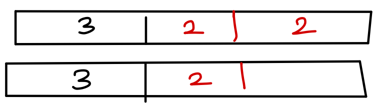
Case 2: $t_n \leq T_{opt}/3$

$$T_{opt} \geq \frac{\sum t_i}{m} \geq \frac{m(T - t_n) + t_n}{m} \geq T - \left(\frac{m-1}{3}\right) T_{opt}$$

$$\Rightarrow \boxed{T_{alg} \leq \left(\frac{4m-1}{3m}\right) T_{opt}}$$

For large m, this is a $4/3$ — approximation.

For $m=2$, $\frac{4m-1}{3m} = \frac{7}{6}$, which is attained by $T_i = (3, 3, 2, 2, 2)$




$T_{alg} = 7$


$opt = 6$

---

**Lecture** 1/2/23

## Scheduling identical machines

FPTAS if no. of machines is fixed, not part of input ($m$)

If execution times are bounded by $B$, $\exists$ a $(Bn)^m n$ time dp algorithm

$f(T_1, T_2, i)$ : true iff first $i$ tasks can be scheduled s.t. machine $i$ finishes at time $\leq T_i$

$$f(T_1, T_2, i) = f(T_1 - t_{i+1}, T_2, i) \vee f(T, T_2 - t_{i+1}, i)$$

Use this by scaling the execution times $\frac{\varepsilon T_{max}}{n}$

Modified execution time $= \left\lfloor \frac{t_i}{\frac{\varepsilon T_{max}}{n}} \right\rfloor \leq \frac{n}{\varepsilon}$

} scale only objective $f^n$ values using this.

So, find optimal to this in polynomial time. Use the same solution for original task

Every processor has atmost $n$ tasks $\Rightarrow$ increase in value by atmost $n \times \frac{\varepsilon}{n} T_{max}$ after scaling back

$\Rightarrow$ we get a $(1+\varepsilon) T_{opt}$ makespan.

If $m$ is part of input, no FPTAS possible since the problem is strongly NP-complete.

PTAS (poly in $n$, exp in $1/\varepsilon$ is ok.) #machines is also part of input

for $(1+\varepsilon)$ — approximation.

$k$ is fixed parameter depending on $\varepsilon$ $= \lceil \frac{1}{\varepsilon} \rceil$

Guess an optimum $T$.

Idea: Do binary search on parameter $T$.

↳ Either find a schedule with completion time $\le (1+\frac{1}{k}) T$.

(or) show there is no solution with completion time $\le T$

Consider jobs with $t_i \le \frac{T}{k}$ as small jobs.

↳ if $\exists$ a solution with completion time $\le T$, then scheduling small jobs greedily will give a solution with completion time $\le T(1+\frac{1}{k})$

Large jobs $t_i > \frac{T}{k}$

Each machine can execute atmost $k$ large jobs if it completes at time $\le T$

→ Scale the jobs by $\frac{T}{k^2}$, i.e. $t_i' = \left\lfloor \frac{t_i}{T/k^2} \right\rfloor$ ($t_i \le T$ o/w output no schedule with $\le T$ possible)

Now, maximum possible value of any large job is atmost $k^2$, atleast $k$

We have an instance s.t. every task has execution time b/w $k$ and $k^2$, each machine executes at most $k$ tasks.

| 0 | 1 | .... | $k^2$ |
|---|---|---|---|
| $n_1$ | $n_2$ | ... | $n_{k^2}$ |

$n_i$ = #tasks with execution time $i$, $0 \le i \le k^2$

The choice for each machine can be described by a similar vector

E.g. 0 1 5 6 ... ← valid configuration if total execution time in the modified instance is $\le k^2$ ← $\lfloor T/(T/k^2) \rfloor$

i.e. $(n_0 \; n_1 \; \cdots \; n_{k^2})$ s.t. $\sum i n_i \le k^2$

Find minimum number of machines to complete all the jobs.

There are only a constant number of choices for each machine configurations (since $k$ is constant)

$$\min(n_0, n_1, \ldots, n_{k^2}) = \min_{\substack{(m_0, \ldots, m_{k^2}) \\ \sum i m_i \le k^2}} (n_0 - m_0, n_1 - m_1, \ldots, n_{k^2} - m_{k^2}) + 1$$

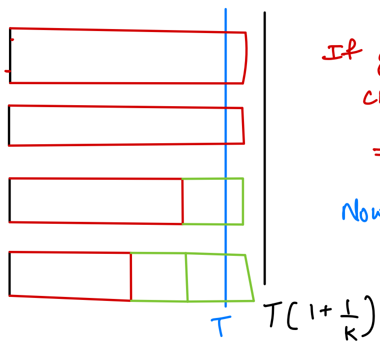
There are $\sim n^{k^2}$ possible vectors for inputs, but possible choices for machine bounded by a constant

This recurrence gives $\sim n^{k^2}$ time algorithm.

Error term $\le \frac{T}{k^2} \times$ #jobs per machine $= \frac{T}{k}$

Small objects $\left(t_i \le \frac{T}{k}\right)$

We have a solution for large objects with $\le T\left(1 + \frac{1}{k}\right)$




If greedy filling of green (small) objects crosses $T$ for all machines $\Rightarrow$ sum of times $\geq T$

$\Rightarrow$ no solution with makespan $T$ is possible.

Now, amount exceeded by filling greedily $\le$ size of last object crossing blue line $\le \frac{T}{k}$

$$\boxed{A(I) \le T\left(1 + \frac{1}{k}\right)}$$

Now, binary search over $T$, we get

initial $\rightarrow \left[\frac{\sum t_i}{m}, \frac{\sum t_i}{m} + T_{max}\right]$ (length $\sim T_{max}$)

Use binary search on this interval to get the approximate solution.

$$\text{Time} = \underbrace{\log(T_{max})}_{\#\text{bits in input}} (\ldots\ldots) \longleftarrow$$ so, we get Poly but not strongly polynomial algorithm.

Eg : Max cardinality subset s.t. sum of sizes $\le S$, sum of weights $\le W$ (reduce from subset sum $= k$)

$\rightarrow$ solve the LP, nuke fractional values.

$\max \sum x_i$

$\sum s_i x_i \le S$

$\sum w_i x_i \le W$

$0 \le x_i \le 1$

$\exists$ opt with atmost two fractional $x_i$

(if three fractional $x_i$, augment them to make one of them integer)

$$s_1 x_1 + s_2 x_2 + s_3 x_3 = S$$

$$w_1 x_1 + w_2 x_2 + w_3 x_3 = W$$

$\delta_1$ $\delta_2$ $\delta_3$

$s_1\delta_1 + s_2\delta_2 + s_3\delta_3 = 0$

$w_1\delta_1 + w_2\delta_2 + w_3\delta_3 = 0$

$\Rightarrow \exists \delta_1, \ldots, \delta_3 \neq 0.$

Gives an additive approximation.

but, running time depends on #bits in numbers for solving LP.

(not strongly polynomial time)

Change in cost $= \delta_1 + \delta_2 + \delta_3$. (if +ve ✓)

(if -ve, flip signs to get inc in cost)

$x_1 + x_2 < 2$

$\Rightarrow$ integral solution discards both

$\Rightarrow$ <u>additive approximation of 1</u> since $A(I) > Opt - 2$
$\ge opt - 1$.

