

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 solⁿ it can be verified in polynomial time if it's actually a valid solⁿ, cost can be computed in poly time.

Trivial: Enumerate over solⁿ, so v. much decidable problems.

But, bottleneck = time.

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

Finding optimal solution is NP-hard. \Leftarrow

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

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

- ① n objects. i^{th} object size s_i
Two bags with capacity C each.
Find maximum # objects in the two bags.

NP-hard opt problem.

Alg: Order objects so that $s_1 \leq s_2 \leq \dots \leq s_n$

At i^{th} step put object i in bag 1 if fits, o/w put in bag 2 o/w

discard all remaining objects.

} $\xrightarrow{\text{claim:}}$ gives at least $\text{OPT} - 1$

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

$\text{opt}(I)$ = opt " — " I

Bound on $|A(I) - \text{opt}(I)|$ \leftarrow Absolute error

Bounded ideally by a "constant".

$\frac{|A(I) - \text{opt}(I)|}{\text{opt}(I)}$ \leftarrow Relative error

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

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

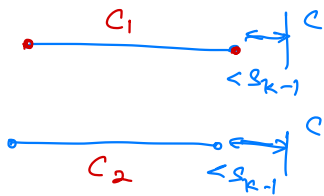
take objects greedily.

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

instead of using the $\text{opt}(I)$ which is unknown bound error by comparing to upper bound for max, lower bound for min problem

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

$$\sum_{i=1}^k s_i \leq 2C.$$

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 = Δ



$$\text{opt}(I) \geq \Delta$$

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

Deciding if $\text{opt} = \Delta$ 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 $A(I) \leq \text{opt}(I) + 1$ for all I unless $P = NP$.

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

give input = two copies



Idea:

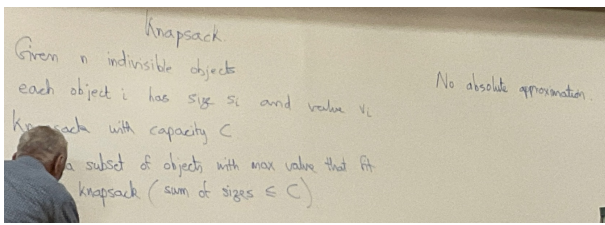
Construct instances where next suboptimum solution is far away from optimum.

Approx \Rightarrow exact.

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

$A(I) \leq \text{opt}(I) + \frac{n}{2}$ ✓ (using max-match) Question not studied well for additive.

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



(Amplification of gaps)

8/1/24

If \exists k -constant approx

\exists subset with val $\geq V$

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

yes $\rightarrow \text{OPT} \geq (k+1)V$

no $\rightarrow \text{OPT} \leq (k+1)(V-1) = (k+1)V - (k+1)$

Run approx algo on I'

\Rightarrow Ans to I is yes iff the solⁿ 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 arguments.

Special case of Integer linear program

$x_i \rightarrow$ takes value 0 or 1

indicates whether i th 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.

{indicators using bits}

variables restricted to be 0, 1 lin constraints and objective.

\hookrightarrow LP relaxation (pretty bad. high density high size objects)

$$0 \leq x_i \leq 1$$

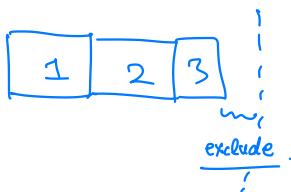
Allow variables to take fractional values between 0 and 1

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 LP so that integrality gap is less.

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



$$\begin{array}{ccc} v & 2 & C \\ s & 1 & C \end{array}$$

Cap = C

Greedy: value = 2
opt = C.

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

(fractional object
can contribute at most v_i)

$$\text{opt} \leq A + v_i \quad (v_i \gg A)$$

Issue!

Fix?

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

Output = larger of these two solutions.

Still

$$\text{opt} \leq A + v_i$$

$$A \geq v_i$$

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

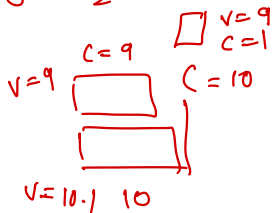
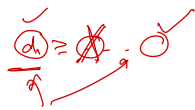
An algorithm for a maximization problem is an α -approx. algorithm if for every instance I ,

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

For a minimization problem,

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

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



$$\begin{array}{cccc} v+8 & v+8 & 2v & 2v \end{array}$$

$$\begin{array}{cccc} \frac{C}{4} + \epsilon & \frac{C}{4} + \epsilon & \frac{C}{2} - \epsilon & \frac{C}{2} - \epsilon \end{array}$$

$$\left\{ \begin{array}{l} \text{opt} \sim 4v \\ A \sim 2v \end{array} \right.$$

Further improvements?

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

1. Take all possible subsets of size at most k which are feasible & select these objects.
"cardinality" (wlog)
2. For remaining objects remove objects with value $\geq \frac{\text{min value of } k \text{ objects}}{\text{capacity} \geq \text{rem. cap.}}$, or

If opt has $\leq k$ 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 $\text{opt} \setminus k \cup \underbrace{v_i}_{\substack{\text{value} \leq \frac{\text{opt}}{k+1}}}$ is feasible.

As

$$\text{opt} \leq A + v_i \quad \leftarrow \text{fractional object in greedy LP bound solution.}$$

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

lecture

9/1/24

Polynomial-time approximation scheme

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

Minimization: $A(\epsilon)(I) \leq (1+\epsilon) \text{opt}(I)$ for any instance I

Maximization: $A(\epsilon)(I) \geq (1-\epsilon) \text{opt}(I)$

$$\text{opt} \leq A + v_i \quad \leftarrow \text{fractional object in greedy LP bound solution.}$$

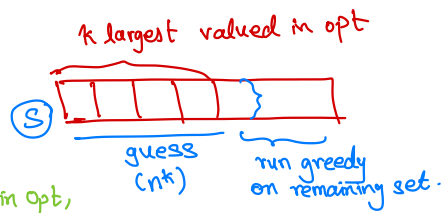
↓

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 at most k that are feasible (sum of sizes $\leq C$)



1) Select all objects in S \leftarrow If $\leq k$ objects in opt , we get optimum.

2) Remove all objects whose value $>$ the min value of S \leftarrow when $S = k$ largest valued objects in optimum, this step doesn't change opt .

3) For remaining objects fill using greedy

Output will be the best solution over all choices of S

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

greedy solution $> opt$ - value of fractional obj. $\geq opt \cdot \frac{k}{k+1}$

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

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

$\underbrace{\left(\frac{C}{k+1}, \dots, \frac{C}{k+1}, \epsilon \right)}_{k+1 \text{ objects}}$ \leftarrow tight example for approximation ratio.

Value $\left(\frac{C}{k+1}, \dots, \frac{C}{k+1}, 2\epsilon \right)$

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

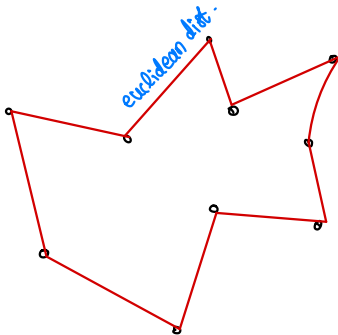
<u>Alg</u>	<u>opt</u>
$\frac{k}{k+1} C + 2\epsilon$	C

Euclidean TSP

[Sanjeev Arora]

\downarrow
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)

PTRAS \geq FPTAS.

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

our earlier alg was $O(\frac{1}{\epsilon} n^{1/\epsilon + 1}) \leftarrow$ not an FPTAS.

$$\begin{array}{ccccccc} S_1 & S_2 & S_3 & \dots & S_n \\ v_1 & v_2 & \dots & \dots & v_n \end{array}$$

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 $0 \leq v \leq Bn$ find smallest size subset whose value is $\geq v$.

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

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

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

Running time = $O(Bn^2)$

$$\left. \begin{array}{ccccccc} S_1 & S_2 & S_3 & \dots & S_n \\ v_1 & v_2 & \dots & \dots & v_n \end{array} \right\} \begin{array}{l} \text{Reduce values of these} \\ \text{objects by scaling them with} \\ \text{an appropriate factor} \end{array}$$

$$v_i' = \left\lfloor \frac{n v_i}{\epsilon v_{\max}} \right\rfloor \quad \left. \begin{array}{l} B \text{ is} \\ \text{upper bounded by} \\ \text{polynomial in } n. \end{array} \right\} \begin{array}{l} \text{max} \\ \text{value of an object.} \end{array}$$

$$= \left\lfloor \frac{v_i}{\left(\frac{\epsilon v_{\max}}{n}\right)} \right\rfloor \quad \leftarrow \begin{array}{l} \text{number of} \\ \text{these units.} \end{array}$$

Question : How bad is the approximation?

$$v_{i' \max} = \frac{n}{\epsilon}$$

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

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

$$\text{Hence, amount lost in total} \leq n \times \frac{\epsilon V_{\max}}{n} \\ = \epsilon V_{\max} \leq \epsilon \text{ Opt.}$$

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

↑
modified
instance

$$\geq (1 - \epsilon) \text{Opt}(I)$$

lecture

11/1/24

Set Cover

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

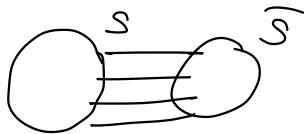
Cost = the sum of weights.

choose min cost solⁿ (NP-hard. reduce from vertex cover)

Min. wt spanning tree \leftarrow special case (PTIME solvable)

Connected subgraph with minimum weight

Need to cover all cuts in a graph



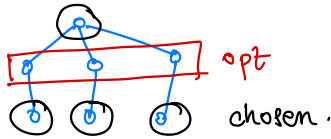
↑ at least one included in spanning tree.

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



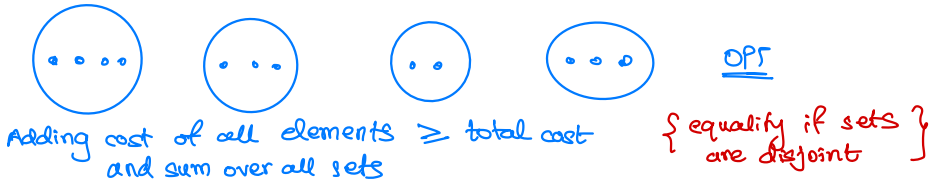
Hence, not an optimum algorithm clearly.

How bad is this algorithm? [Analysis]

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

To each of the new elements covered.

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



for any set S in C , let cost of $S = \text{sum of costs of elements } S$

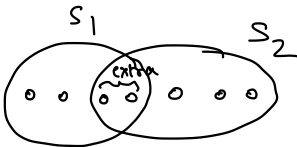
$A(I) = \text{sum of costs of all elements}$

For any optimal solution C'

$\text{sum of costs of sets in } C' \geq \text{sum of costs of all elements}$
 $= A(I)$

Idea: Cost is \leq much more than the weight

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



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

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

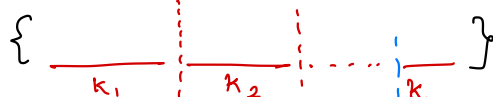
$\{x\}$ ratio = $\frac{w}{1}$. This is smallest or something else covers x with a smaller ratio.

i.e. $\text{Cost} \leq w$.

$$\text{cost}(S) \leq w(S) \left(1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k} \right)$$

why?

Divide into groups based on when it got covered



$$\text{cost} \leq \frac{w}{k}$$

$$\begin{aligned} \text{remaining} &= k - k_1 \\ \text{cost} &\leq \frac{w}{k - k_1} \end{aligned}$$

in the r th iteration.

$$\text{cost} \leq \frac{w}{k_r}$$

Total cost of all elements

$$\leq w(S) \left(1 + \frac{1}{2} + \dots + \frac{1}{k} \right)$$

$H_k = k$ th Harmonic number $= \ln(k)$

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

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

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

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

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

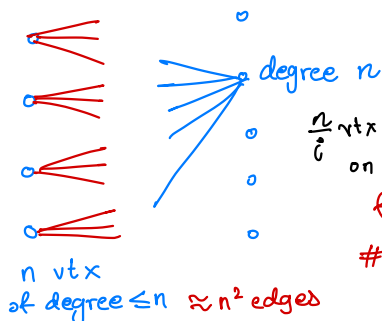
If every set has size at most k , $H_k \leq \ln(k)$

$$\left[\sum_{S \in \mathcal{C}'} w(S) \right] H_k \geq \text{ALG}$$

$$\Rightarrow \boxed{\text{ALG} \leq \text{OPT} \times H_k} \leftarrow \text{In general not a constant factor approx}$$

$k \leq n$ so H_n - approximate algorithm

Idea: only RHS vertices are chosen!



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

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

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

$U = \{\text{set of edges in an undirected graph}\}$ each vertex has a weight.

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

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

→ every element in U contained in exactly 2 sets

More general case, every element is contained in at most f sets, for some constant f

LP 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 , $|U| = n$, $|C| = m$

$$\sum_{j, e \in S_j} x_j \geq 1$$

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

LP relaxation

drop $x_i \leq 1$ without changing optimum

$$0 \leq x_i$$

$$\sum_{j, e \in S_j} x_j \geq 1$$

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

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 at most f sets

(For vertex cover, there are exactly two)

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

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

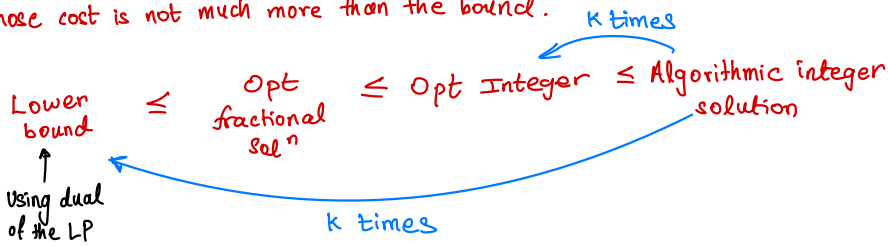
Gives a valid solution, and cost is multiplied by at most f .

$$x_i' \leq f x_i$$

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

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.



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

$$y_e \geq 0$$

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

Add up all of these.

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

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 at most 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.

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

↑ y_e until some dual inequality becomes tight for some set

→ Include all such sets in the set cover.

Repeat till all elements have been covered

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

Compare cost of integer solution to the dual solution.

$$\begin{aligned} \text{Cost of solution obtained} &= \sum_{\substack{\text{for all } S \\ S \text{ in solution}}} w(S) \stackrel{\text{dual constraint tight}}{=} \sum_{\substack{S \text{ in} \\ \text{solution}}} \left(\sum_{e \in S} y_e \right) \leq f \sum_e y_e \\ &\quad \uparrow \\ &\quad \text{A particular } y_e \text{ appears at most } f \text{ times} \end{aligned}$$

cost of dual

Gives an f -approximation

Idea: Bound # sets with $y_e \neq 0$

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

Shortest Path

Reverse deletion

↓ Set cover obtained by greedy algorithm may contain redundant sets

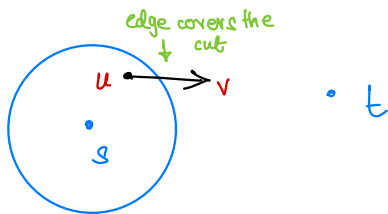
In reverse order in which the sets were selected,

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 .