---

Lecture 5/2

- n tasks, $i^{th}$ task has release time $r_i$ and execution time $t_i$

Single machine $\rightarrow$ one task at a time
$\rightarrow$ Task must be completed once started
[no pre-emption]

- Minimizing max finish time is easy $\rightarrow$ keep running any available task

Finish time $\ge$ max over all subsets $S$ of tasks $(r(S) + t(S))$

$$r(S) = \min_{i \in S} r_i, \quad t(S) = \sum_{i \in S} t_i$$

If T is the finish time of algo, look at the latest time before which the processor was free. Then no task executed after this was released before $\Rightarrow$ greedy is optimal

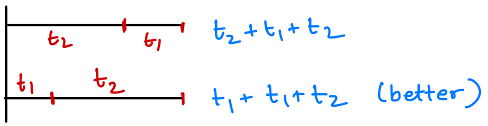
- Minimising the average completion time :

$$\sum_{\text{all tasks } i} C_i \longrightarrow C_i = \text{completion time of } i$$

- If no pre-emption is allowed, then the problem is NP-hard.

- Can be solved easily if pre-emption is allowed
  → Scheduled as per min. remaining time
  → Only need to check if a task finishes or a new task is released

At some point, if $t_1$ had the least remaining time, and $t_2$ was executed,




If we first schedule $t_1$ in these slots and then do $t_2$, total execution time increases

⇒ (min remaining time) is optimal

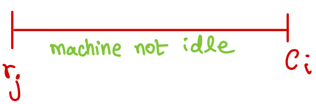
- Consider a non pre-emptive soln in which tasks are executed in the order in which they complete in the pre-emptive schedule. The task is executed as early as possible in this order.

Number the tasks 1 to n in the order in which they complete in the soln with pre-emption

If $C_i$ is the completion time here, and $C_i^*$ with pre-emption,

$$C_i = \max(C_{i-1}, r_i) + t_i$$

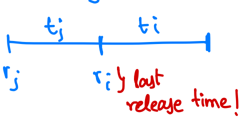
<u>Claim</u> : For every task $i$, $C_i \leq 2C_i^* \Rightarrow A(I) \leq 2\,\text{Opt}(I)$




Last release time after which machine isn't idle ($j \leq i$)

$\boxed{r_i \leq r_j}$

If $r_i > r_j$ then




We know $r_j \leq C_j^* \leq C_i^*$ (because of chosen order of exec.)

Also, $C_i^* \geq \sum_{j=1}^{i} t_j$ } best case scenario (all tasks run without machine being idle)

$$C_i = \max(C_{i-1}, r_i) + t_i$$

$$\leq \underset{\substack{\uparrow \\ \text{last release time after which not idle}}}{r_j} + \underset{\substack{\uparrow \\ \text{max running time}}}{\sum_{j=1}^{i} t_j} \leq C_i^* + C_i^* = 2C_i^*$$

## Weighted Completion time

$$\min \sum_{i=1}^{n} w_i C_i$$

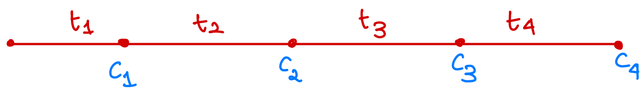
- Finding optimal schedule with pre-emption allowed is also NP-hard
- In this case, we formulate a linear program in which the completion times of the tasks are variables.

$C_i$ are now variables.

$C_i \geq r_i \quad \forall i$ [Can also use $C_i \geq r_i + t_i$]

We want to impose some ordering on the task s.t. completion time of the $i^{th}$ task is atleast the sum of execution time of the tasks that come before it.

Consider a subset $S$ of tasks,




$C_1 = t_1$
$C_2 = t_1 + t_2$
$C_3 = t_1 + t_2 + t_3$
$C_4 = t_1 + t_2 + t_3 + t_4$

$$\Rightarrow \sum_{i \in S} t_i \underbrace{C_i}_{\text{includes } t_j \ \forall j \leq i} \geq \frac{1}{2}\left(\sum_{i \in S} t_i\right)^2 + \frac{1}{2}\sum_{i \in S} t_i^2$$

$\Rightarrow$ includes $t_i t_j$ for every pair $(i,j) \in S$, and $t_i^2 \ \forall i$

$$\Rightarrow \sum_{i \in S} t_i C_i \geq \frac{1}{2}\left(\sum_{i \in S} t_i\right)^2 \quad \Rightarrow \quad \boxed{\sum_{i \in S} t_i C_i \geq \frac{(t(S))^2}{2}}$$

So, we get an LP,

$$\boxed{\begin{array}{l} \min \sum w_i C_i \\ C_i \geq r_i \\ \sum_{i \in S} t_i C_i \geq \frac{1}{2} t(S)^2 \quad \forall \text{ subset } S \text{ of tasks} \end{array}}$$

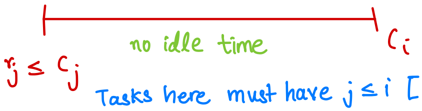
$\leftarrow$ exponentially many # constraints

- Even though there are exp. inequalities, there are solvers for this since in the dual, there exists an optimal solution where most of the vars are zero [ellipsoid method]
- If it is possible to check efficiently whether a given solution satisfies all constraints, then we pick a subset of inequalities, and check if the resulting soln satisfies all constraints.

If we solve the LP finding the optimal $c_i$ values ($c_i^*$), construct the schedule by executing in order of non-decreasing $c_i^*$. Let $c_i$ be the completion time in this schedule.

**Claim** : $C_i \le 3 C_i^* \;\forall i$

If machine was idle before $r$, then $r$ is the release time of some task




Tasks here must have $j \le i$ [ $C_j^* \le C_i^*$ ]

Consider $S$ to be the set of tasks with index $\le i$

$$\Rightarrow \quad C_i^* \underbrace{\sum_{j \in S} t_j}_{t(S)} \;\ge\; \sum_{\underbrace{j\in S}_{S = \text{first } i \text{ tasks}}} t_j C_j^* \;\ge\; \frac{1}{2} t(S)^2 \qquad \text{[feasible pt in LP]}$$

$$\Rightarrow \quad C_i^* \ge \frac{t(S)}{2} \quad \text{and} \quad C_i^* \ge r_j \quad (\because r_j \le C_j^* \le C_i^*)$$

$$\Rightarrow \quad C_i \le r_j + t(S) \le C_i^* + 2C_i^* \;\Rightarrow\; \boxed{C_i \le 3 C_i^*}$$

, hence gives a 3-approx algorithm

---

## Worst Case of Approximation

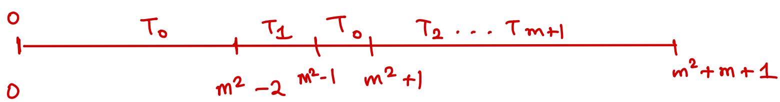
6/2/24

| | | | m tasks | |
|---|---|---|---|---|
| $r$ | $0$ | $m^2-2$ | $m^2+1$ | $m^2+m$ |
| $t$ | $m^2$ | $1$ | $1$ | |
| | $T_0$ | $T_1$ | $T_2 \dots T_{m+1}$ | |

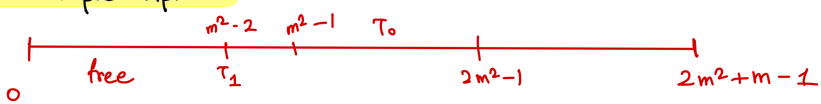
**Questions** :

1. Does $\exists$ a 3-approx tight example for weighted case
2. Is weighted LP exact ? Are all sol$^n$ in it feasible ?
3. Best bound on quality of the approx?

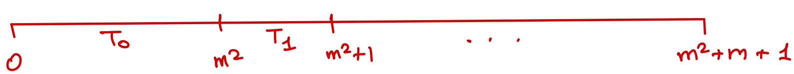
### pre-emptive schedule




### without pre-emption




### Opt




$A(I) \Rightarrow 2m^3 + \alpha m^2 + \beta m + \gamma$

$\mathrm{opt}(I) \Rightarrow m^3 + \alpha_1 m^2 + \beta_1 m + \gamma_1$

As $m \to \infty$, $\dfrac{A(I)}{\mathrm{opt}(I)} \to 2$

Hence, this is the worst case instance for the given algorithm. The optimal feasible soln must be close to the optimal relaxed soln but A(I) is far.

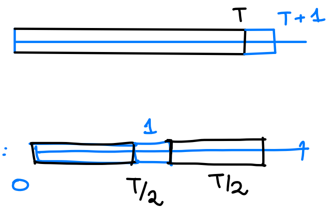
Quality of bound – instance where optimal feasible soln. is far from the optimal relaxed soln.

$$\text{Quality} = \text{opt}(I)/\text{relaxed-opt}(I)$$

| | | |
|---|---|---|
| r | 0 | T/2 |
| t | T | 1 |

Opt : (min remaining time)

Relaxed-opt :




$T + T+1 = 2T+1$

$T+1+\frac{T}{2}+1 = \frac{3T}{2}+2$

Hence, quality of bound $\geqslant 4/3$

Note : $\exists$ an example where we get $18/13$ in place of $4/3$

## LP relaxation for weighted scheduling

$$\min \sum_i w_i c_i$$

$$c_i \geqslant r_i \quad \forall i$$

For all subset $S$ : $\sum_{i\in S} t_i c_i \geqslant \frac{1}{2}(t(S))^2$

A separation oracle is an algorithm such that given a particular soln $c$, either shows that it is feasible or outputs a constraint that $C$ violates.

Ellipsoid method $\equiv$ Efficient separation oracle $\Rightarrow$ efficient soln. of LP.

Given values of $c_i$, do they satisfy all inequalities ?

- $c_i \geqslant r_i$ checked easily

Order the sets s.t. $c_1 \leq c_2 \leq c_3 \leq \cdots \leq c_n$

Consider the sets $S_i = \{1, 2, \ldots, i\}$

Claim: If the constraints are satisfied for these $n$ sets, then it implies that they are satisfied for all sets.

Proof : If there is a set $S$ for which the constraint is violated, then $\exists S_i$ for which it is violated.
(to show)

$$\sum t_i c_i < \frac{1}{2} t(S)^2$$

Suppose $j$ is some object in $S$, and we remove $j$ from $S$

$\Rightarrow$ LHS decreases by $t_j c_j$, RHS decreases by $\frac{1}{2}t(S)^2 - \frac{1}{2}(t(S) - t_j)^2$

$$= \frac{1}{2}(t(S)^2 - t(S)^2 - t_j^2 + 2t(S)t_j) = \frac{t_j}{2}(2t(S) - t_j)$$

If decrease in LHS $\geq$ RHS, constraint is still violated.

$$t_j c_j \geq \frac{t_j}{2}(2t(S) - t_j) \quad \text{iff} \quad \boxed{c_j \geq t(S) - \frac{1}{2}t_j}$$

$$\text{iff} \quad c_j \geq t(S - \{j\}) + \frac{1}{2}t(j)$$

Let $l$ be the largest index in $S$. If $c_l \geq t(S - \{l\}) + \frac{1}{2}t_l$ we can remove that index. Repeat until this does not hold.

Then,

$$c_l \leq t(S - \{l\}) + \frac{1}{2}t_l$$

↳ Show that if $S$ does not contain all objects from $1 \ldots l$, adding any missing object gives a set that still violates the constraint

$[\, c_j \leq c_l \,,\; t(S) - \frac{1}{2}t_j \geq t(S) - \frac{1}{2}t_l \,]$

<u>Note</u>: We can't remove all the useless inequalities since we don't know the order of $c_i$'s for Opt in general.

## Prize Collecting Steiner Tree

Graph → each edge has a +ve weight $[c_e]$

→ each vertex has a penalty, except a specified root $r$ $[\pi_i]$

Find a tree containing $r$ that minimizes sum of weights of edges in the tree + sum of penalties of vertices not in the tree

Find $\pi_i = \infty$, the optimal solution is the min. wt. spanning tree.

Steiner tree — Connect a specified set of vertices

↳ $\infty$ for terminals, $0$ for others in this problem gives a steiner tree.

Find a tree $T$ s.t.

$$\sum_{e \in T} c_e + \sum_{v \notin T} \pi_v \quad \text{is minimized}$$

Opt will be a tree

if not prune off cycles!

$y_i \rightarrow$ for each vertex $i$

$y_i = 1 \Leftrightarrow$ vertex $i$ is in the tree

$x_e \rightarrow$ for each edge $e$

$\forall S$ s.t. seperates $r$ from $v_i$

$$\sum_{e \in \delta(S)} x_e \geq y_i$$

$\delta(S) \rightarrow$ set of edges with exactly one endpoint in $S$

LP - relaxation

$$\sum_{\text{edges}} c_e x_e + \sum_{\text{vertices } i} \pi_i (1 - y_i)$$

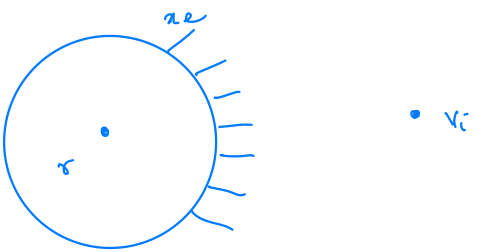
$\leftarrow$ not exact formulation.

$$\sum_{e \in \delta(S)} x_e \geq y_i \quad \text{where } S \text{ separates root } r \text{ from } v_i$$

} exp. many constraints

$$x_e \geq 0, \quad y_i \leq 1$$

Given a solution $x', y'$ it can be checked efficiently if it satisfies all constraints, and if not a violated constraint must be found

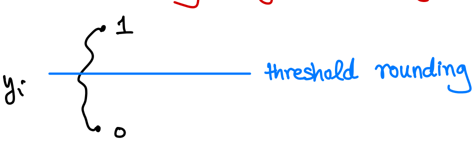



Take $x_e'$ values as capacity of edges and check that max flow from $r$ to $v_i$ is atleast $y_i'$.

Checking feasibility reduces to finding max flow. (strongly polynomial)

Hence, LP can be solved using ellipsoid algorithm. in polynomial - time

$\hookrightarrow$ rounding to get an integer solution



Threshold rounding : If $y_i \geq \alpha$, round it up to 1
otherwise, round it down to 0.

In this case, take $\alpha = \frac{2}{3}$

Include all vertices with $y_i \geq \frac{2}{3}$ in the tree

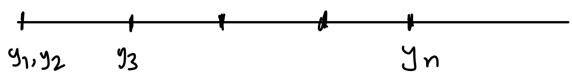
Penalty by algorithmic soln $= \sum_{i: y_i < \frac{2}{3}} \pi_i$

Penalty in opt soln $= \sum_i \pi_i (1 - y_i)$

$$\geq \sum_{y_i < \frac{2}{3}} \pi_i (1-y_i) \geq \frac{1}{3} \sum_{i: y_i < 2/3} \pi_i$$

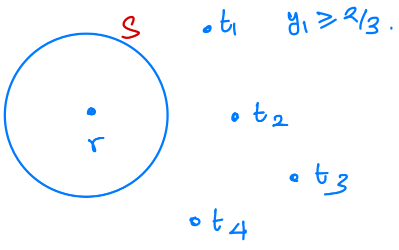
$$\geq \text{Algo penalty}/3$$

<u>Note</u> : Rounding has limited choices for $y_i$. Arrange in ↑ order and threshold choices are only values of $y_i$




Chosen the set of vertices of vertices to include

→ finds a steiner tree with r and selected vertices on terminal (has a 2-approx)




In LP solution for any selected vertex $t_i$ and any cut S separating r and $t_i$

$$\sum_{e \in \delta(S)} x_e \geq y_i \geq \frac{2}{3}$$

But for Steiner tree, we need $\sum_{e \in \delta(S)} x_e \geq 1$.