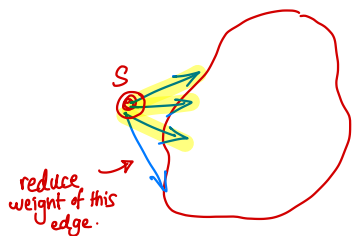
Elements of universe are s - t cuts, sets correspond to edges and an edge (u, v) covers all cuts s.t. $u \in S, v \notin S$

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

Dual variables \equiv cuts
inequalities for edges.

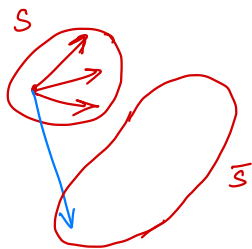
$$\sum_{e \text{ covers } S} y_S \leq w_e$$

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 .

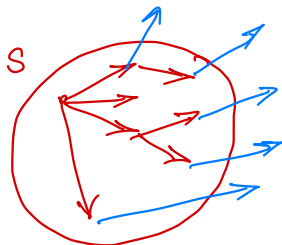


Increase the dual for this cut

$$S = \{s\} \cup \{\text{all vertices nearest to } s\}$$

← update step in dijkstra's algorithm

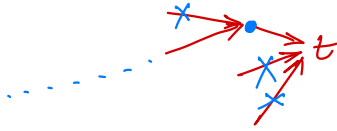
Increase in dual variable will be equal to difference in distances between vertices at level 2 and level 1.



and, repeat the procedure.

Continue till you get an s - t path

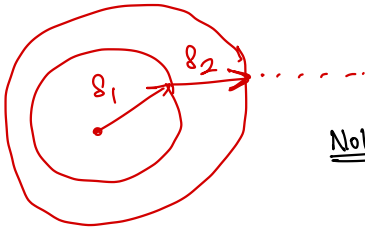
Applying reverse deletion, delete all redundant edges, till you are left with a single path.



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

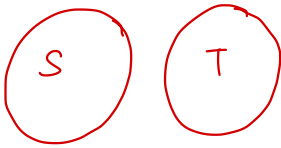
Weight of the path = total increase in dual variables

\Rightarrow optimal fractional solution \Rightarrow optimal integral solution.



Note: 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 .

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

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

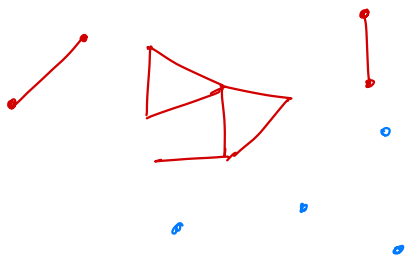
$u \xrightarrow{\quad} v$ covers all partitions with u, v in different parts. 2^{n-2} different such partitions

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

$$\sum_{e \text{ covers } s} y_s \leq w_e$$

If everything inc. by δ , 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.

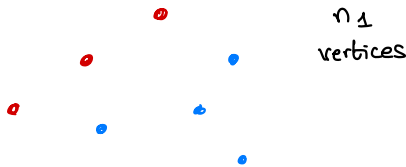


Already increased dual cost by

$$S_1 = \frac{w_{\min}}{2^{n-2}} \text{ for all cuts.}$$

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

Consider this as a smaller graph



$$\text{Effective weight} = w_{\text{original}} - \text{dual cost already assigned}$$

$$= w_{\text{original}} - w_1$$

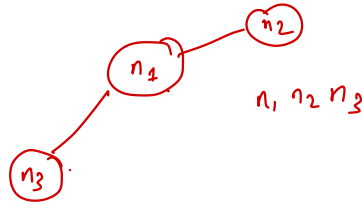
Pick minimum weight edges again.

(reverse deletion)

Finally, use a tree in each connected component \leftarrow gives a min weight ST.

So, Kruskal algorithm \equiv primal dual set cover algorithm.

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



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

Every cut is covered by atleast two edges

\rightarrow min wt 2-edge-connected subgraph

\rightarrow general problem.

NP-hard

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

Network Design

[ref: vazirani]

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

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

Integrality Gap

Complete graph will all edges of weight 1.

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

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

a valid fractional solⁿ with cost $= \frac{n}{2}$

Integer optimal $= n-1$



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

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

worst case.

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

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

Integrality gap of an LP relaxation of an ILP problem

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

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

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

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.

Include all min weight edges.

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

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

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 = $(n-1) w_{\min}$ for 1st step.
edges in tree

Excess cost = +ve.

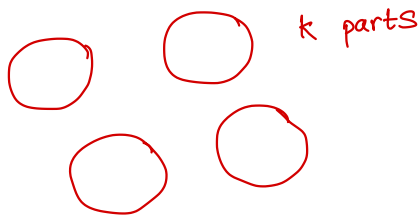
Opt : $(n-1) w_{\min}$

Dual cost : $\frac{n w_{\min}}{2}$

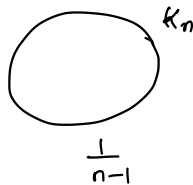
2 approx in each step.

Idea : Add more constraints to LP which eliminate fractional bound.

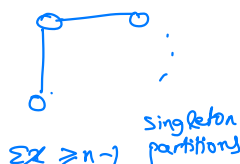
[Generalises isolating cuts]



Example

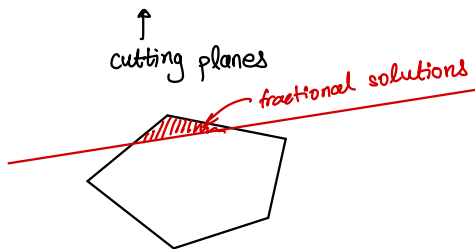


Violation



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



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 $(\forall e \in E, \sum_{\text{partition} \ni e} y_{\text{partition}} \leq w_e)$

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

All min wt edge tight.

$$\uparrow \text{Y parti.} \left(\sum_{e \in \text{partition}} x_e \geq n-1 \right)$$

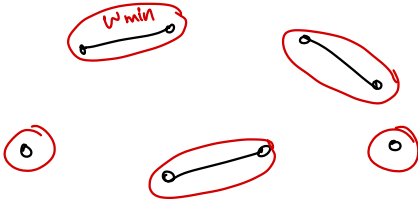
Dual cost = RHS $\times \delta$,

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

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

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

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



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

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

$$\left. \begin{array}{l} \sum_{e \in \text{cut}} x_e \geq 2 \\ 0 \leq x_e \leq 1 \end{array} \right\} \begin{array}{l} \text{Has an integrality} \\ \text{gap of 2.} \end{array} \leftarrow \begin{array}{l} \text{2-approx algorithm} \\ \text{Problem is NP-hard} \end{array}$$

(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 $\{ \geq \text{Vertex Cover} \}$

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.

$U = \{\text{cycles in the graph}\}$

$S_i \rightarrow$ corresponds to a vertex v_i
it covers all cycles $\ni v_i$

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

$$\text{every cycle } C \sum_{v \in C} x_v \geq 1$$

$$\min \sum w_v x_v$$

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

Construct an example where optimal integer solⁿ \uparrow , fraction solⁿ \downarrow

For all $g \geq 3$ there exists a ^{3-regular} cubic graph with 2^g vertices and no cycles has length less than g . ($g = \text{girth of the graph} = \text{length of smallest cycle}$) # To do: show construction

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

Optimal integral solⁿ contains $\frac{2^g}{4}$ vertices at least \leftarrow counting argument.

FV set of size k , then $n-k$ vertices don't have a cycle
 $\Rightarrow n-k-1$ edges left at most

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

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

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

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

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

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

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

each e_i has weight w_i

Maximum (NP-hard)
coverage

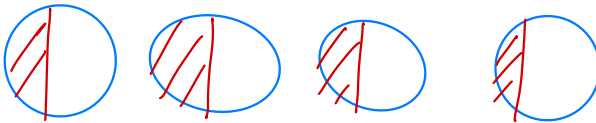
$$C = \{S_1, S_2, \dots, S_m\}, S_i \subseteq U$$

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)

at least one set $wt \geq \frac{W_{\text{opt}}}{k}$

{first step
of greedy}

$(i+1)^{\text{th}}$ 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

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

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

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

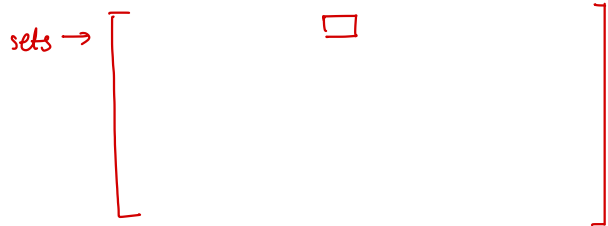
$$\Rightarrow W_{\text{opt}} - w_{i+1} \leq \underbrace{\left(1 - \frac{1}{k}\right)}_{\text{gap at } i^{\text{th}} \text{ step reduced by } 1 - \frac{1}{k}} (W_{\text{opt}} - w_i)$$

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

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

Float optimization problem

elements



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

Choose a set S of k rows such that

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

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

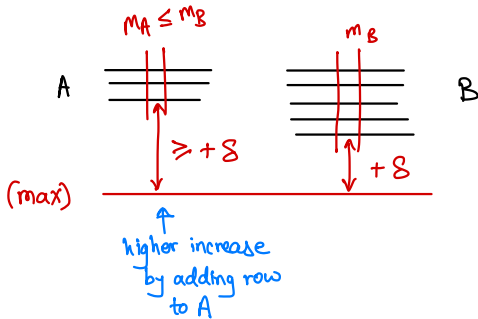
Exact same argument as before works, we get $1 - \frac{1}{e}$ approx algorithm using a greedy approach.

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

1. $f(S)$ is monotone: if $S_1 \subseteq S_2$, $f(S_1) \leq f(S_2)$

2. Submodular: If $A \subseteq B$ and $x \notin B$ then $f(A \cup \{x\}) - f(A)$

$$\geq f(B \cup \{x\}) - f(B) \quad [\text{decreasing marginal return}]$$



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

$$opt = \{v_1, v_2, \dots, v_k\}$$

This gives an increase of at least $w_{opt} - w_i$

$$w_i \leftarrow S_i$$

adding all rows in opt to S_i gives weight $\geq w_{opt}$ (monotone)

* No approx algo known for non monotone submodular f

$$S \xrightarrow{w_i} S \cup v_1 \xrightarrow{k \text{ steps}} S \cup v_1 \cup v_2 \dots \xrightarrow{} S \cup v_1 \cup \dots \cup v_k \geq w_{\text{opt}}$$

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

$$S \cup v_1 \cup \dots \cup v_j \xrightarrow{v_{j+1}} S \cup v_1 \cup \dots \cup v_{j+1} \geq w \quad \text{by submodularity}$$

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

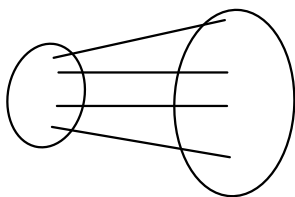
Cut-function [Max-cut]

f : defined on subset of vertices

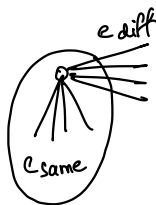
$f(S) = \# \text{ edges with one end point in } S, \text{ other not in } S$

max \curvearrowright (easy) min \curvearrowright max in n dim with many vertices.

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



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



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

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

Cardinality: size of set is at most k

\hookrightarrow each element has a cost, find subset with cost $\leq C$ having maximum f value

[Knapsack constraint] where f is monotone, submodular.

\uparrow
also has a constant factor approx.

Just greedy doesn't work,
need to compare two different solutions.

(Analogous to knapsack)

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. $|S| \leq k$, $\max_{v \in V} \min_{u \in S} d(u, v)$ 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 x 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.

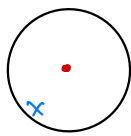
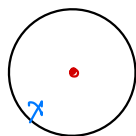
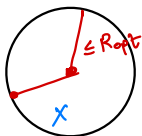
So, better than 2-approximation is NP-hard.

Algorithm (2-approx)

Initially pick an arbitrary vertex

At each step pick a vertex that is furthest away from currently selected

til k vertices are selected



Optimal Radius is R_{opt}

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

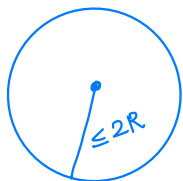
distance b/w any pair in the same group is atmost $2R_{opt}$.

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 2nd 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 $2R$



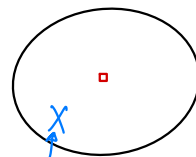
Pick arbitrary point,
remove all points
with distance $\leq 2R$

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 $\leq R$

opt $\left[\begin{array}{c} l \\ 0, \dots, R \end{array} \right]$ \downarrow $\left[\begin{array}{c} u \\ \text{max distance} \end{array} \right]$



deletes the entire cluster!

{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.

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

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

$$d(e_1) \leq d(e_2) \leq \dots \leq d(e_{\binom{n}{2}})$$

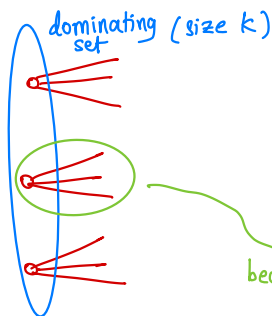
← bottleneck problems (e.g. min of max weight edge in ST)

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.

$$H_i^2 = G_i^2$$

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



$$G_i^2$$



Independent set in G_i^2 has size at most 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 $\leq k$

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

↓
optimal radius is at least $\text{cost}(e_i)$

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

$\text{cost}(e_i)$ is a lower bound on optimal solution.

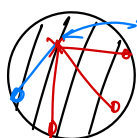
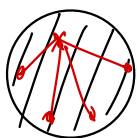
We use the maximal ind set in G_i^2 to get a solution of cost at most $2 \text{cost}(e_i)$

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

MIS is a dominating set $\Rightarrow \exists$ a path of length 2 in G_i , by Δ ineq $\leq 2 \text{cost}(e_i)$ of that edge.

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) \rightarrow 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 ≤ 3 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.

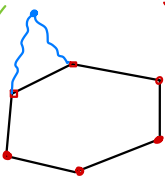
$$OPT \leq 2 \times MWST.$$

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

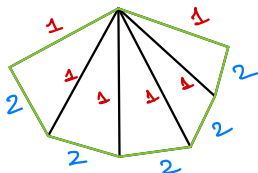
length of cycle \leq length of pre-order traversal $= 2 \times MWST$

#ask: proof?

we Δ ineq
 \rightarrow



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)

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

$$\Rightarrow \boxed{ACI \leq \frac{3}{2} \times \text{opt}}$$

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

$\text{opt} = 1$ iff \exists a hamiltonian cycle in G .
otherwise $\text{opt} = 2$.