Multiplying each $x_e$ by $3/2$ gives a valid solution to the LP-relaxation of Steiner tree.

⇒ we can find a weighted steiner tree whose cost is $\leq 2 \times$ cost of LP-relaxation

i.e. cost of steiner tree $\leq \cancel{2} \times \frac{3}{\cancel{2}}$ times connection cost in the original LP.

⇒ Cost of solution is atmost 3 times cost of the LP solution.

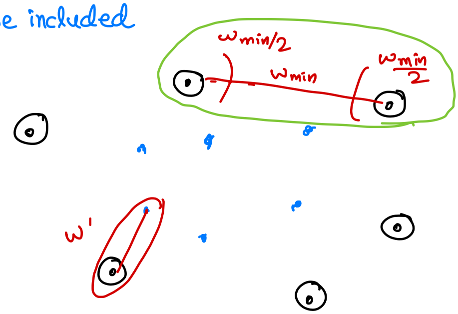
(MST is 2-approx)

## Steiner-tree approximation

(Primal-dual method for set cover)

Given a set of terminals in graph, find min-cost tree including all of them

- For any set $S$, separates some pair of terminals, atleast one edge in $\delta(S)$ must be included




Find a tree whose cost is atmost $2\times$ cost of dual

Increase dual variable for all singleton cuts by an equal amount $\delta$ till some edge becomes tight (constraint)

$$\sum_{\delta(S)\ni e} y_S \le c_e$$

Algorithm

1. Increase dual variable by $\delta$ for all singleton cuts

$$\delta = \min\left\{\frac{w_1}{2}, w_2\right\}$$

$w_1$ is min wt edge joining terminals,
$w_2$ is min wt edge joining terminal to Non-term.

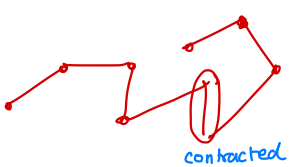
→ Dual cost increases by $\delta\,|T|$ ← terminals.

↳ Contract any edge that becomes tight
reduce weights of all edges incident with $T$ by $\delta$

→ Assume by induction you can find a tree in the remaining graph with cost atmost $2\times$ dual solution

↓

i.e. in each step, increase in cost $\le 2\times$ increase in dual

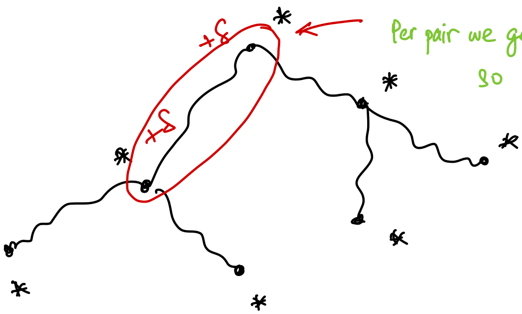



Increase weight of all edges incident with $T$ in tree by $\delta$ and add cost of contracted edge.

The tree obtained has only terminals as leaf vertices.

In any tree if $T \subseteq V$ including all leaves, then # edges incident with $T$ is atmost $2|T| - 2$ (easy to see)

Hence, increase in cost (primal) $\leq$ $2|T|\delta$, showing the 2-approximation.




Per pair we get $+\delta$, so for $|T|-1$ pairs (upon contracting edges with one non-term. end in arbitrary order) we get atmost $2\delta|T|$ increase.

## Facility location

12/2/24

- Set $F$ of facilities
- Set $D$ of clients
- Cost $f_i$ of opening a facility
- Cost $c_{ij}$ for connecting client $j$ to facility $i$

Find a subset of facilities and connect the client to the nearest open facility to minimize total cost

i.e. we need $\min\limits_{S \subseteq F} \left[ \sum_{i \in S} f_i + \sum_{j \in D} \min_{i \in S} c_{ij} \right]$

If $c_{ij}$ are arbitrary then set cover can be reduced to this

Greedy: Select a facility $i$ which minimizes

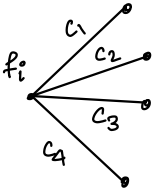
$$\min_{\text{over all non-empty subsets } S \text{ of } D} \frac{f_i + \sum_{j \in S} c_{ij}}{|S|}$$

→ open facility $i$ and connect all elements of $S$ to facility $i$

→ Then remove $S$ from clients and continue

This $S$ can be found by sorting $c_{ij}$

**Analysis** – For any facility $i$ and subset $S$ of clients,



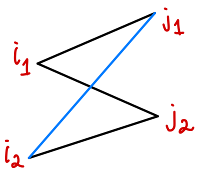

$$f_i + \sum_{j \in S} c_{ij} \geq \underbrace{\text{sum of costs assigned to elements in } S}_{H_{|S|}}$$

[same arguement as set cover]

first elem covered → cost assigned ≤ avg. cost

## Metric Facility Problem

- Facilities and clients are points in a metric space
- Connection cost is the distance between points




$$C_{i_2 j_1} \leq C_{i_1 j_1} + C_{i_1 j_2} + C_{i_2 j_2}$$

### LP – Formulation

$y_i$ → for facility $i$, $y_i = 1$ iff $f_i$ is open

$x_{ij}$ → client $j$ is connected to facility $i$

$$\min \quad \sum_i f_i y_i + \sum_{i,j} c_{ij} x_{ij}$$

$\sum_i x_{ij} = 1 \quad \forall j$ ← each client connected to 1 facility

$y_i - x_{ij} \geq 0 \quad \forall i,j$ ← If client $j$ connected to $f_i$, $f_i$ is open.

$x_{ij}, y_i \in \{0,1\}$ ← relaxation is just $x \geq 0, y \geq 0$

In dual, for each client we have a variable $v_j$ (unconstrained since constraint is an equality) and $w_{ij} \geq 0$

**Dual**:

$$\max \sum_{j \in D} v_j$$

$f_i \geq \sum_{j \in D} w_{ij} \quad \forall i$

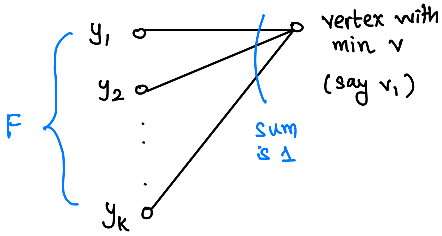
$v_j - w_{ij} \leq c_{ij} \quad \forall i,j$

$w_{ij} \geq 0$

Dual LP-opt = LP opt.

If a variable that is $\geq 0$ has +ve value in LP-opt then corresponding inequality in the dual must be tight [complementary slackness]

Consider optimal LP soln. to the facility location problem. Then if $x_{ij} > 0$, then we have $v_j - w_{ij} = c_{ij}$ 13/2/24

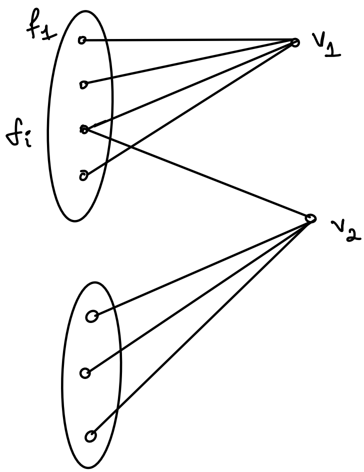



For facilities with $x_{ij} > 0$, Pick facilities with min $f_i$ (say $f_1$)

Dual opening cost of facilities in F

$$= \sum f_i y_i \geq f_1 \sum y_i \geq f_1 \sum x_{i1} = f_1$$

when $x_{ij} > 0$, $v_j = c_{ij} + w_{ij} \geq c_{ij}$




Connect $v_1$ to $f_1$ and also connect all clients with $x_{ij} > 0$ to $f_i$ $(x_{1j} > 0)$ (to $f_1$)

**Algorithm**

- While $\exists$ an unconnected client $j$, pick the client with min. value of $v$
- Let F be the set of facilities $i$ such that $x_{ij} > 0$ in the optimal solution
- Pick a facility $m$ in F with min. value of $f_i$
- Connect all clients (unconnected) to the facility $m$ such that $x_{ik} > 0$ for some $i$ in F
- Repeat this.