2-approximation

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

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

$\Rightarrow \text{opt} \geq d(e_i)$ \leftarrow may or may not have a hamiltonian cycle. but previous G_i s definitely don't.

Fleischer'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

$$\leq 2d(e_i) \quad \{ \Delta \text{ inequality} \} \quad [\text{wt of edge } (i,j) = \text{length of shortest 2-hop}]$$

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



$$\Rightarrow \boxed{d(e_i) \leq \text{opt} \leq 2d(e_i)}$$

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

3-approximation

Stop as soon as G_i is connected and for any tree T , τ^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

$$\text{Lateness} = \max(0, \text{completion time} - \text{deadline})$$

- 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

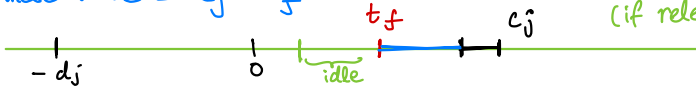
Algorithm: Whenever the machine is free and a task is available, start executing any available task

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

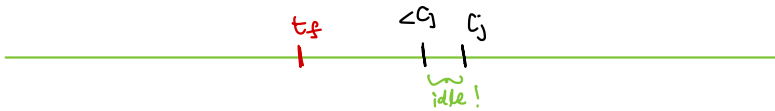
Let t_f be the last time before c_j when machine was idle for sometime just before t_f

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 ≤ 0 , lateness in any schedule $\geq c_j$

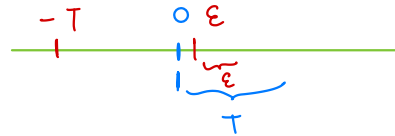
Also, in any schedule, the max lateness $\geq d_j$ (since release $\geq 0 \Rightarrow \text{lateness}_j \geq d_j$)

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

Tight example

r_i	0	ϵ
d_i	0	$-T$
T_i	T	ϵ

$\Rightarrow \text{max lateness} = 2T + \epsilon$



Another version of Scheduling

$r_i \rightarrow$ Pre-processing time
 t_i
 $d_i \rightarrow$ Post processing time
 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 $\text{opt} \geq d_j$

#ask: why?

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

- m identical machines
- n tasks with i th task having execution time t_i
- ↳ Assign tasks to the machines to minimize the makespan

$$\text{makespan} = \max \text{ completion time of a machine}$$

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

- $T_{\text{opt}} \geq T_{\text{max}} = \max(t_i)$
 - $T_{\text{opt}} \geq \frac{\sum t_i}{m}$
- } 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
- ⇒ All machines were executing until at least $T - t_j$.

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

$$(\because T_{\text{opt}} \geq t_j) \geq T - \frac{m-1}{m} T_{\text{opt}}$$

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

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

[LPT: longest processing time first]

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

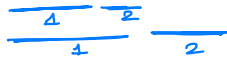
$$t_1 \geq t_2 \geq \dots \geq t_n$$

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 at most 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.



\Rightarrow can swap to reduce makespan

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

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

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

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

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

3	2	2
3	2	

$T_{alg} = 7$

3	3
2	2
	2

$opt = 6$

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 $\leq T_i$

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

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

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

$\}$ 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 at most n tasks \Rightarrow increase in value by at most $n \times \frac{\epsilon}{n} T_{\max}$ after scaling back

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

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

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

for $(1+\epsilon)$ -approximation.

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

Guess an optimum T .

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

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

Idea: Do binary search on parameter T .

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

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

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

Each machine can execute at most k large jobs if it completes at time $\leq T$

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

Now, maximum possible value of any large job is at most k^2 , at least 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 , $0 \leq i \leq k^2$

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

E.g. 0 1 5 6 ...

\leftarrow valid configuration if total execution time in the modified instance is $\leq k^2 \leftarrow \lfloor \frac{T}{k^2} \rfloor$

i.e. $(n_0, n_1, \dots, n_{k^2})$ s.t. $\sum i n_i \leq k^2$

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)

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

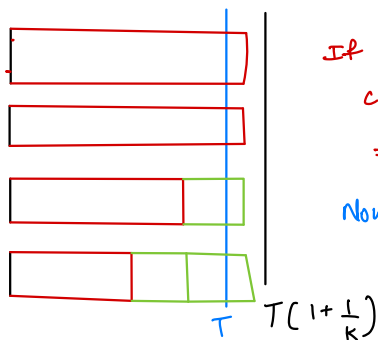
There are $\sim n^{k^2}$ possible vectors for inputs, but possible choices for machine bounded by a constant

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

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

Small objects ($t_i \leq \frac{T}{k}$)

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



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 \leq
 size of last object crossing blue line
 $\leq \frac{T}{k}$

$$A.C.T. \leq T(1 + \frac{1}{k})$$

Now, binary search over T , we get

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

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

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

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

→ solve the LP, nuke fractional values.

$$\begin{aligned} \max \sum x_i & \quad \exists \text{ opt with at most two fractional } x_i \\ \sum s_i x_i & \leq S \\ \sum w_i x_i & \leq W \\ 0 \leq x_i & \leq 1 \end{aligned}$$

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

↑
 Gives an additive approximation.

$$\begin{aligned} s_1 x_1 + s_2 x_2 + s_3 x_3 &= S \\ w_1 x_1 + w_2 x_2 + w_3 x_3 &= W \end{aligned}$$

$\uparrow \quad \quad \uparrow \quad \quad \uparrow$
 $\delta_1 \quad \delta_2 \quad \delta_3$

$$\begin{aligned} s_1 \delta_1 + s_2 \delta_2 + s_3 \delta_3 &= 0 \\ w_1 \delta_1 + w_2 \delta_2 + w_3 \delta_3 &= 0 \\ \Rightarrow \exists \delta_1, \dots, \delta_3 \neq 0. \end{aligned}$$

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

(not strongly polynomial time)

Change in cost = $s_1 + s_2 + s_3$. (if +ve ✓)
 if -ve, flip signs to get inc in cost)

$x_1 + x_2 < 2$
 \Rightarrow integral solution discards both
 \Rightarrow additive approximation of 1 since $A(\mathbb{I}) > \text{opt} - 2 \geq \text{opt} - 1$.

Lecture

5/2

• n tasks, i th task has release time r_i and execution time t_i

Single machine → one task at a time

→ Task must be completed once started

[no pre-emption]

• Minimizing max finish time is easy → keep running any available task

Finish time $\geq \max$ 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 :
 $\sum_{\text{all tasks } i} C_i \rightarrow C_i = \text{completion time of } i$

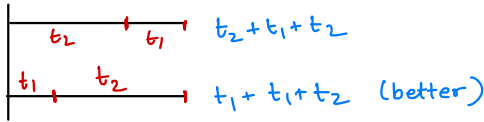
• If no pre-emption is allowed, then the problem is NP-hard.

• 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
 \Rightarrow (min remaining time) is optimal

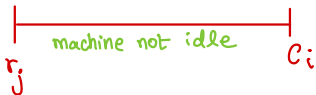
• Consider a non pre-emptive soln in which tasks are executed in the order in which they complete in the pre-emptive schedule. The task is executed as early as possible in this order.

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,

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

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



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

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

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

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

$$\leq r_j + \sum_{j=1}^i t_j \leq C_i^* + C_i^* = 2C_i^*$$

↑ last release time after which not idle
 ↑ max running time

$$r_i \leq r_j$$

If $r_i > r_j$ then



Weighted Completion time

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

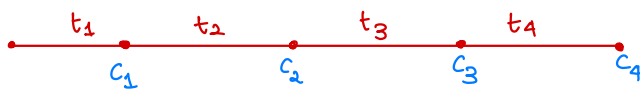
- Finding optimal schedule with pre-emption allowed is also NP-hard
- In this case, we formulate a linear program in which the completion times of the tasks are variables.

C_i are now variables.

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

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,



$$\begin{aligned} C_1 &= t_1 \\ C_2 &= t_1 + t_2 \\ C_3 &= t_1 + t_2 + t_3 \\ C_4 &= t_1 + t_2 + t_3 + t_4 \end{aligned}$$

$$\Rightarrow \sum_{i \in S} t_i C_i \geq \frac{1}{2} (\sum_{i \in S} t_i)^2 + \frac{1}{2} \sum_{i \in S} t_i^2$$

$\underbrace{\quad}_{\text{includes } t_j \forall j \leq i}$
 \Rightarrow includes $t_i t_j$ for every pair $(i, j) \in S$, and $t_i^2 \forall i$

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

So, we get an LP,

$$\min \sum w_i C_i$$

$$C_i \geq r_i$$

$$\sum_{i \in S} t_i C_i \geq \frac{1}{2} t(S)^2 \quad \forall \text{ subset } S \text{ of tasks}$$

\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 : $C_i \leq 3C_i^* \forall i$

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

$r_j \leq C_j$ no idle time C_i
Tasks here must have $j \leq i$ [$C_j^* \leq C_i^*$]

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

$$\Rightarrow C_i^* \sum_{j \in S} t_j \geq \sum_{j \in S} t_j C_j^* \geq \frac{1}{2} t(S)^2 \quad [\text{feasible pt in LP}]$$

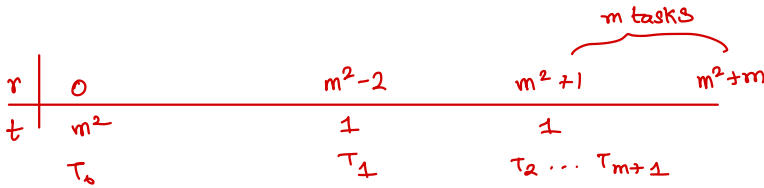
$\underbrace{\sum_{j \in S} t_j}_{t(S)} \quad \underbrace{S = \text{first } i \text{ tasks}}$

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

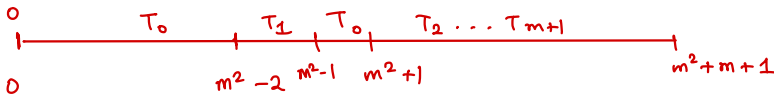
$$\Rightarrow C_i \leq r_j + t(S) \leq C_i^* + 2C_i^* \Rightarrow \boxed{C_i \leq 3C_i^*} \quad , \text{ hence gives a } 3\text{-approx algorithm}$$

Worst Case of Approximation

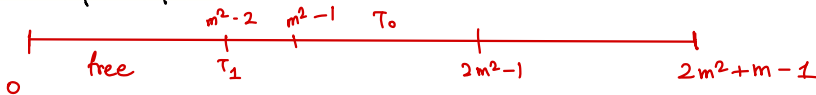
6/2/24



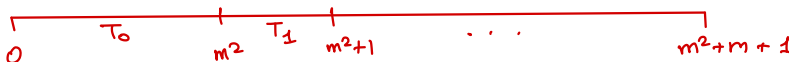
pre-emptive schedule



without pre-emption



Opt



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

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

$$\text{As } m \rightarrow \infty, \frac{A(I)}{\text{opt}(I)} \rightarrow 2$$

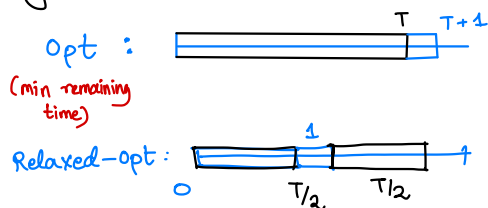
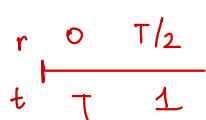
Questions :

- Does \exists a 3-approx tight example for weighted case
- Is weighted LP exact?
Are all solⁿ in it feasible?
- Best bound on quality of the approx?

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(E) is far.

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

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



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

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

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

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

LP relaxation for weighted scheduling

$$\begin{aligned} \min \sum_i w_i c_i \\ c_i \geq r_i \quad \forall i \end{aligned}$$

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

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 \Rightarrow Efficient separation oracle \Rightarrow efficient soln. of LP.

Given values of c_i , do they satisfy all inequalities?

• $c_i \geq r_i$ checked easily

Order the sets s.t. $C_1 \leq C_2 \leq C_3 \leq \dots \leq C_n$

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

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.

\uparrow (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

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

If decrease in $\text{LHS} \geq \text{RHS}$, 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,

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

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

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

Note: 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 \rightarrow each edge has a +ve weight $[c_e]$

\rightarrow 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

$\hookrightarrow \infty$ for terminals, 0 for others in this problem gives a Steiner tree.

Find a tree T s.t.

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

Opt will be a tree
if not prune off cycles!

$y_i \rightarrow$ for each vertex i

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

$x_e \rightarrow$ for each edge e

$\forall S$ s.t. separates 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

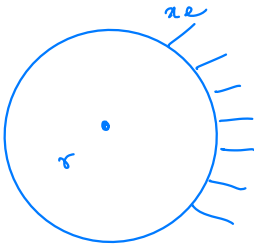
LP-relaxation

$$\sum_{\text{edges}} c_e x_e + \sum_{\text{vertices } i} \pi_i (1 - y_i) \quad \leftarrow \text{not exact formulation.}$$

$$\sum_{e \in \delta(S)} x_e \geq y_i \quad \left. \begin{array}{l} \text{where } S \text{ separates} \\ \text{root } r \text{ from } v_i \end{array} \right\} \text{exp. many constraints}$$

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

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



v_i

Take x'_e values as capacity of edges and check that max flow from r to v_i is at least y'_i .

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

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

\rightarrow 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

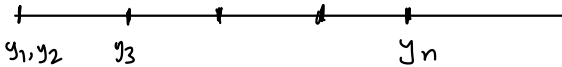
Penalty = $\sum_{i: y_i < \frac{2}{3}} \pi_i$
by algorithmic soln

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

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

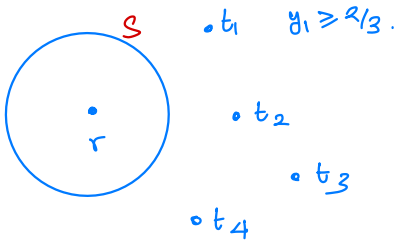
$$\geq \text{Algo penalty} / 3$$

Note: Rounding has limited choices for y_i . Arrange in \uparrow order and threshold choices are only values of y_i



Chosen the set of vertices of vertices to include (has a 2-approx)

\rightarrow finds a steiner tree with r and selected vertices on terminal



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

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

But for steiner tree, we need

$$\sum_{e \in S(S)} x_e \geq 1.$$

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

\Rightarrow we can find a ^{weighted} steiner tree whose cost is $\leq 2 \times$ cost of LP-relaxation

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

⇒ Cost of solution is at most 3 times cost of the LP solution.

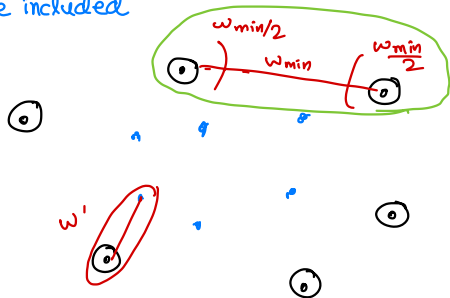
(MST is 2-approx)

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, at least one edge in $\delta(S)$ must be included



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

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

Algorithm

- Increase dual variable by δ for all singleton cuts

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

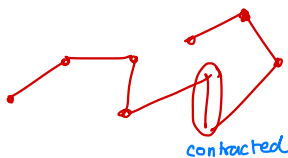
w_1 is min wt edge joining terminals,

w_2 is min wt edge joining terminal to non-term.

→ Dual cost increases by $\delta |T|$

↳ Contract any edge that becomes tight

reduce weights of all edges incident with T by δ



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

$$\sum y_S \leq c_e$$
$$\delta(S) \geq e$$

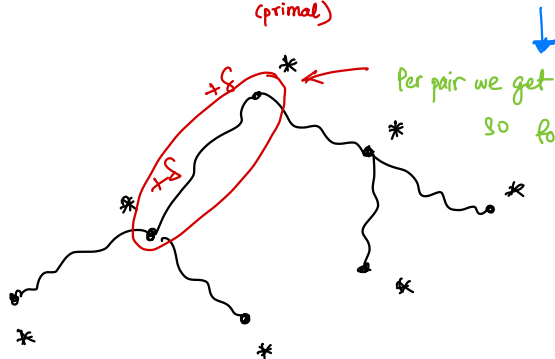
→ Assume by induction you can find a tree in the remaining graph with cost at most $2 \times$ dual solution

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

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 at most $2|T| - 2$ (easy to see)

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



Per pair we get $+\delta$,

so for $|T|-1$ pairs (upon contracting edges with one non-term. end in arbitrary order)

we get at most $2 \delta |T|$ 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
$$\min_{S \subseteq F} \left[\sum_{i \in S} f_i + \sum_{j \in D} \min_{i \in S} c_{ij} \right]$$

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

Greedy: Select a facility i which minimizes

$$\frac{f_i + \sum_{j \in S} c_{ij}}{|S|}$$

min over all non-empty subsets S of D

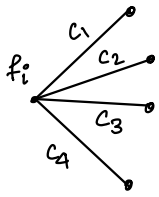
→ open facility i and connect all elements of S to facility i
→ Then remove S from clients and continue

This S can be found by sorting c_{ij}

Analysis - For any facility i and subset S of clients,

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

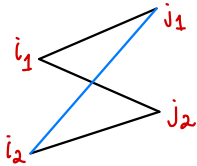
[same argument as set cover]



first elem covered \rightarrow cost assigned \leq 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

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

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

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

$$\sum_j x_{ij} = 1 \quad \forall j \quad \leftarrow \text{each client connected to 1 facility}$$

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

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

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

Dual :

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

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

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

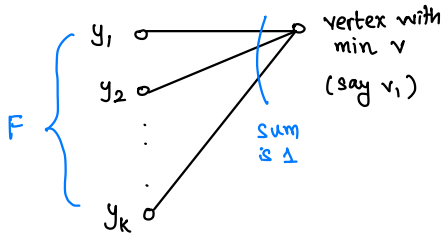
$$w_{ij} \geq 0$$

Dual LP-opt = LP opt.

If a variable that is ≥ 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 $x_{ij} > 0$, 13/2/24

then we have $v_j - w_{ij} = c_{ij}$

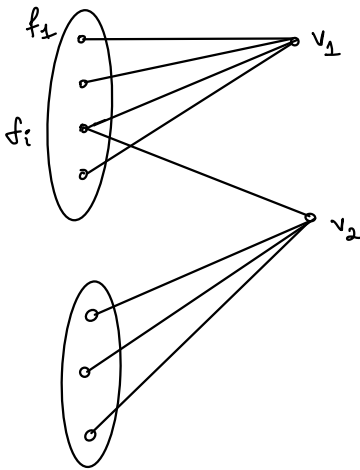


For facilities with $x_{ij} > 0$,
Pick facilities with min f_i (say f_1)

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.

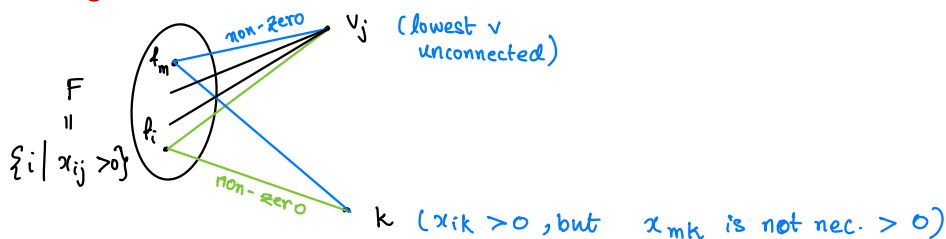
Pf: The dual cost of opening facilities in F

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

$y_i > 0$ 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 $\leq \sum_i f_i y_i \leq \text{LP-opt}$

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



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]

$$\leq v_j + v_j + v_k$$

$$\leq 3 v_k \quad (v_j \text{ was lowest cost})$$

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

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

$$\Rightarrow A(\mathcal{I}) = \text{connection cost} + \text{Opening Cost}$$

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

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

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 , $\sum_j w_{ij} \leq f_i$, $v_j - w_{ij} \leq c_{ij}$ $\{\max \sum v_j\}$

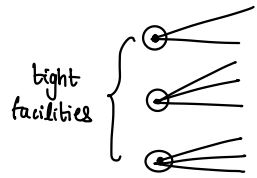
Start with all $v_i = 0$, $w_{ij} = 0$

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



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

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

1. If some client becomes a neighbor of a tight facility ($v_j = c_{ij}$)
 2. Some new facility becomes tight
- Stop increasing the dual only for this client
Stop increasing the dual for all neighbors of this facility

Claim: \exists an integer solution whose cost is at most 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)

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

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

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

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

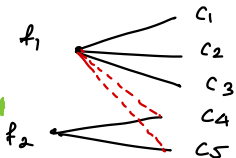
\leftarrow If $N(f_i)$ are disjoint, we get exactly

$$\sum_{j \in N(f_i)} v_j = \sum_{j \in S} v_j = \text{dual cost} \Rightarrow \text{primal cost corresponds to optimal primal.}$$

Construction of solution: (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

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

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

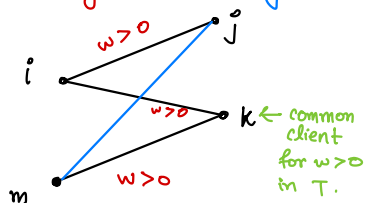
- Consider a graph on T in which two are adjacent if some client has non-zero value of w_{ij} in both facilities

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 $T' \Rightarrow$ connect it to any one of them

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

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



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

$$C_{ij} \leq v_j \quad [\text{they are neighbors}]$$

$$C_{ik} \leq v_k$$

$$C_{mk} \leq v_k$$

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

$w_{ik} > 0 \Rightarrow k$ can't become neighbor of i after it becomes tight.

v_j can't increase because i became tight. Hence, $v_k \leq v_j$
[by construction]

So k cannot increase after i became tight $\Rightarrow k$ was removed either before or when i became tight

$\Rightarrow v_k$ didn't increase after j stopped increasing

$$\text{Hence, } C_{mj} \leq 3v_j$$

$$\text{Total cost} = \sum_{i \in T'} (f_i + \sum_{j \in N(i)} C_{ij}) \leq \sum_{i \in T'} (\sum_{\text{direct conn.}} v_j + 3 \sum_{\text{indirect conn.}} v_j)$$

$$\leq 3 \sum v_j \leq 3 \text{ opt } (I)$$

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 \rightarrow assigning tasks to machines

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

ILP Formulation

$$\begin{aligned} \min \quad & t \\ \sum_j x_{ij} t_{ij} & \leq t \quad \forall \text{ machine } j \\ \sum_j x_{ij} & = 1 \quad \forall \text{ task } i, \text{ assign to exactly 1 machine} \\ x_{ij} & \in \{0, 1\} \end{aligned}$$

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

Ex: One task, m machines, $t_{1j} = m \quad \forall j$

Optimal integral time = m

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

\downarrow

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 $\leq T$ or finds one with completion time at most $2T$.

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 $\geq nT \Rightarrow \text{opt} \geq nT/m$

\uparrow
a possible assignment

\downarrow
will give m -approximation

Modified LP Formulation [modelling as LP feasibility problem]

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

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

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

Claim : If this LP has a feasible solution, then we can find an assignment with completion time at most $2T$, 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 at most r of these inequalities to be tight \Rightarrow in any such set, at least $r - (n+m)$ inequalities must be of the form $x_{ij} = 0$

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

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

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

\Rightarrow If there are k fractionally assigned tasks and $(n-k)$ integrally assigned tasks, # non-zero x_{ij} is at least $(n+k)$

$$\Rightarrow \frac{\# \text{non-zero } x_{ij}}{n+k} \leq \frac{\max \# \text{non-zero } x_{ij}}{n+m} \Rightarrow \boxed{k \leq m}$$

$\leftarrow \frac{2k}{2x_{ij} > 0 \text{ per fractional task}} + \frac{n-k}{\text{integral}}$

\Rightarrow The fractionally assigned tasks can be integrally to machines such that every machine is assigned at most one task.

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

\downarrow
Next page

\Rightarrow Each machine's execution time increases by at most T

\Rightarrow Actual completion time $\leq 2T$

$\forall i, j, t_{ij} \leq T$.

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

Construct a graph with tasks, machines as edges where $(t_i, m_j) \in E$

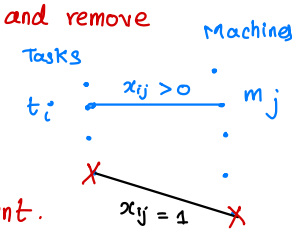
iff $x_{ij} > 0$

atmost $n+m$ x_{ij} are non-zero.

(Rounding Algorithm)

\Rightarrow Has $(n+m)$ vertices, atmost $(n+m)$ 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. pick alternate edges.

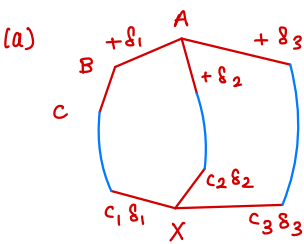
has min # edges given connected, $|E| = |V|$.

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

20/2

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)



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 $-\delta_1 \frac{t_{AB}}{t_{CB}}$

We can similarly propagate the changes along the edges.

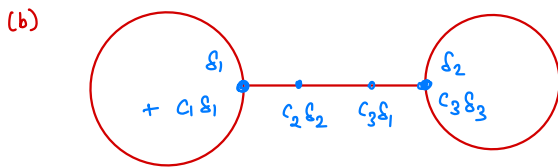
If X is a task $\Rightarrow c_1 \delta_1 + c_2 \delta_2 + c_3 \delta_3 = 0$

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 has a smaller no. of fractional edges.



Here also, we can propagate and get a solution.

#ask?

Vertex Cover

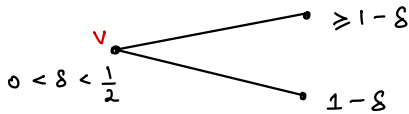
$$\min \sum_i w_i x_i$$

$$x_i + x_j \geq 1 \quad \forall (ij) \in E$$

$$x_i \geq 0$$

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

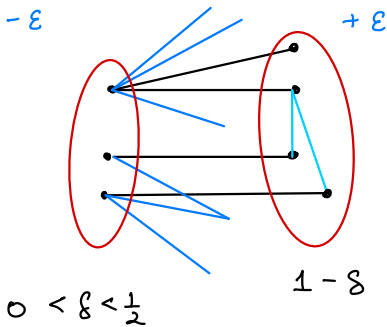
If any x_i has weight $\delta \in (0, \frac{1}{2})$, then every neighborhood will have weight $(1-\delta)$



increasing # tight constraints.

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

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



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

Also, for small enough ϵ , we can increase the left by ϵ and decrease the right by ϵ

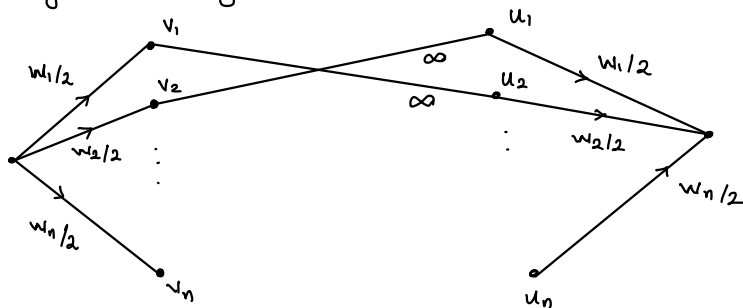
\Rightarrow The soln is not basic

(since more constraints are becoming tight)

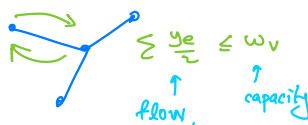
[no internal edges since weight $< \frac{1}{2}$]

Solving the LP using max-flow

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



If (i,j) are adjacent connect (v_i, u_j) and (v_j, u_i)



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 $\Rightarrow \max \sum y_e$

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

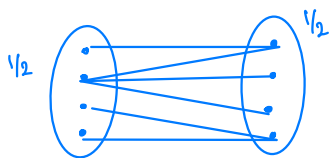
$$y_e \geq 0$$

dual of vertex cover LP

\Rightarrow They have the same optimal value

Rounding: [For bipartite graph]

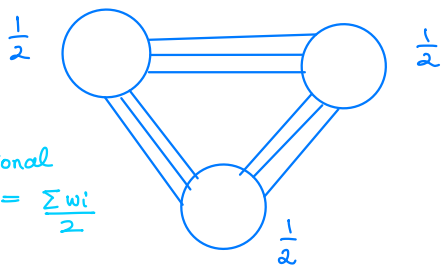
In the fractional solution first pick all vertices with integer cost. For the half integral weights,



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

\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 $\leq \frac{2}{3} \sum w_i$ \leftarrow Picking 2 least weight sides

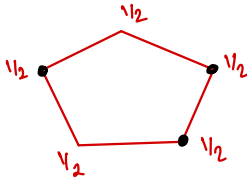
\Rightarrow Atmost $\frac{4}{3}$ times the fractional cost

[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

2.0 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.

Case 1: $\forall e \ x_e > 0$ in the optimal primal soln.