Note: After this, no unconnected clients have a non-zero $x$ value to any facility in F.

Claim: At each step, the total cost incurred in opening the facility and connecting the clients to it is atmost $4 \sum v$ values of clients connected.

Pf: The dual cost of opening facilities in F

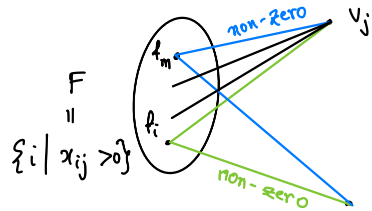
$(F = \{i \mid x_{ij} > 0\})$ $\sum_{i \in F} f_i y_i \geq f_m \sum_{i \in F} y_i \geq f_m \sum_{i \in F} x_{ij} = f_m$

$y_i > 0$ only if $x_{ij} > 0$ for some facility $i$, given client $j$. $f_m$ covers all clients covered by F, so only that is counted.

In this case, the total cost of opening the facility $f_m$ is atmost $\sum_{i \in F} f_i y_i$

$\Rightarrow$ Total cost of opening the facilities $\leq \sum_i f_i y_i \leq$ LP-Opt

By complementary slackness, $x_{ij} > 0 \Rightarrow v_j - w_{ij} = c_{ij} \Rightarrow v_j \geq c_{ij}$ (since, $w_{ij} \geq 0$)




(lowest v unconnected)

k ($x_{ik} > 0$, but $x_{mk}$ is not nec. $> 0$)

Connection cost of connecting j to m $\leq v_j$ ($c_{ij} \leq v_j$)

Cost of connecting k to m $\leq C_{mj} + C_{ij} + C_{ik}$ [metric]

$\leq v_j + v_j + v_k$

$\leq 3 v_k$ ($v_j$ was lowest cost)

$\Rightarrow \sum$ (connection costs) $\leq 3 \sum v_k$ (each $v_k$ cost is counted once due to cover by facilities)

$= 3 \text{ opt}(I)$ (dual opt is $\sum v_k$)

$\Rightarrow A(I) =$ connection cost + Opening Cost

$\leq 3 \text{ opt}(I) + \text{opt}(I)$

$\Rightarrow \boxed{A(I) \leq 4 \text{ opt}(I)}$

## Primal-dual Algorithm

Keep increasing the v values for some subset of clients till some dual inequality becomes tight [keep increasing all non-tight v together]

For each facility i, $\sum_j w_{ij} \leq f_i$, $v_j - w_{ij} \leq c_{ij}$ $\{\max \sum v_j\}$

Start with all $v_i = 0$, $w_{ij} = 0$

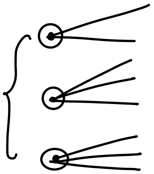
Assume we have a current dual solution. Define a facility i to be a neighbor of client j if $v_j \geq c_{ij}$. Also, a facility i contributes to client j if $w_{ij} > 0$.

If facility i contributes to j, then it is a neighbor of j (by construction)

Since we increase $w_{ij}$ only when the inequality becomes tight

The first iteration will stop when the dual inequality becomes tight for some facility

tight facilities



For a tight facility, the value of $\underset{\sim}{v}$ cannot be increased for any neighbor. $(w_{ij})$

There are 2 possibilities when we have to stop increasing the dual $(v)$ :

Stop increasing the dual only for this client ← 1. If some client becomes a neighbor of a tight facility $(v_j = c_{ij})$

Stop increasing the dual for all neighbors of this facility ← 2. Some new facility becomes tight

<u>Claim</u> : $\exists$ an integer solution whose cost is atmost 3 times the cost of the dual solution.

$S$ = all clients for which dual variable is being increased

17/2

$T$ = Set of all facilities for which dual inequality becomes tight

At the end, every client will be a neighbour of some facility that is tight (some facility in $T$)

(if $v_j < c_{ij}$ for all $i$, increase $v_j$ further)

For any $f_i \in T$, $\displaystyle f_i = \sum_{j \in N(f_i)} w_{ij} = \sum_{j \in N(f_i)} v_j - c_{ij}$

Suppose we connect all clients in $N(f_i)$ to $f_i$,

$$\underset{(\text{of opening } f_i)}{\text{Cost}} = f_i + \sum_{j \in N(f_i)} c_{ij} = \sum_{j \in N(f_i)} v_j$$

If $N(f_i)$ are disjoint, we get exactly

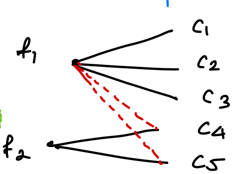
$$\sum_{j \in N(f_i)} v_j = \sum_{j \in D} v_j = \text{dual cost}$$

$\Rightarrow$ primal cost corresponds to optimal primal.

<u>Construction of solution</u> : (Intuition)

- Select any tight facility and open it
- Connect all neighbors of that facility to it $(v_j \geq c_{ij})$
- If one of these neighbors also has a non-zero $w_{ij}$ value for some other facility, connect all neighbors of that facility to this

[We don't pick 2 facilities which have a non-zero contribution to 2 clients ]




$\Rightarrow$ don't select $f_2$, connect $c_4$ and $c_5$ to $f_1$

<u>Note</u> : If these are no clients with $>1$ neighbors, then the algorithm gives the optimum value. (why?)

Note : we can't use the same arguement with the optimal LP dual solution , since we can't argue $v_k \le v_j$ in that case . [Ordering matters] .

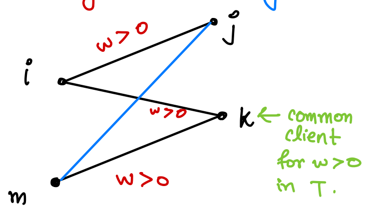
- Consider a graph on T in which two are adjacent if some client has non-zero value of $w_{ij}$ in both facilities

Let T' be any maximal independent set in this graph . Open all facilities in T'.

Any client that is a neighbor of any facility in T' ⇒ connect it to any one of them

For clients that are not in T' ⇒ Let i be the facility that caused the dual variable j to stop increasing

If i is not in T', ∃ a client k that has +ve w values to i and some facility in T' [say m]




$$C_{mj} \le C_{ij} + C_{ik} + C_{mk}$$

$C_{ij} \le v_j$ [they are neighbors]

$C_{ik} \le v_k$

$C_{mk} \le v_k$

Also, $v_k \le v_j$ because j was removed from S because i became tight

$w_{ik} > 0$ ⇒ k can't become neighbor of $f_i$ after it becomes tight.

$v_j$ can't increase beacuse $f_i$ became tight . Hence, $v_k \le v_j$
[by construction]

So k cannot increase after i became tight ⇒ k was removed either before or when i became tight

⇒ $v_k$ didn't increase after j stopped increasing

Hence, $C_{mj} \le 3 v_j$

$$\text{Total cost} = \sum_{f_i \in T'} \Big( f_i + \sum_{j \in N(f_i)} c_{ij} \Big) \le \sum_{f_i \in T'} \Big( \underbrace{\sum v_j}_{\text{direct conn.}} + 3 \underbrace{\sum v_j}_{\text{indirect conn.}} \Big)$$

$$\le 3 \sum v_j \le 3\, \mathrm{Opt}(I)$$

## Scheduling Related Parallel Machines [Ref: Vazirani]

- $n$ independent tasks
- $m$ machines
- For every task $i$ and machine $j$, $t_{ij}$ is the time taken by task $i$ on machine $j$
- Find a schedule that minimizes the max completion time → assigning tasks to machines

$$x_{ij} = \begin{cases} 1, & \text{task } i \text{ assigned to machine } j \\ 0, & \text{otherwise} \end{cases}$$

## ILP Formulation

$$\min \; t$$

$$\sum_i x_{ij} t_{ij} \le t \quad \forall \text{ machine } j$$

$$\sum_j x_{ij} = 1 \quad \forall \text{ task } i \text{, assign to exactly 1 machine}$$

$$x_{ij} \in \{0, 1\}$$

The problem in LP relaxation is that it can distribute large tasks over all machines so the LP-opt / Opt can be very small

**Eg**: One task, $m$ machines, $t_{1j} = m \;\; \forall j$

Optimal integral time $= m$

Optimal relaxed, $x_{1j} = \frac{1}{m}$, $\quad x_{1j} t_{1j} \le 1 \;\; \forall j$

↓

Opt relaxed $= 1$

Getting an $m$-approximate is easy – schedule each task to the fastest machine.

**Using Binary Search**: Guess a value $T$ for the optimal.

Construct an algorithm that either shows that there is no solution with completion time $\le T$ or finds one with completion time atmost $2T$.

Since $\text{Opt}(I) \in \left[ \frac{\sum \min_j t_{ij}}{m}, \; \sum_i \min_j t_{ij} \right]$, we can perform a binary search using this

Total time $\ge N^r \Rightarrow$ opt $\ge N^r/m$

a possible assignment

will give $m$-approximation

## Modified LP Formulation [modelling as LP feasibility problem]

whenever $t_{ij} > T \Rightarrow$ do not allow that assignment

$$\sum_j x_{ij} = 1 \quad \forall \text{ task } i \qquad \forall i,j : \; x_{ij} \ge 0$$

$$\sum_{i : t_{ij} \le T} x_{ij} t_{ij} \le T \quad \forall \text{ machine } j$$

Claim : If this LP has a feasible solution, then we can find an assignment with completion time atmost $2T$, and if not, there is no solution with completion time $\leq T$.

If there is a feasible solution, there is also a basic feasible soln which is obtained by selecting a set of constraints that are tight and solving the corresponding linear system of equations

Suppose there are $r$ valid pairs $(i,j)$ such that $t_{ij} \leq T$

# variables $= r$

# constraints $= m+n+r$

since there are $r$ variables

If there exists a feasible solution obtained by choosing atmost $r$ of these inequalities to be tight $\Rightarrow$ in any such set, atleast $r-(n+m)$ inequalities must be of the form $x_{ij} = 0$

$\Rightarrow \exists$ a feasible soln in which atmost $(m+n)$ variables are non-zero. $\leftarrow$ We are looking at rounding this feasible solution

("corner")

In the basic feasible soln, a task $i$ is fractionally assigned to machine $j$ if $0 < x_{ij} < 1$, and integrally assigned otherwise.

"corner"

If a task is fractionally assigned to one machine, then it must be fractionally assigned to atleast 2 machines

$\Rightarrow$ If there are $k$ fractionally assigned tasks and $(n-k)$ integrally assigned tasks, # non-zero $x_{ij}$ is atleast $(n+k) \leftarrow \underbrace{2k}_{\substack{\text{atleast} \\ 2\ x_{ij} > 0 \\ \text{per fractional task}}} + \underbrace{n-k}_{\text{integral}}$

$$\Rightarrow \underbrace{n+k}_{\#\text{non-zero } x_{ij}} \leq \overbrace{n+m}^{\max \#\text{non-zero } x_{ij}} \Rightarrow \boxed{k \leq m}$$

$\Rightarrow$ The fractionally assigned tasks can be integrally to machines such that every machines is assigned atmost one task.

So we can round some of the fractional values to 1 s.t. every machine gets atmost one rounded (to 1) value. } How to do this?

Next page

$\Rightarrow$ Each machine's execution time increases by atmost $T$

$\Rightarrow$ Actual completion time $\leq 2T$

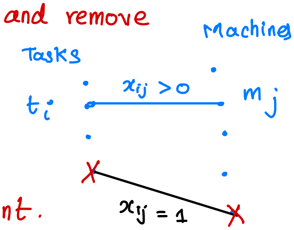
$\forall ij,\ t_{ij} \leq T$.
and LP ensures $\sum_i x_{ij} t_{ij} \leq T\ \forall j$ machine

Construct a graph with tasks, machines as edges where $(t_i, m_j) \in E$ iff $x_{ij} > 0$

( Rounding Algorithm )

$\Rightarrow$ Has $(n+m)$ vertices, atmost $\underbrace{(n+m)}_{\text{atmost } n+m \ x_{ij} \text{ are non-zero.}}$ edges

For tasks which are integrally assigned, just do that assignment and remove those tasks from the graph.




$\Rightarrow$ New graph also has no. of edges $\leq$ no. of vertices

If the new graph has a matching, then we get the required assignment.

If a machine has degree 1, remove that and the task adjacent to it.

Now we have a graph with deg $\geq 2$

In every connected component of this graph, we have $|E| \leq |V|$

$\Rightarrow$ Each connected component is an even cycle that has a perfect matching $\leftarrow$ pick alternate edges.

has min # edges given connected, $|E| = |V|$.

**Claim**: Any connected component of the graph has atmost as many edges as vertices

2012

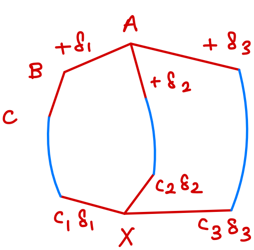
**Proof**: Choose a feasible solution with as few fractionally assigned as possible

[ we just need a basic feasible solution ]

Suppose we have n vertices and (n+1) edges $\Rightarrow$ There are atleast 2 cycles

The component has one of the following structures $\rightarrow$ Cases (a), (b)

(a)




If A is a task, for the new solution to be feasible, we need $\delta_1 + \delta_2 + \delta_3 = 0$.

Also, in this case, B is a machine and C is a task $\Rightarrow$ For $\sum t_{ij} x_{ij} \leq T$, BC needs to change by $-\delta_1 \dfrac{t_{AB}}{t_{CB}}$

We can similarly propogate the changes along the edges.

If X is a task $\Rightarrow c_1\delta_1 + c_2\delta_2 + c_3\delta_3 = 0$

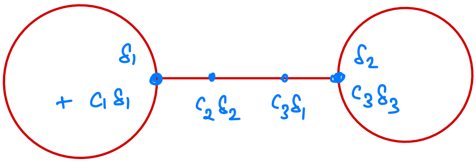
If X is a machine $\Rightarrow t_{1j} c_1\delta_1 + t_{2j} c_2\delta_2 + t_{3j} c_3\delta_3 = 0$

In either case, we have 2 equations in 3 vars and this has a solution. (other than (0,0,0))

Also, $(-\delta_1, -\delta_2, -\delta_3)$ also satisfies this

$\Rightarrow$ There is a solution that as a smaller no. of fractional edges.

(b)




Here also, we can propogate and get a solution.

#ask ?

---

## Vertex Cover

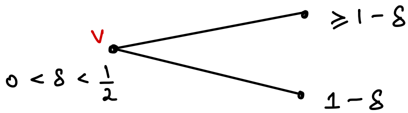
$$\min \sum_i w_i x_i$$

$$x_i + x_j \geq 1 \quad \forall (i,j) \in E$$

$$x_i \geq 0$$

Any basic feasible solution to this LP has only $\{0, \frac{1}{2}, 1\}$ as values for $x_i$'s [$\frac{1}{2}$-integral]

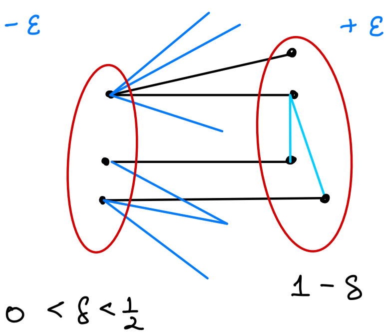
If any $x_i$ has weight $\delta \in (0, \frac{1}{2})$, then every neighborhood will have weight $(1-\delta)$




increasing # tight constraints.