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

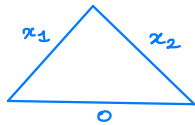
$$\Rightarrow \sum_e w_e = \sum_e \sum_{t \ni e} y_t = 3 \sum y_t$$

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

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

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

Case 2: $\exists e : x_e = 0$



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

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

\Rightarrow In remaining graph, set of x_e still feasible

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

Induction hypothesis

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

Now, adding edge back

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

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: $\sum_{i=1}^n w_i X_i$

$$\begin{aligned} \Rightarrow \mathbb{E}[\text{cost}] &= \sum_{i=1}^n w_i \mathbb{E}[X_i] = \sum_{i=1}^n w_i \Pr(C_i \text{ is satisfied}) \\ &= \sum_{i=1}^n w_i \left(1 - \left(\frac{1}{2}\right)^{|C_i|}\right) \end{aligned}$$

$$\text{and, } 1 - \left(\frac{1}{2}\right)^{|C_i|} \geq \frac{1}{2}$$

$$\Rightarrow \mathbb{E}[\text{cost}] \geq \sum_i \frac{w_i}{2}$$

$$\Rightarrow \sum_i w_i \geq \text{max-cost} \geq \mathbb{E}[\text{cost}] \geq \frac{\sum_i w_i}{2} \quad , \text{i.e., gives a } \frac{1}{2}\text{-approx [in expectation]}$$

Derandomization (method of conditional expectations)

$$\begin{aligned} \mathbb{E}[\text{cost}] &= \frac{1}{2} (\mathbb{E}[\text{cost} | X_1 = 1] + \mathbb{E}[\text{cost} | X_1 = 0]) \\ &\leq \max(\mathbb{E}[\text{cost} | X_1 = 1], \mathbb{E}[\text{cost} | X_1 = 0]) \end{aligned} \quad \left. \vphantom{\begin{aligned} \mathbb{E}[\text{cost}] &= \frac{1}{2} (\mathbb{E}[\text{cost} | X_1 = 1] + \mathbb{E}[\text{cost} | X_1 = 0]) \right\}} \text{here } X_1 = 1 \text{ denotes value of } X_1 \text{ set to 1.}$$

Select $X_i = 1$ or 0 s.t. $\mathbb{E}[\text{cost} | X_i]$ 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.

$$\{w(\bar{x}_i) \leq w(x_i) \forall i\} \\ \text{if } x_i \in C \text{ or } \bar{x}_i \in C$$

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

$$\begin{aligned} \Pr(C \text{ is satisfied}) &= 1 - (1-p)^a p^b \quad (p > \frac{1}{2}) \\ &\geq 1 - p^{a+b} \\ &\geq 1 - p^2 \end{aligned}$$

Choose p s.t. $p = 1 - p^2$, $p = \frac{\sqrt{5}-1}{2}$

if $x_i \in C$, $\bar{x}_i \notin C$, $\Pr(x_i \text{ sat}) = p = 1 - p^2$

clause has x_i, \bar{x}_j , $\geq 1 - p^2$

clauses $x_i, \bar{x}_i \in C$, cost = $p w_1 + (1-p) w_2 \geq$

$$\Rightarrow \mathbb{E}[\text{cost}] = \left(\frac{\sqrt{5}-1}{2}\right) \sum w_i \Rightarrow \text{Better than } \frac{1}{2} \text{-approximation.}$$

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

$$\text{Expected cost} = \frac{\sum w_i}{2}$$

Derandomization

Place x_1 in set which gives higher $\mathbb{E}[C | x_1]$.

1. Interpret relaxed LP soln. as the probability that the variable takes value 1.
2. Compare expected cost with the optimal LP-cost

The LP formulation of MAX-SAT is

$$y_i \rightarrow 1 \text{ if } x_i \text{ is true}$$

$$z_j \rightarrow 1 \text{ if clause } j \text{ is satisfied}$$

$$\max \sum w_j z_j$$

$$\sum_{i \in P_j} y_i + \sum_{i \in N_j} (1 - y_i) \geq z_j \quad \forall \text{ clause } j$$

$$z_j, y_i \in \{0, 1\} \xrightarrow{\text{relax.}} 0 \leq z_j, y_i \leq 1$$

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 = $\sum_j w_j \Pr(\text{clause } j \text{ satisfied})$

$$\begin{aligned} \Pr(\text{clause } j \text{ not satisfied}) &= \prod_{i \in P_j} (1 - y_i^*) \prod_{i \in N_j} y_i^* \\ &\leq \left(\frac{\sum_{i \in P_j} (1 - y_i^*) + \sum_{i \in N_j} y_i^*}{l_j} \right)^{l_j} \quad \leftarrow \text{no. of literals} \\ &\leq \left(1 - \frac{z_j^*}{l_j} \right)^{l_j} \end{aligned}$$

$$= \sum_j w_j \left(1 - \left(1 - \frac{z_j^*}{l_j} \right)^{l_j} \right)$$

Need to show that $\Pr(\text{clause is satisfied}) > c z_j^* \geq c \times \text{opt-LP cost}$

Since, $1 - \left(1 - \frac{z_j^*}{l_j} \right)^{l_j}$ is concave for $0 \leq z_j^* \leq 1$, using Jensen

$$\begin{aligned} 1 - \left(1 - \frac{z_j^*}{l_j} \right)^{l_j} &\geq (1 - z_j^*) (0) + z_j^* \left(1 - \left(1 - \frac{1}{l_j} \right)^{l_j} \right) \\ &\geq \left(1 - \frac{1}{e} \right) z_j^* \end{aligned}$$

$$\geq \left(1 - \frac{1}{e} \right) \sum_j w_j z_j^* \geq \left(1 - \frac{1}{e} \right) \text{LP-opt} \geq \left(1 - \frac{1}{e} \right) \text{opt.}$$

If we take max of solutions obtained by setting each variable true with prob. $\frac{1}{2}$ by rounding the LP.

$$\max(E_1, E_2) \geq \frac{E_1 + E_2}{2}$$

$$\text{Expected value of } E_1 = \sum w_j (1 - (\frac{1}{2})^{l_j}) \geq \sum w_j z_j^* (1 - (\frac{1}{2})^{l_j})$$

$$\text{Expected value of } E_2 \geq \sum w_j (1 - (1 - \frac{1}{2^{l_j}})^{l_j}) z_j^*$$

$$\Rightarrow \frac{E_1 + E_2}{2} \geq \sum \frac{1}{2} w_j z_j^* (2 - (1 - \frac{1}{2^{l_j}})^{l_j} - (\frac{1}{2})^{l_j})$$

$$\geq \frac{3}{4} \text{ LP-opt} \quad (l_j \in \mathbb{N})$$

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.

$$1 - \frac{1}{4^y} \leq f(y) \leq 4^{y-1} \quad [\forall y \in [0,1], 1 - 4^{-y} \leq 4^{y-1}]$$

$$\begin{aligned} \Pr(\text{clause } j \text{ not satisfied}) &= \prod_{i \in P_j} (1 - f(y_i^*)) \prod_{i \in N_j} f(y_i^*) \leq \prod_{i \in P_j} 4^{-y_i^*} \prod_{i \in N_j} 4^{y_i^*-1} \\ &= 4^{-(\sum_{i \in P_j} y_i^* + \sum_{i \in N_j} (1 - y_i^*))} \leq 4^{-z_j^*} \end{aligned}$$

$$\Pr(\text{clause } j \text{ satisfied}) \geq 1 - 4^{-z_j^*} \geq \frac{3}{4} z_j^* \quad (\text{jensen})$$

We can show that $\Pr(\text{clause } j \text{ is satisfied}) \geq \frac{3}{4} z_j^*$

$$\Rightarrow \text{Expected cost} \geq \frac{3}{4} \times \text{Opt-LP}$$

Note: Integrality gap of this is $\frac{3}{4}$, so this is the best possible approximation using this LP.

Example for $\frac{3}{4}$: $x_1 \vee x_2, \bar{x}_1 \vee x_2, x_1 \vee \bar{x}_2, \bar{x}_1 \vee \bar{x}_2$

wt = 1 per clause

$$\text{LP opt} = 3$$

$$\text{LP opt} = 4 \quad (y_1^* = y_2^* = \frac{1}{2})$$

Given an undirected graph with weights on edges and a specified vertex r and a penalty π_i for every other.

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

ILP Formulation

x_e for edge e

y_i for vertex i , $i \neq r$

$$\min \sum_e c_e x_e + \sum_i \pi_i (1 - y_i)$$

for any cut that separates i from r ,

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

$$x_e, y_i \in \{0, 1\}$$

Previous deterministic Algorithm

1. Solve the LP optimally

2. If $y_i \geq \frac{2}{3}$ round it up to 1

3. Find a 2-approximation to the Steiner tree with r and all the y_i included

This gives a 3-approximation

If we choose α as the threshold, penalty cost is at most $\frac{1}{1-\alpha}$ prev. penalty cost and connection cost is at most $\frac{2}{\alpha}$ LP connection.

choose α randomly and uniformly in the interval $[r, 1]$

All vertices with $y_i < r$ will be excluded with $p_r = 1$.

If $y_i \in [r, 1]$, the vertex is excluded with prob. = $\frac{1-y_i}{1-r}$

Expected penalty cost = $\sum_i \pi_i$ (Pr(vertex i is excluded))

$$\leq \sum \pi_i \left(\frac{1-y_i}{1-r} \right) = \frac{1}{1-r} \times \left(\begin{array}{c} \text{Penalty in} \\ \text{LP solution} \end{array} \right)$$

(if $y_i < r$ then $\frac{1-y_i}{1-r} > 1$)

Connection cost = $\sum_{e \in \delta(S)} x_e \geq y_i \geq r$

Scaling x_e by $\frac{1}{r}$ gives valid LP soln for Steiner tree (2-approx)

So,

$$\leq \frac{2}{r} \times \text{LP-connection cost}$$

$$\Rightarrow \text{Expected connection cost} \leq \int_r^1 \frac{2}{\alpha} \times (\text{LP-conn}) \underbrace{\frac{d\alpha}{1-\alpha}}_{\text{prob. of lying in } [r, 1] \text{ interval.}}$$

$$= -\frac{2 \ln r}{1-r} \times (\text{connection cost of LP})$$

Now, $-\frac{2 \ln r}{1-r} = \frac{1}{1-r}$ for $r = \frac{1}{\sqrt{e}}$.

Putting $r = \frac{1}{\sqrt{e}}$, approximation ratio is $\frac{1}{1 - \frac{1}{\sqrt{e}}} \approx 2.54$

To de-randomise this, we can select one vertex at a time and apply conditional expectation.

Minimising Weighted Sum of Completion Times

- 1 machine
- n tasks, release time r_i , execution time t_i , weight w_i
- non pre-emptive schedule that minimizes $\sum_i w_i C_i$ where C_i is completion time of task i

We use a different LP formulation here and then use randomized rounding

Max completion time is atmost $\max_i r_i + \sum t_i$ (exponential in input size)

We assign variables for every unit time slot . $x_{it} = 1$ if task i is executed in time slot t , $0 \leq t \leq \text{max time slot}$

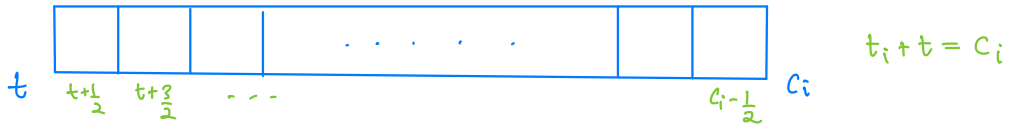
$$\sum_{i=1}^n x_{it} \leq 1 \quad \text{for all slots } t$$

$$\sum_{t=1}^T x_{it} = t_i \quad \text{for all tasks } i$$

$$x_{it} = 0 \quad \forall t \leq r_i$$

$$x_{it} \geq 0 \quad \forall t$$

With these constraints pre-emption is allowed . If a task is executed consecutively in some time slots , define the mean busy time of a task as the average of mid points of these slots



$$\text{Mean busy time} = t + \frac{t_i}{2} = C_i - \frac{t_i}{2}$$

$$C_i = \text{mean busy time} + \frac{t_i}{2}$$

If task was executed in non-consecutive time slots , $C_i \geq \text{mean busy time} + \frac{t_i}{2}$

$$\Rightarrow C_i = \frac{t_i}{2} + \underbrace{\frac{1}{t_i} \sum_{t=1}^T (t - \frac{1}{2}) x_{it}}_{\text{mean busy time}}$$

and we need $\min \sum w_i C_i$

$$\sum_{t=1}^T x_{it} = 1 \quad \text{for each task } i$$

$$\sum_{i=1}^n x_{it} \leq 1 \quad \text{for slot } t$$

$$x_{it} \geq 0 \quad \forall t$$

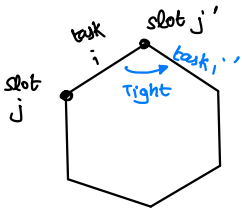
$$x_{it} = 0 \quad \forall t < r_i$$

$$\min \sum w_i C_i$$

$C_i \leftarrow$ estimate of completion time of task i

$$C_i = \underbrace{\frac{b_i}{2}}_{\text{mean busy time}} + \underbrace{\frac{1}{b_i} \sum_{t=1}^T (t - \frac{1}{2}) x_{it}}_{\text{spread dependent}}$$

This LP is integral. Every vertex is integral



Now, even cycle of fractional variables can be augmented to get integral points

[increases # l.i. tight constraints / lin comb. of vertex]

Note: If slot j' , j is not tight, $x_{ij'}$ can be increased until slot j' is tight (x_{ij} reduced by same amt)

Since every vertex is integral \exists an optimal with $x_{it} = 0$ or 1 for all i

Optimal schedule can be found by a greedy algorithm. At each step choose an available task with $\min_{(t \geq r_i)} \frac{w_i}{t_i}$ [contribution to opt is $\frac{1}{t_i} w_i (t - \frac{1}{2})$]
[opt for mean busy time]

This does not guarantee minimum weighted completion time with pre-emption.

Convert this solution to the LP to a pre-emptive soln using randomised rounding

Some completion times for all tasks

\rightarrow One possible way of rounding is to execute non-preemptively in same order

Note: Mean busy time also gets arbitrarily bad

[Construct an example where this is arbitrarily bad]

Optimal pre-emptive solution

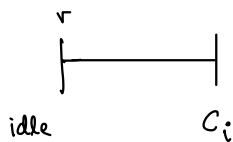


this chosen as time for ordering (bad)

Choose a random variable x_i for task with probability $\frac{x_{it}}{t_i}$ for value $t - \frac{1}{2}$
do this for each task and order based on values x_i

Show that expected integral cost is at most twice LP cost

for each task i , expected completion time of task is at most $2C_i^*$



(non-pre-emptive) schedule

Assume tasks are numbered $X_1 \leq X_2 \leq \dots \leq X_n$

optimal mean busy time

Maximum release time of tasks $1, 2, \dots, i-1$ is at least r

$$\mathbb{E}[C_i] \leq 2C_i^*$$

$$\mathbb{E}[C_i | X_i = X]$$

$$X_i = X$$



$$C_i \leq r + \sum_{j=1}^{i-1} t_j + t_i$$

$$r \leq \max_{1 \leq j \leq i} r_j \leq X - \frac{1}{2}$$

release time of task i

$X_j \leq X_i \Rightarrow$ there exists a slot before $X - 1/2$ (or equal) when j executed.

$$\mathbb{E}[\sum_{j=1}^{i-1} t_j] = \sum_{\substack{\text{all tasks} \\ \neq i}} t_k \Pr(\text{task } k \text{ is executed before task } i)$$

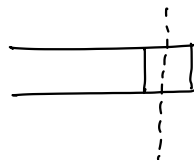
$$= \sum_{\substack{\text{all tasks} \\ \neq i}} t_k \Pr(X_k \leq X)$$

$$= \sum t_k \left(\sum_{t=1}^{\lfloor X + 1/2 \rfloor} x_{kt} \right) \cdot \frac{1}{t_k}$$

same slot won't be selected for more than 1 so $X - 1/2$ is fine.