Atleast one has $(1-\delta)$, else we can reduce $\delta$ (till some $e \in \delta(v)$ is tight)

Look at connected component with vertices having $\delta$ and $1-\delta$




[no internal edges since weight $< \frac{1}{2}$]

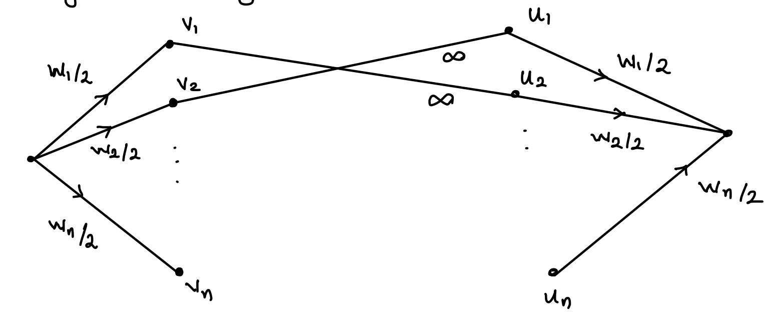
Increasing left side by $-\varepsilon$ and right side by $+\varepsilon$ is still feasible.

Also, for small enough $\varepsilon$, we can increase the left by $\varepsilon$ and decrease the right by $\varepsilon$

$\Rightarrow$ The soln is not basic

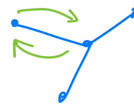
(since more constraints are becoming tight)

## Solving the LP using max-flow




Key . Opt is symmetric, so $y_{12} = y_{21} = y_e$ .

If $(i,j)$ are adjacent connect $(v_1, u_2)$ and $(v_2, u_1)$




Optimal LP soln = Value of max-flow $\Rightarrow$ $\frac{1}{2}$ -integer if $w_i$ are integers $\leftarrow$ ford fulkerson with 0.5 as units.

LP for max flow $\Rightarrow$

$$\max \sum y_e$$

$$\sum_{e \in \delta(v)} y_e \leq w_v \quad \forall v \in V$$

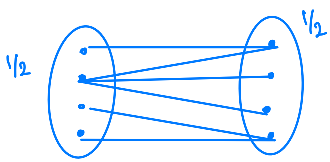
$$y_e \geq 0$$

dual of vertex cover LP

$\Rightarrow$ They have the same optimal value

**Rounding** : [ For bipartite graph ]

In the fractional solution first pick all vertices with integer cost . For the half integral weights,

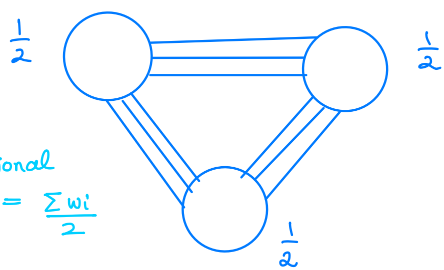



Total wt. = $\frac{\sum w_i}{2}$

$\Rightarrow$ Pick all vertices on the side with smaller weight

$\Rightarrow$ gives the optimal soln

For 3-colorable graphs ,




fractional opt = $\frac{\sum w_i}{2}$

Integral cost $\leq$ $\underbrace{\frac{2}{3} \sum w_i}_{\text{Picking 2 least weight sides}}$

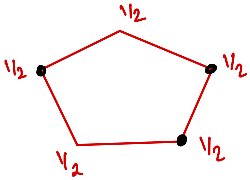
$\Rightarrow$ Atmost $\frac{4}{3}$ times the fractional cost

[ pick the 2 least weight sides ]

- For bipartite graphs,

  Optimal fractional soln = integral soln for an arbitrary weight function

The converse is also true, since if we have an odd cycle




$\Rightarrow wt = \frac{5}{2}$ but rounded is atleast 3

- Given a weight function, is it true that optimal integral = optimal fractional for that weight function? } don't know the answer

- Optimal integral vertex cover $\geq$ Optimal fractional vertex cover
  [NP-complete]
  $=$ Optimal fractional matching
  $\geq$ Optimal integral matching [Poly time]

Check if the two are equal to see if the solution is optimal.

---

**Midsem Question Pattern**

1. One set cover $\Rightarrow$ Needs LP
2. Another knapsack kind of $\Rightarrow$ probably DP

---

Lecture ( Midsem Discussion ) 6/3/24

2. c. LP formulation

$$\min \sum w_i x_i$$

$$\forall t : x_{e_1} + x_{e_2} + x_{e_3} \geq 1$$

$$x_{e_i} \geq 0$$

LP dual

$$\max \sum_e y_e$$

$$\forall e : \sum_{e \in t} y_t \leq w_e$$

$$y_t \geq 0$$

First, remove all edges that are not part of any triangle.

<u>Case 1</u> : $\forall_e \; x_e > 0$ in the optimal primal soln.

Because of compl. slackness, $\forall e$, $\sum_{e \in t} y_t = w_e$

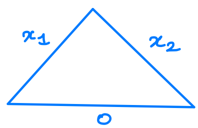
$$\Rightarrow \sum_e w_e = \sum_e \sum_{t \ni e} y_t = 3 \sum y_t$$

$$\Rightarrow \text{dual opt} = \frac{\sum w_e}{3}$$

Since complement of bipartite graph from (a) has weight atmost $\frac{\sum w_e}{2}$.

$$\Rightarrow \frac{3}{2}\text{-approximation}.$$

<u>Case 2</u> : $\exists_e : x_e = 0$




$\Rightarrow x_1 \geq \frac{1}{2}$ or $x_2 \geq \frac{1}{2}$

Round that $x_e \to 1$, remove the edge

$\Rightarrow$ In remaining graph, set of $x_e$ still feasible

$$\Rightarrow \text{Opt(new)} \leq \text{Opt} - \underbrace{\frac{w_e}{2}}_{\text{removed edge}}$$

Induction hypothesis

$$A(I_{new}) \leq 2\,\text{opt(new)} \leq 2\,\text{Opt} - w_e$$

Now, adding edge back

$$\Rightarrow A(I) \leq 2 \cdot \text{Opt}$$

## Max-SAT

Given $m$ clauses in $n$ boolean variables, each clause has a weight.
Find an assignment that maximises sum of weights of clauses that are satisfied.

1. Each clause contains atleast 1 literal
2. Does not contain literal and its complement
3. No literal is repeated

Goal: Find a solution whose expected cost is close to the optimum.

We pick any assignment uniformly at random

Let $X_i$ denote the random variable that takes value 1 if clause $C_i$ is satisfied, 0 otherwise

Cost is also a random variable: $\sum_{i=1}^{n} w_i X_i$

$$\Rightarrow \mathbb{E}[\text{cost}] = \sum_{i=1}^{n} w_i \mathbb{E}[X_i] = \sum_{i=1}^{n} w_i \Pr(C_i \text{ is satisfied})$$

$$= \sum_{i=1}^{n} w_i \left(1 - \left(\tfrac{1}{2}\right)^{|C_i|}\right)$$

and, $1 - \left(\frac{1}{2}\right)^{|C_i|} \geq \frac{1}{2}$

$$\Rightarrow \mathbb{E}[\text{cost}] \geq \frac{\sum_i w_i}{2}$$

$\Rightarrow \sum_i w_i \geq \text{max-cost} \geq \mathbb{E}[\text{cost}] \geq \frac{\sum_i w_i}{2}$, i.e., gives a $\frac{1}{2}$-approx [in expectation]

## Derandomization (method of conditional expectations)

$$\mathbb{E}[\text{cost}] = \frac{1}{2}\left(\mathbb{E}[\text{cost} \mid x_1 = 1] + \mathbb{E}[\text{cost} \mid x_1 = 0]\right)$$
$$\leq \max\left(\mathbb{E}[C \mid x_1 = 1], \mathbb{E}[C \mid x_1 = 0]\right)$$

here $x_1 = 1$ denotes value of $x_1$ set to 1.