Probability slot for $X_k \leq X$

$x_{kt} \rightarrow$ distribution for choosing X_k



$$\sum x_{kt} = t_k$$

$$= \sum_{k \neq i} \sum_{t=1}^{X+1/2} x_{kt} = \sum_{t=1}^{X+1/2} \sum_{k \neq i} x_{kt} \leq X + \frac{1}{2}$$

in a given slot only 1 task scheduled

$$C_i \leq \underbrace{r}_{\leq X - 1/2} + \underbrace{\sum_{j=1}^{i-1} t_j}_{\leq X + 1/2} + t_i$$

$$\mathbb{E}[C_i | X_i = x] \leq t_i + 2x$$

$$\begin{aligned} \mathbb{E}[C_i] &= \sum_{t=1}^T \Pr(X_i = t - \frac{1}{2}) \mathbb{E}[C_i | X_i = t - \frac{1}{2}] \\ &= \sum_{t=1}^T \frac{x_{it}}{t_i} (t_i + 2(t - \frac{1}{2})) \quad C_i^* = \frac{t_i}{2} + \frac{1}{t_i} \sum (t - \frac{1}{2}) x_{it} \\ &= t_i + 2 \frac{1}{t_i} \sum (t - \frac{1}{2}) x_{it} \\ &= 2C_i^* \quad , \text{ hence we get a 2-approx.} \end{aligned}$$

Alternate Algo: Choose $\alpha \in (0, 1)$ pick ordering using prs where α fraction of task is completed.

[ref: similar to steiner tree]

There is a discrete set of $\alpha \in (0, 1)$ where fraction of some task i completed changes. Enumeration over this gives a derandomised algorithm

Alternate Algo: If processor is idle

1. $t \geq r_i, t_i$. Among such tasks pick min $\frac{w_i}{t_i}$
2. Execute and repeat

Edge Disjoint Paths

18/3

- Rounding
- Integer multicommodity flow

Given a graph (undirected/directed) and a collection of pairs of vertices (s_i, t_i) $i=1, 2, \dots, k$

Choose a path from s_i to t_i for each i such that the congestion (max no. of paths through an edge) is minimized.

Simple LP Formulation

Consider all simple paths from s_i to t_i . Variable $x_p \in \{0, 1\}$ for each path p

$$\sum_{\substack{\text{all } (s_i, t_i) \\ \text{paths } p}} x_p = 1 \quad \forall i \quad [\text{one path for each pair}]$$

for any edge e , $\sum_{\substack{\text{all paths} \\ p \text{ st. } e \in p}} x_p \leq w$

Objective : min w

An equivalent formulation

Assume a directed graph. For every (s_i, t_i) pair, we have a commodity that flows from s_i to t_i .

For every edge there is a variable f_i indicating the amount of commodity i flowing through the edge.

$$\text{For } s_i, \sum_{\substack{\text{all outgoing} \\ \text{edges } e}} f_{ie} - \sum_{\substack{\text{all incoming} \\ \text{edges } e}} f_{ie} = 1$$

For t_i , Net outgoing flow = 1

For any other vertex, net outgoing flow = 0

for any edge e , $\sum_i f_{ie} \leq w$

Objective : min w

Given a solution to this LP, we can find a solution to the previous LP by decomposing the flows from s_i to t_i into edge disjoint paths.

Assume we have optimal for this [first LP]. Think of this soln. as a probability distribution on the paths from s_i to t_i .

Pick any of these paths P with probability x_p for each (s_i, t_i) pair.

We now find expected value of congestion.

For a given edge e , $x_e = 1$ if s_i, t_i contains e and 0 otherwise (in the integer soln.)

Expected no. of paths that contain $e = \sum_{P \text{ contain } e} x_p \leq w$

For a given edge, expected no. of paths that contain $e \leq w^*$

We want to bound the \max_e (no. of paths containing e).

So, we bound the prob. that for a given edge, no. of paths exceeds the expected value by a certain factor.

Chernoff bound

If $X = \sum_i X_i$, X_i are i.i.d 0-1 random variables.

$$\Pr(X \geq (1+\delta)u) \leq e^{-u\delta^2/3} \quad \text{for } 0 < \delta < 1 \text{ and } u \geq \mathbb{E}[X]$$

$$\Pr(X \geq (1+\delta)u) \leq \left(\frac{e^{1+\delta}}{(1+\delta)^{(1+\delta)}} \right)^u \quad \leftarrow \text{more general.}$$

Prob. that it exceeds the expected value for some edge is at most the no. of edges \times prob. for a single edge.

If this is small, it bounds the prob that the maximum exceeds the expected value.

For any edge, expected value of no. of paths is $\leq W$

$$\Rightarrow \sum_i \text{expected no. of } (s_i, t_i) \text{ paths containing } e \leq W$$

If $W \geq c \ln(n)$, choosing $\delta = \sqrt{\frac{c \ln n}{W}} \leq 1$

$$\Pr(\text{no. of paths exceeds } W + \sqrt{cW \ln n}) \leq e^{-\frac{W}{3} \left(\frac{c \ln n}{W} \right)} = \frac{1}{n^{c/3}}$$

Since there are at most n^2 edges, probability that it exceeds for some edge is at most $n^{2-c/3}$.

For $c > 6$, the prob. that the max congestion is $\geq W + \sqrt{cW \ln n}$

$$\text{is } < \frac{1}{n^{c/3-2}} \text{ which } \rightarrow 0 \text{ as } n \rightarrow \infty. \quad \left(\frac{c}{3} > 2 \right) \quad \left[\text{this is a high probability event} \right]$$

$$\text{If } W \geq c \log n, \text{ then } W + \sqrt{cW \ln n} \leq 2W$$

If we only assume that $W \geq 1$, [W can be small], use $u = c \ln n$ and $\delta = 1$

$$\Rightarrow \Pr(X \geq 2 \log n W) \leq e^{-c \log n / 3} = n^{-c/3}$$

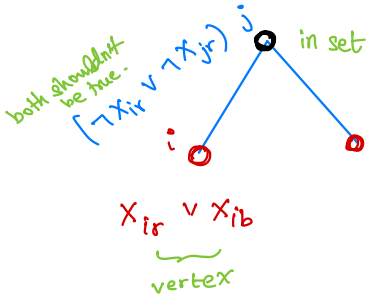
3 - Coloring Dense Graphs

Given a graph that is 3-colorable. Find a coloring with as few colors as possible.
(ideally 3-colors).

For graphs in which each vertex has degree $d \geq \delta n$ for some δ (dense graphs)
we can find this efficiently.

Select a "small" subset of vertices (randomly) s.t. every vertex not in the subset is adjacent to some vertex in the subset [small = $O(\log n)$]

Then, find a coloring for the small subset, every other vertex has only 2-colors possible which can be solved using 2-SAT.



Enumerate over all colors for $\log n$ size set
We get $3^{\log n} = \text{poly}(n)$.

Multi-way Cuts

Given an undirected graph, +ve integer edge weights and k terminal vertices s_1, \dots, s_k .

Find the min-wt. set of edges whose removal disconnects each s_i from all other s_j .

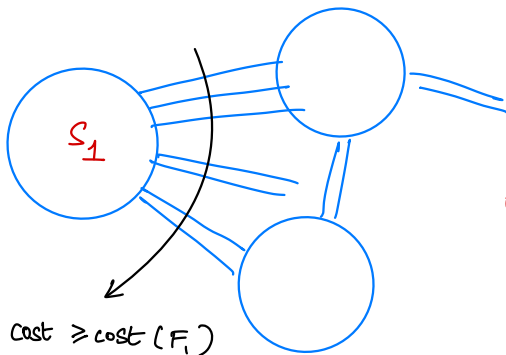
2-approximation

For all s_i , find a minimum cut that separates it from all other terminals.

This can be done by adding a dummy sink and adding ∞ weight edges to other terminals.

Take such a cut for each s_i and take the union of these. This gives a k -way cut.

Let F_i be the cut that separates s_i from all others [in $A(\mathbb{I})$]

Optimal

Each edge is counted at most twice, so,

$$\min_{\text{cut}} k\text{-way} \geq \frac{1}{2} (\text{cost}(F_1) + \dots + \text{cost}(F_k))$$

$$\geq \frac{1}{2} A(\mathbb{I})$$

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

Different LP Formulation

Partition the vertex set into k -parts s.t. s_i belongs to the i^{th} part and sum of weights of edges with endpoints in different parts is minimized.

Assign a unit vector to each vertex where i^{th} coordinate is 1 if the vertex is in the i^{th} part ($x_v = [x_{v_1}, x_{v_2}, \dots, x_{v_k}]$ for all v)

$\forall e=(u,v), z_{uv} \rightarrow 1$ iff (u,v) in different parts

$$z_{uv} \geq x_{ui} - x_{vi} \quad \forall i, e=(u,v)$$

$$z_{uv} \geq x_{vi} - x_{ui}$$

$$\sum_{i=1}^k x_{ui} = 1 \quad \forall u \leftarrow \text{for vertices other than } S_i$$

$$x_{S_i} = \hat{e}_i \quad \forall i=1, \dots, k$$

$$\min \sum_{e=(u,v)} c_{uv} z_{uv}$$

This can be re-written as,

$$\text{cost} = \frac{1}{2} \sum_{e=(u,v)} c_e \|x_u - x_v\|_1$$

Each x_u lies in the k -dimensional sample s.t. $\sum x_{ui} = 1$. For S_i , x_u is \hat{e}_i

Idea

- Solve LP Optimally
- Use randomized rounding

Define the ball of radius r around S_i as $\{u \mid \frac{1}{2} \|e_i - x_u\|_1 \leq r\}$

Choose an r uniformly at random in $(0, 1)$. Choose a permutation of $1, 2, \dots, k$ uniformly at random [call it π].

At the i^{th} step, for $1 \leq i \leq k-1$, assign all unassigned vertices in the r -ball around S_{π_i} and assign them to the π_i^{th} partition [all of these vertices assigned e_{π_i}]

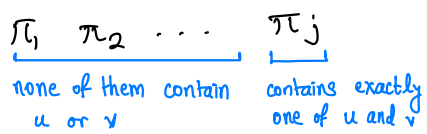
Anything that is left unassigned is assigned to the vector e_{π_k} .

Claim: Expected cost of the integral solution is at most

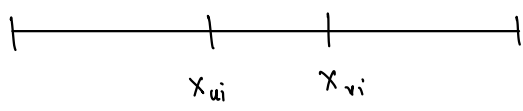
$$\frac{3}{4} \sum_{(u,v) \in E} c_{uv} \|x_u - x_v\|_1 \leq \frac{3}{2} \text{ (optimal LP cost)}$$

\Leftarrow Prob. that an edge belongs to the cut obtained is at most $(\frac{3}{4}) \|x_u - x_v\|_1$

An edge (u,v) is in the cut iff \exists an index i in permutation s.t. $B(s_i, r)$ must contain exactly one of (u,v) [i is the smallest such index]



$$\Pr(X_u \in B(s_i, r)) = X_{ui} \quad \left[\text{since } \sum X_{ui} = 1, \text{ for the ball to contain } u, \frac{1}{2}((1-X_{ui}) + (1-X_{vi})) \leq r \right]$$



Probability that for an index i , exactly one of (u,v) belongs to $B(s_i, r)$ is $|X_{ui} - X_{vi}|$

Let l be the index s.t. $(1 - X_{ul})(1 - X_{vl})$ is min. amongst all i

Note: We don't need to sample a random order of vertices. We can just use the order of $(1 - X_{ui})(1 - X_{vi})$ and its reverse, and take the min. 21/3

$$u \rightarrow (X_{u1}, X_{u2}, \dots, X_{uk})$$

$$v \rightarrow (X_{v1}, X_{v2}, \dots, X_{vk})$$

↓
index that maximises the coordinate among u and v
 $l = \arg \max (\max(1 - X_{ul}, 1 - X_{vl}))$

when going in increasing order probability of this index being i for $i > l = 0$

when going in descending order is 0 for $i < l$

$$\Pr(\text{edge } e \text{ is in the cut when going in increasing order}) = \sum_{i=1}^l |X_{ui} - X_{vi}|$$

$$\begin{aligned} \text{Average cost of the sum of the two} &= \frac{1}{2} \sum_e c_e (\sum_{i=1}^k |X_{ui} - X_{vi}| + X_{ul} - X_{vl}) \\ &= \text{LP-cost} + \frac{1}{2} \sum_e c_e |X_{ul} - X_{vl}| \end{aligned}$$

Also easy to show that $\|X_u - X_v\|_1 \leq \frac{1}{2} \|X_u - X_v\|_1$

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

Multicut

Given k pairs of vertices (s_i, t_i) find a min weight subset of edges whose removal disconnects s_i from t_i

Note: Multi-way cut in trees can be done using DP.

Multi-cut for trees is a generalisation of vertex cover.

Set cover formulation

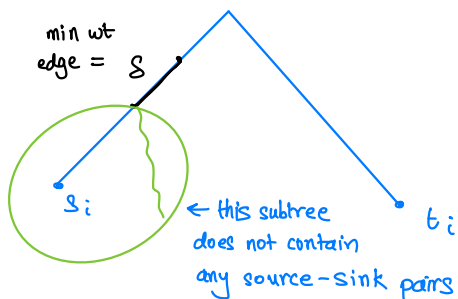
$$\min \sum c_e x_e$$

$$\text{s.t. for every } s_i - t_i \text{ path, } \sum_{e \in p} x_e \geq 1$$

In case of trees, we can get a primal dual algorithm that gives a 2-approximation.

- Root the tree at any vertex
- select the path s_i, t_i s.t. the least common ancestor of s_i and t_i has max depth
- Increase its dual variable until some edge becomes tight

Now, we find an integral solution whose cost is at most twice the dual solution
Let this path be from s_i to t_i

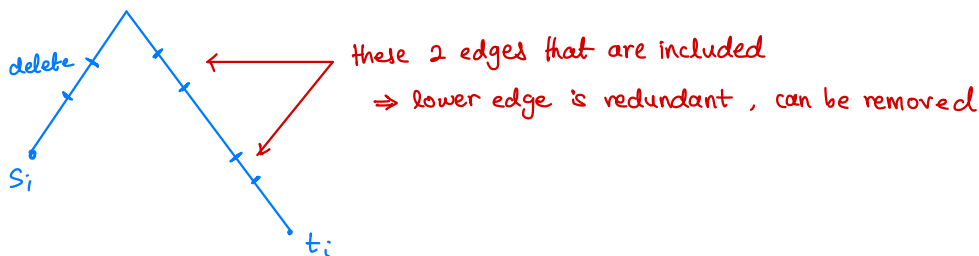


This edge also covers all pairs for which exactly one of the vertices is in the subtree

So, the problem is reduced to a new tree with that subtree removed and the upper subtree weight reduced by 8
(weight of edges along the path in upper subtree)

By induction, we find an integral solution whose cost is at most twice the dual.