Select $x_i = 1$ or $0$ s.t. $\mathbb{E}[C \mid x_i]$ is higher and continue.

Solution obtained has cost $\geq \frac{\sum w_i}{2}$ hence, gives a $\frac{1}{2}$-approximation.

Relabel the literals so that weight of a clause with a single negative literal $\leq$ wt of clause with a single positive literal. $\{ w(\overline{x_i}) \leq w(x_i) \ \forall i \}$ if $x_i \in C$ or $\overline{x_i} \in C$

Suppose no clauses with single negative literal. Suppose we set a variable to be true with $p > \frac{1}{2}$. What is the probability that a clause is satisfied?

For a clause with single literal (because it is +ve) $= p$

For clauses with $a$ +ve literals, $b$ -ve literals

$$\Pr(C \text{ is satisfied}) = 1 - (1-p)^a p^b \qquad (p > \tfrac{1}{2})$$
$$\geq 1 - p^{a+b}$$
$$\geq 1 - p^2$$

Choose $p$ s.t. $p = 1 - p^2$, $p = \frac{\sqrt{5}-1}{2}$

if $x_i \in \mathcal{C}$, $\overline{x_i} \notin \mathcal{C}$, $\Pr(x_i \text{ sat}) = p = 1 - p^2$

clause has $x_i, \overline{x_j}$, $\geq 1 - p^2$

clauses $x_i, \overline{x_i} \in \mathcal{C}$, cost $= p w_1 + (1-p) w_2 \geq$

$\Rightarrow \mathbb{E}[\text{cost}] = \left(\frac{\sqrt{5}-1}{2}\right) \sum w_i \quad \Rightarrow$ Better than $\frac{1}{2}$-approximation.

This can be derandomized in the same way as the previous algorithm.

## Max-Bipartite Subgraph [MAX-CUT]

Algorithm: Put a vertex in A or B with equal probability, include all edges between A, B

Expected cost $= \frac{\sum w_i}{2}$

Derandomization

Place $x_1$ in set which gives higher $\mathbb{E}[C \mid x_1]$.

## $(1 - 1/e)$ Approximation (using Randomized Rounding)

1. Interpret relaxed LP soln. as the probability that the variable takes value 1.
2. Compare expected cost with the optimal LP-cost

The ILP formulation of MAX-SAT is

$y_i \rightarrow 1$ if $x_i$ is true

$z_j \rightarrow 1$ if clause $j$ is satisfied

$$\max \sum w_j z_j$$

$$\sum_{i \in P_j} y_i + \sum_{i \in N_j} (1 - y_i) \geq z_j \quad \forall \text{ clause } j$$

$$z_j, y_i \in \{0,1\} \xrightarrow{\text{relax.}} 0 \leq z_j, y_i \leq 1$$

Consider the optimal soln. $y_i^*, z_j^*$ to this LP

$$\text{LP-cost} = \sum_j w_j z_j$$

Rounding scheme : set $y_i$ to 1 with prob. $y_i^*$ (independently)

Expected cost after rounding $= \sum_j w_j \Pr(\text{clause } j \text{ satisfied})$

$$\Pr(\text{clause } j \text{ not satisfied}) = \prod_{i \in P_j} (1 - y_i^*) \prod_{i \in N_j} y_i^*$$

$$\leq \left( \frac{\sum_{i \in P_j} (1 - y_i)^* + \sum_{i \in N_j} y_i^*}{l_j} \right)^{l_j} \leftarrow \text{no. of literals}$$

$$\leq \left( 1 - \frac{z_j^*}{l_j} \right)^{l_j}$$

$$= \sum_j w_j \left( 1 - \left( 1 - \frac{z_j^*}{l_j} \right)^{l_j} \right)$$

Need to show that $\Pr(\text{clause is satisfied}) \geq c z_j^* \geq c \times \text{opt-Lp cost}$

Since, $1 - \left(1 - \frac{z_j^*}{l_j}\right)^{l_j}$ is concave for $0 \leq z_j^* \leq 1$, using jensen

$$1 - \left(1 - \frac{z_j^*}{l_j}\right)^{l_j} \geq (1 - z_j)(0) + z_j \left( 1 - \left(1 - \frac{1}{l_j}\right)^{l_j} \right)$$

$$\geq \left(1 - \frac{1}{e}\right) z_j^*$$

$$\geq \left(1 - \frac{1}{e}\right) \sum_j w_j z_j^* \geq \left(1 - \frac{1}{e}\right) \text{Lp-opt} \geq \left(1 - \frac{1}{e}\right) \text{opt}.$$

If we take max of solutions obtained by setting each variable true with prob. $\frac{1}{2}$ by rounding the LP.

$$\max(E_1, E_2) \geq \frac{E_1 + E_2}{2}$$

Expected value of $E_1 = \sum w_j \left(1 - \left(\frac{1}{2}\right)^{l_j}\right) \geq \sum w_j z_j^* \left(1 - \left(\frac{1}{2}\right)^{l_j}\right)$

Expected value of $E_2 \geq \sum w_j \left(1 - \left(1 - \frac{1}{l_j}\right)^{l_j}\right) z_j^*$

$$\Rightarrow \quad \frac{E_1 + E_2}{2} \geq \sum \frac{1}{2} w_j z_j^* \left(2 - \left(1 - \frac{1}{l_j}\right)^{l_j} - \left(\frac{1}{2}\right)^{l_j}\right)$$

$$\geq \frac{3}{4} \text{LP-opt} \qquad (l_j \in \mathbb{N})$$

Gives a $\frac{3}{4}$-approximation algorithm.

The same bound can be obtained by rounding the solution differently

Round a variable $y_i^*$ to 1 with prob. $f(y_i^*)$

Choose $f$ s.t.

$$1 - \frac{1}{4^y} \leq f(y) \leq 4^{y-1} \qquad [\forall y \in [0,1],\ 1 - 4^{-y} \leq 4^{y-1}]$$

$$\Pr(\text{clause } j \text{ not satisfied}) = \prod_{i \in P_j} (1 - f(y_i^*)) \prod_{i \in N_j} f(y_i^*) \leq \prod_{i \in P_j} 4^{-y_i^*} \prod_{i \in N_j} 4^{y_i^* - 1}$$

$$= 4^{-\left(\sum_{i \in P_j} y_i^* + \sum_{i \in N_j} (1 - y_i^*)\right)} \leq 4^{-z_j^*}$$

$$\Pr(\text{clause } j \text{ satisfied}) \geq 1 - 4^{-z_j^*} \geq \frac{3}{4} z_j^* \quad (\text{jensen})$$

We can show that $\Pr(\text{clause } j \text{ is satisfied}) \geq \frac{3}{4} z_j^*$

$\Rightarrow$ Expected cost $\geq \frac{3}{4} \times \text{Opt-Lp}$

Note: Integrality gap of this is $3/4$, so this is the best possible approximation using this LP.

Example for $3/4$: $x_1 \vee x_2$, $\bar{x}_1 \vee x_2$, $x_1 \vee \bar{x}_2$, $\bar{x}_1 \vee \bar{x}_2$

wt = 1 per clause

ILP opt = 3

LP opt = 4 $\quad (y_1^* = y_2^* = \frac{1}{2})$