On coming back to the original problem, dual increases by δ .



\Rightarrow Total increase in integer cost at most 2δ

General graphs : $O(\ln k)$ approx

In this case, we solve the actual LP

$$\begin{aligned} \min \quad & \sum c_e x_e \\ \forall s_i, t_i \quad & \sum_{e \in P} x_e \geq 1 \end{aligned}$$

Solve the LP and use rounding.

Cuts \rightarrow some $x_e \rightarrow$ LP relax $\rightarrow x_e$ as a distance \rightarrow a metric

26/3

Steiner is the "complement" of Multiway cut

- pick min wt subset of edges to keep st. $\forall i \neq j$ $s_i - s_j$ has a path

In general graphs, we need to cover (expo) paths by edges

$$\min \sum c_e x_e$$

$$\sum_{e \in P_i} x_e \geq 1, \text{ for each } P_i \text{ from } s_i \text{ to } t_i$$

Exp many constraints but there is a separating oracle. Given an assignment of values x_e , feasibility (or a violating constraint path), can be checked by finding shortest paths from $s_i - t_i$ with x_e being the weights of edges (\neq verifying it's always ≥ 1)

Another trick : Dynamic program the LP formulation

We basically want : shortest path $s_i - t_i$ (with wts x_e) ≥ 1

\Rightarrow instead of searching over paths, dynamic program the answer in the LP formulation

y_u^i for each u, i : indicates distance of vertex u from s_i
(weights x_e)

$$\min \sum c_e x_e$$

$$y_{t_i}^i \geq 1 \text{ for all } i$$

$$\text{for each edge } e = (u, v), y_v^i \leq y_u^i + x_e$$

$$y_u^i \leq y_v^i + x_e$$

$$y_{s_i}^i = 0$$

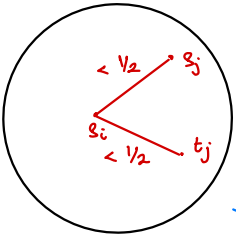
$O(kn+m)$ constraints [polynomial]

Solve any of the LPs, we have
opt solution x_e^* , let

$$V^* = \sum_e c_e x_e^*$$

Consider the distance metric of
shortest dist. w.r.t. weights x_e

$c_e \equiv \text{area}$ V^* is like volume.
 $x_e \equiv \text{length}$



Ball of radius
 $r < 1/2$

A ball of radius $< \frac{1}{2}$
around any s_i (any vertex)
cannot contain both elements of
any source-sink pair

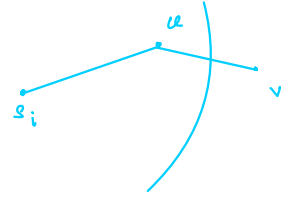
\therefore undirected, there is $s_j - t_j$
 < 1 via s_i

(LP is feasible)

(& obv, $t_i \notin B(s_i, r < \frac{1}{2} < 1)$)

$$B(s_i, 0) = \{s_i\}$$

$$B(s_i, r) = \{v \in V : d(s_i, v) \leq r\}$$



The volume $\text{vol}(B(s_i, \infty))$
 $= \text{vol}(V) = V^*$

$$V(s_i, r) = \frac{V^*}{k} + \sum_{\substack{e=uv \\ u, v \in B(s_i, r)}} c_e x_e^* + \sum_{\substack{e=uv \in \delta(B(s_i, r)) \\ \text{the cut edges}}} \underbrace{c_e (r - d(u, s_i))}_{< x_e}$$

Claim : For any s_i , \exists a radius $r \in (0, \frac{1}{2})$ such that the cost ($\sum c_e$) of the cut
 $\delta(B(s_i, r)) \leq 2 \ln(k+1) V(s_i, r)$

Given this, we get a $4 \ln(k+1)$ - approximation algorithm

- Pick an s_i , get an r
- Include $\delta(B(s_i, r))$ in the multicut and remove all the edges of the ball from the graph (removing the "subtree")
- Repeat this process with any further source-sink pairs

if $\text{vol} \downarrow$ by V , multicut cost \uparrow by $\leq 2 \ln(k+1) V$ \leftarrow Actually, the whole cut edge x_e is removed, not just the part contributing to the volume
next time $k' \leq k$ ($< k$) actually

multicut \uparrow by $\sum_{e \in \delta(B(s_i, r))} c_e \cdot 1$
 $\underbrace{\hspace{1cm}}_{\text{call this } F_i}$

cut: $\sum_{e \in F_i} c_e$, $\text{vol dec. by } V^*/k$

$V_i = \sum c_e x_e$
for edges e
removed in i^{th} step

$$s_0, v_1 = \sum_{e \text{ in ball}} c_e x_e + \sum_{e \in F_1} c_e x_e \geq r - d(s_i, u) \geq r - \frac{v^*}{k}$$

⇒ Atmost k steps

$$F_1 \leq 2 \ln(k+1) \left[v_1 + \frac{v^*}{k} \right]$$

$$F_2 \leq \quad \quad \quad \left[v_2 + \frac{v^*}{k} \right]$$

...

$$F_{l \leq k} \leq \quad \quad \quad \left[v_l + \frac{v^*}{k} \right]$$

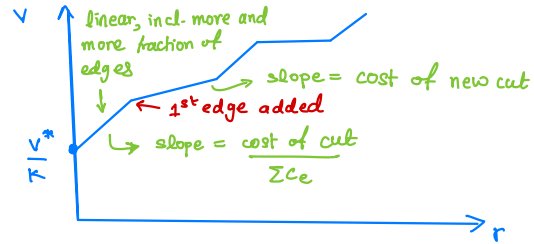
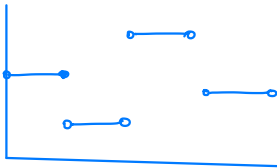
$$v_1 + v_2 + \dots + v_l = v^*$$

$$\begin{aligned} \Rightarrow \sum F_i &\leq 2 \ln(k+1) \left[v^* + \leq v^* \right] \\ &\leq 4 \ln(1+k) v^* \\ &\leq 4 \ln(1+k) \text{ Opt.} \end{aligned}$$

It remains to establish the claim. → Simply look at how the volume varies with r as $r: 0 \text{ to } 1/2$.

at $r = 0^+ \quad v(s_i, r) : v^*/k$

the claim is simply that $\frac{v'(r)}{v(r)}$ is small at some point



v is piecewise linear, note that it may not be continuous.

$$\text{note } v(1/2) \leq \frac{v^*}{k} + v^*$$

$$\text{let } h(r) := \frac{v'(r)}{v(r)} ; \text{ want to upper bound } \min_{r \in (0, 1/2)} h(r)$$

$$= (\ln v(r))'$$

$$\begin{aligned} \text{By "MVT", } \exists x \in (0, 1/2) &\leq \frac{v^*}{k} (1+k) \leq v^*/k \\ \text{(fn not cont, but can still get it to work)} &\frac{\ln v(1/2) - \ln v(0)}{1/2} = h(x) \end{aligned}$$

$$\text{i.e. } h(x) \leq 2 \ln(k+1)$$

finding x , deterministic the min will wlog happen just before a breakpoint
⇒ just find the value at the breakpoints

Vertex Multiway Cut

Given an undirected graph G with a specified set S of terminals and a cost to each vertex not in S . S is an independent set in G . Find a min cost set of vertices not in S whose removal separates all the terminals.

LP Formulation

$x_v \leftarrow$ for each vertex not in S

$$\min \sum c_v x_v$$

$$\sum_{\substack{\text{all internal} \\ \text{vertices } v}} x_v \geq 1 \quad \text{for any path from } S_i \text{ to } S_j$$

Claim: This LP always has an optimal solution where each $x_i \in \{0, \frac{1}{2}, 1\}$ [$\frac{1}{2}$ -integral]

Rounding: Include all vertices with $x_v \geq \frac{1}{2}$

Given any optimal solution, use complementary slackness to convert it into $\frac{1}{2}$ -integral soln.

Dual: $\max \sum_{\substack{\text{for all } S_i-S_j \\ \text{paths } P}} f_p$ $f_p \leftarrow$ flow in path P

$$\sum_{\substack{\text{all paths} \\ P \text{ that include } v}} f_p \leq c_v \quad \forall v \notin S$$

Complementary Slackness

If some $x_v > 0$ in the optimal, then the sum of flows through v is equal to c_v

If a path has non-zero flow, its length must be exactly 1

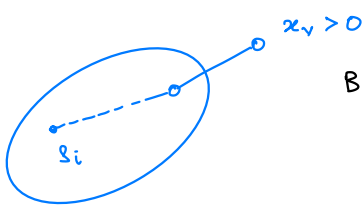
[length = sum of x_v along the path]

$S_i \leftarrow$ set of vertices at distance 0 from S_i

[reachable a path having $x_v = 0$, including the vertex at the ends]

Note: S_i, S_j are disjoint $\forall i \neq j$

Moreover, no edge in S_i has an edge to a vertex in S_j . Also, any edge from S_i goes to a vertex with $x_v > 0$

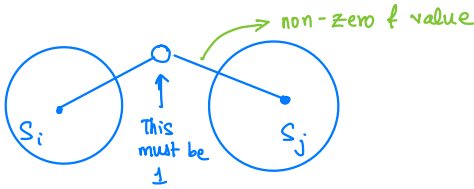


$B(S_i) =$ Set of vertices that are not in S_i but adjacent to some vertex in S_i

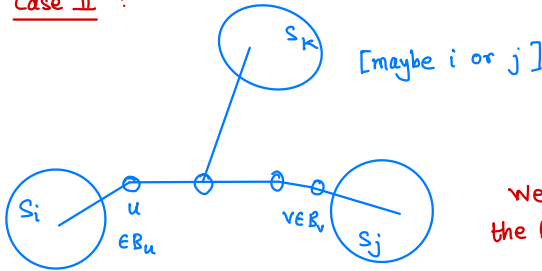
Case I : There is a vertex in $B_i \cap B_j$

\Rightarrow That vertex has $x_v \geq 1$

Also, all paths that have non-zero flow have length exactly 1 [complementary slackness]



Case II :



Claim : Any path with non-zero flow value will pass through at most 2 boundary vertices [\nexists such a w]

We know that $x_u > 0$ and $x_v > 0$. Also, the length of this path is exactly 1 [by compl. slack.]

Also, $k \neq i$ or $k \neq j$. If $k \neq i$, the path from k to i has length < 1 . Since i to j was 1 and $x_v > 0 \Rightarrow$ contradiction.

Construct the $1/2$ -integral solution as follows :

If a vertex belongs to 2 or more boundary sets, set $x_v = 1$, and if it belongs to exactly 1 boundary set, set $x_v = 1/2$.

\Rightarrow This is a feasible soln for the original LP.

Claim : This is an optimal soln

Proof : We show that this satisfies complementary slackness

$$\text{Dual cost} = \sum_{\substack{\text{all paths } P \\ \text{with } f_p > 0}} f_p$$

Also, in our soln, all paths with $f_p > 0$ have length 1. The other condition is satisfied because any vertex with non-zero x value also had non-zero x value in the original soln.

\Rightarrow This gives a $1/2$ -integral optimal soln.

Balanced cuts

1/4

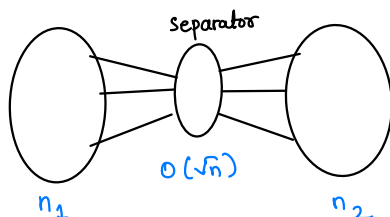
A b -balanced cut, for some $0 < b \leq 1/2$, in a graph is a partition of the vertex set into two parts each of size at least $\lfloor bn \rfloor$ and at most $\lceil (1-b)n \rceil$.
 $b = 1/2$ is called minimum bisection of the graph.

Cost = sum of wt of edges with an endpoint in both parts (cut weight)
want min cost b -balanced cut

Separator theorems help design efficient divide and conquer algorithms

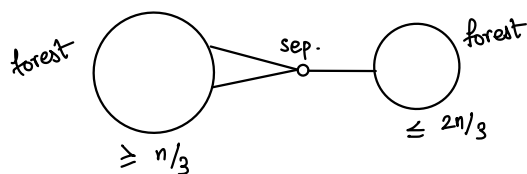
Planar Separator Theorem

In any planar graph with n vertices there exists a set of $O(\sqrt{n})$ vertices whose removal separates the graph into two parts with at least $\lfloor n/3 \rfloor$ and at most $\lceil 2n/3 \rceil$ vertices



Example application:
Fast APSP on planar graphs.

E.g. for trees: One vertex separator exists



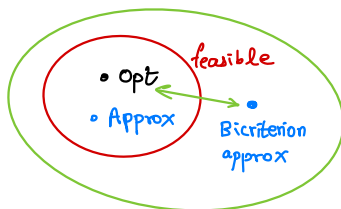
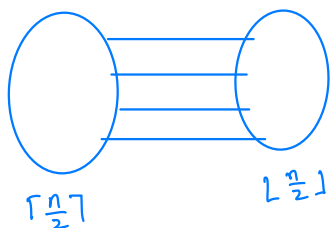
$$b = \frac{1}{2}$$

$$\text{Opt}(\frac{1}{2})$$

minimum bisection

Bicriterion - approximation

Allow more solutions and find one such relaxed solution whose cost is not much more than Opt.



Find a $\frac{1}{3}$ -balanced cut whose cost is at most $O(\log n) \times$ minimum bisection

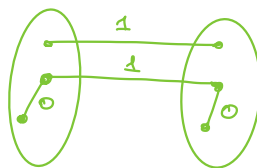
$$\min \sum c_e x_e$$

$x_e \leftarrow$ length assigned to an edge e

for any path P from u to v

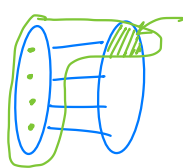
$d_{uv} \leftarrow$ indicates distance between vertices u, v

$$d_{uv} \leq \sum_{e \in P} x_e \leftarrow d_{uv} \text{ is at most the shortest path}$$



LP form. for which any bisection is a feasible solution
 \downarrow
 $\text{opt} \leftarrow \text{min bisection.}$

S of $\lceil \frac{2n}{3} \rceil + 1$ vertices



at least $\lceil \frac{2n}{3} \rceil + 1 - \lfloor \frac{n}{2} \rfloor$ vertices.

$$\geq (\frac{2}{3} - \frac{1}{2})n = \frac{n}{6}$$

for any vertex $u \in S$, $|S| \geq \lceil \frac{2n}{3} \rceil + 1$

$$\sum_{v \in S} d_{uv} \geq (\frac{2}{3} - \frac{1}{2})n$$

} exp many such constraints \leftarrow can be solved by ellipsoid method!

Given values of x_e , d_{uv} it can be checked in polynomial time.

$$d_{uv} \leq \sum_{e \in P} x_e \leftarrow d_{uv} \text{ must be less than shortest path } u \text{ to } v \text{ w/ } x_e \text{ as weights}$$

$$\forall S: |S| \geq \lceil \frac{2n}{3} \rceil + 1, \sum_{v \in S} d_{uv} \geq \frac{n}{6}$$

Fix u . (iterate over u)

$u \quad v_0 \quad v_1 \quad \dots \quad v_{\lceil \frac{2n}{3} \rceil}$

For each u , order rem.

vertices in increasing order
of d_{uv} , check S formed
by first $\frac{2n}{3}$ vertices.

Check that sum of first $\frac{2n}{3}$ values $\geq n/6$.

Hence, we get solution is feasible, or we get a violating constraint.

This ensures that optimal LP solution can be found in polynomial-time using ellipsoid.

Round the LP solution to find a $\frac{1}{3}$ -balanced cut

For every vertex u , $\exists v$ s.t. $d_{uv} \geq 1/6$ (n vertices, sum $\geq n/6$, PHP)

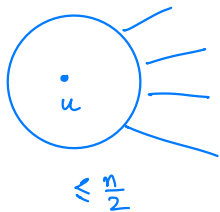


Consider balls of radius $r < \frac{1}{12}$ around u, v

As in the multicut problem, we can find ball around u, v with radius $r < 1/12$
such that the cut is at most $O(\log n)$ volume of the ball.

Construct a $\frac{1}{3}$ -balanced partition

- Choose smaller of two balls, ($\#$ vertices), and put all vertices in the ball in one part



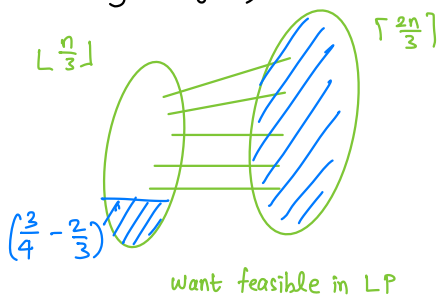
- Put all edges leaving the ball in cut temporarily.
- Delete all vertices in selected ball and edges incident to it.
- Repeat till size of set is $< \lfloor \frac{n}{3} \rfloor$ (S)

Current size S , $|S| < \lfloor \frac{n}{3} \rfloor$, no. of remaining vertices = $n - |S|$

The number added to S is at most $\frac{1}{2} \times (n - |S|)$

New size is at most $\frac{n}{2} + \frac{|S|}{2} \leq \lceil \frac{2n}{3} \rceil$

Generalising we get,



$< \frac{1}{3}$ balanced cut works by changing constants in previous LP

$$|S| \geq \frac{3n}{4}$$

$\frac{1}{4}$ -balanced cut whose cost is at most $O(\log n) \times \text{opt}(\frac{1}{3})$

For $b < b'$

$b \leq \frac{1}{3}$, can get a b -balanced cut with cost $\leq O(\log n) \text{opt}(b')$

e.g. for 0.4-balanced

$$\underbrace{0.4}_{|S|} + \underbrace{\frac{0.6}{2}}_{(\frac{n-|S|}{2})} = 0.7 > 0.6, \text{ so same method doesn't work.}$$

Approximating Metrics

2/4

Metric: Finite set of points with distance $d(u, v) \geq 0$ specified for each pair

$$d(u, w) \leq d(u, v) + d(v, w)$$

$$d(u, v) = 0 \text{ iff } u = v$$

Tree Metric:

- Points are vertices of a tree
- Each edge of a tree has a specified length
- Distance between 2 vertices is the length of the (unique) path between them

For TSP with tree metric, length of optimal tour = $2 \times \sum \text{edge lengths}$

Given a metric space M , a tree approximation of M is a mapping f that maps points of M to vertices of some tree T s.t. for points u, v

$$d(u, v) \leq T(f(u), f(v)) \leq \alpha d(u, v)$$

α = distortion or dilation

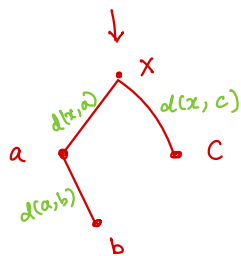
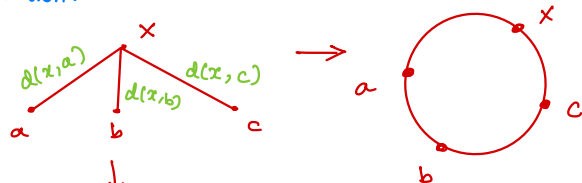
Note : We can allow vertices in the tree that do not correspond to points in M .

The tree metric may not be a good approximation in many cases. For example, consider an unweighted n -cycle with $d(u,v) = \text{shortest path length}$.

Claim : In any tree metric with same no. of vertices, $\text{dilation} \geq n-1$

Proof : There are only $n-1$ pairs of adjacent vertices. Also each edge in tree has weight ≥ 1 .

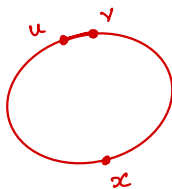
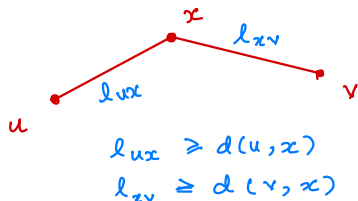
First, we show that we can remove degree 3 vertices in the tree without increasing distortion.



\Rightarrow There is a tree with only $\text{deg} \leq 2$ vertices (i.e. a path) with the same dilation.

Let (u,v) be adjacent vertices in the cycle that are not adjacent in the path (tree)

Let x be between u,v in the path



$$d(u,x) + d(x,v) = n-1$$

$$l_{ux} + l_{vx} \geq n-1$$

Hence, $d(u,v) = 1$, $T(f(u), f(v)) = n-1$ giving a dilation of $n-1$

For any metric space M with n points, \exists a randomized algorithm that constructs a tree approximation with expected dilation $O(\log n)$

$$\forall u,v \quad \mathbb{E}[T(u,v)] \leq O(\log n) d(u,v) \quad [\text{not the expectation of the max}]$$

Construct one random tree (or equivalently)

Construct a large collection of trees s.t. the average distance in the trees b/w u and v is $O(\log n) d(u,v)$

Construction :

Scale s.t. $d(u,v) \geq 1$ for all $u \neq v$

- Leaf nodes correspond to points in the metric space
- Root node corresponds to all points
- Intermediate nodes correspond to subsets of points

Choose a random permutation π of the points.

Choose r uniformly at random in $[\frac{1}{2}, 1)$

At level i , choose value of radius to be $2^i r$, radius $\in [2^{i-1}, 2^i)$

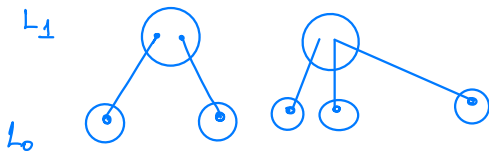
At level 0, all the r balls are disjoint



Partition at level i is constructed by considering points in the given permutation $(\pi_1, \pi_2, \dots, \pi_n)$. Take π_1 , all points within ball of π_1 in one part.

Continue until you find a partition.

For each node at level i , we construct the children at level $i-1$ by partitioning parts in level i by considering balls of radius r_{i-1} in the order of permutation π .

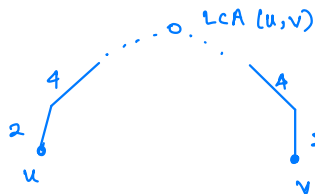


The length of an edge joining a node at level $i-1$ to node at level i is 2^i

Claim : This gives a tree metric where expected distance between u and v is $O(\log n) d(u,v)$

4/4/24

Proof



If nearest common ancestor is at level i ,

$$T(u,v) = 2(2 + 4 + \dots + 2^i) \leq 4 \times 2^i = 2^{i+2}$$

Also, u, v are in the same node at level i , so they are in a ball of radius 2^i from some point.

$$\Rightarrow d(u,v) \leq 2^{i+1}$$

$$\mathbb{E}[T(u,v)] \leq \sum_i 2^{i+2} \Pr(\text{LCA}(u,v) \text{ is at level } i)$$

If LCA at level i , they get separated at level $i-1$

$$\begin{aligned} \Pr(\text{LCA}(u,v) \text{ at level } i) &\leq \Pr(\exists \text{ a vertex } w \text{ that separates } u,v \text{ at level } i-1) \\ &\leq \sum_w \Pr(w \text{ separates } u,v \text{ at level } i-1) \end{aligned}$$

$$\Rightarrow \mathbb{E}[T(u,v)] \leq \sum_i \sum_w 2^{i+2} \Pr(w \text{ separates } u,v \text{ at level } i-1)$$

$$\leq \sum_w 16 d(u,v) \sum_i \Pr(w \text{ sep at } i) \leq \sum_w 16 d(u,v) (\Pr(w \text{ sep } u,v))$$

We need the probability that w is the first vertex that includes either u or v and separates u and v .

Order vertices based on $\min(d(x,u), d(x,v))$. Any vertex in this order appears before w in the permutation then w cannot separate u,v .

The prob. that this happens for j th smallest vertex is atmost $\frac{1}{j}$

w is j th smallest
 \Rightarrow bad case happens if $1 \dots j-1$ comes before j
 (consider $1 \dots j$ as identical j $\frac{1}{j}$)

$$\Rightarrow \text{summed over all } w, \text{ atmost } \sum_{j=1}^n \frac{1}{j} = O(\log n)$$

$$\therefore \mathbb{E}[T(u,v)] = d(u,v) O(\log n)$$

Linear Arrangement

Graph with cost of each edge. Number the vertices $1, 2, \dots, n$ such that

$$\sum_{e=u,v} c_e |f(u) - f(v)| \text{ is minimized}$$

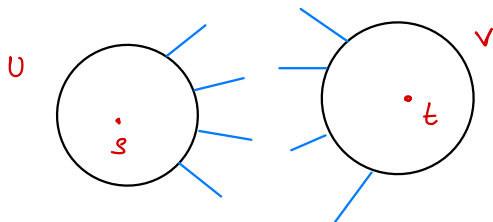
LP Formulation

$$\min \sum_{u,v} c_{u,v} d_{u,v}$$

$$d_{u,v} \geq 1 \quad \forall u,v$$

for any vertex u and S s.t. $u \notin S$

$$\sum_{v \in S} d_{u,v} \geq \frac{1}{4} |S|^2$$



U, V are disjoint but need not be a partition.

$$g(U) = \sum_{\substack{\text{all edges } u,v \\ u \in U, v \notin U}} w_e$$

Given s and t , find subsets U, V , $s \in U, t \in V$ such that $g(U) + g(V)$ is minimized

Claim: $g(S)$ is submodular

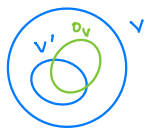
Proof: upon adding vertex, increase more for subset (g', v $v \notin U' \Rightarrow v \notin U$ -)

$$\Rightarrow g(A \cup B) + g(A \cap B) \leq g(A) + g(B)$$

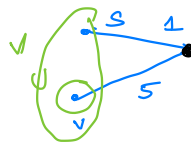
Define $f(U) = \min_{\substack{V \subseteq V(G) \setminus U \\ t \in V}} g(V)$

Claim: $f(U)$ is submodular

Proof: $U \subseteq U'$ then if $s \in \overline{U'}$ is added, i.e. $V' \subseteq V$, $s \in V'$ is removed



$U \subseteq V$ then opt cuts also subsets? \rightarrow not nec.



We know a general polytime algorithm for an arbitrary submodular function

Objective = $f(U) + g(V) \Rightarrow$ also submodular.

Exercise 4.40, 4.41

Q1: PS3



Processor is busy during this time.

Also, all tasks executed in this period have index $\leq j$ [on ordering by w_i/t_i]

$$\Rightarrow c_j \leq r_j + \sum_{i=1}^j t_i$$

$$\Rightarrow \text{Total weighted completion time} \leq \sum_{j=1}^n w_j r_j + \sum_{j=1}^n \sum_{i=1}^j w_j t_i$$

Also, $\sum w_j r_j \leq \text{opt pre-emptive schedule}$.

If we assume $r_j = 0 \ \forall j$. Then the cost can only reduce. In this case,

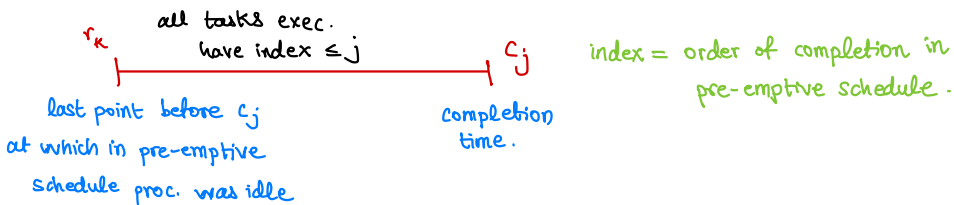
$$\text{opt} = \sum_{j=1}^n \sum_{i=1}^j w_j t_i$$

$$\Rightarrow A(\pi) \leq 2 \text{opt}(\pi)$$

Hence, proved.

(a) Question seems wrong.

Given any pre-emptive schedule we can convert it into a non pre-emptive schedule s.t. the completion time of any task atmost doubles.



$$\Rightarrow c_j \leq r_k + \sum_{i=1}^j t_i \quad (\text{for some } k \leq j)$$

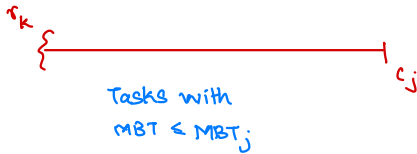
$$r_k \leq c_k^* \quad (\text{completion time in pre-emptive schedule})$$

$$C_j^* \geq \sum_{i=1}^j t_i$$

$$\Rightarrow \boxed{C_j \leq 2C_j^*}$$

Also, the order of MBT gives a 2-approximation for the optimal $\sum w_i C_i$ with pre-emption. Hence, overall, this gives a 4-approximation for $\sum w_i C_i$

Execute tasks non pre-emptively in order of MBT



C_j^* = completion time of j in the pre-emptive schedule

$$\Rightarrow r_k \leq C_j^*$$

$$C_j \leq r_k + \sum_{i=1}^j t_i$$

$$\text{Also, } C_j^* \geq \frac{\sum_{i=1}^j t_i}{2} \Rightarrow C_j \leq C_j^* + 2C_j^* = 3C_j^*$$

Hence, get a 3-approx.

EndSem

- No problems from ch 8
- Mostly from 1-4, 7
- Randomization : Simple Problems
- Things that can be argued from first principles.