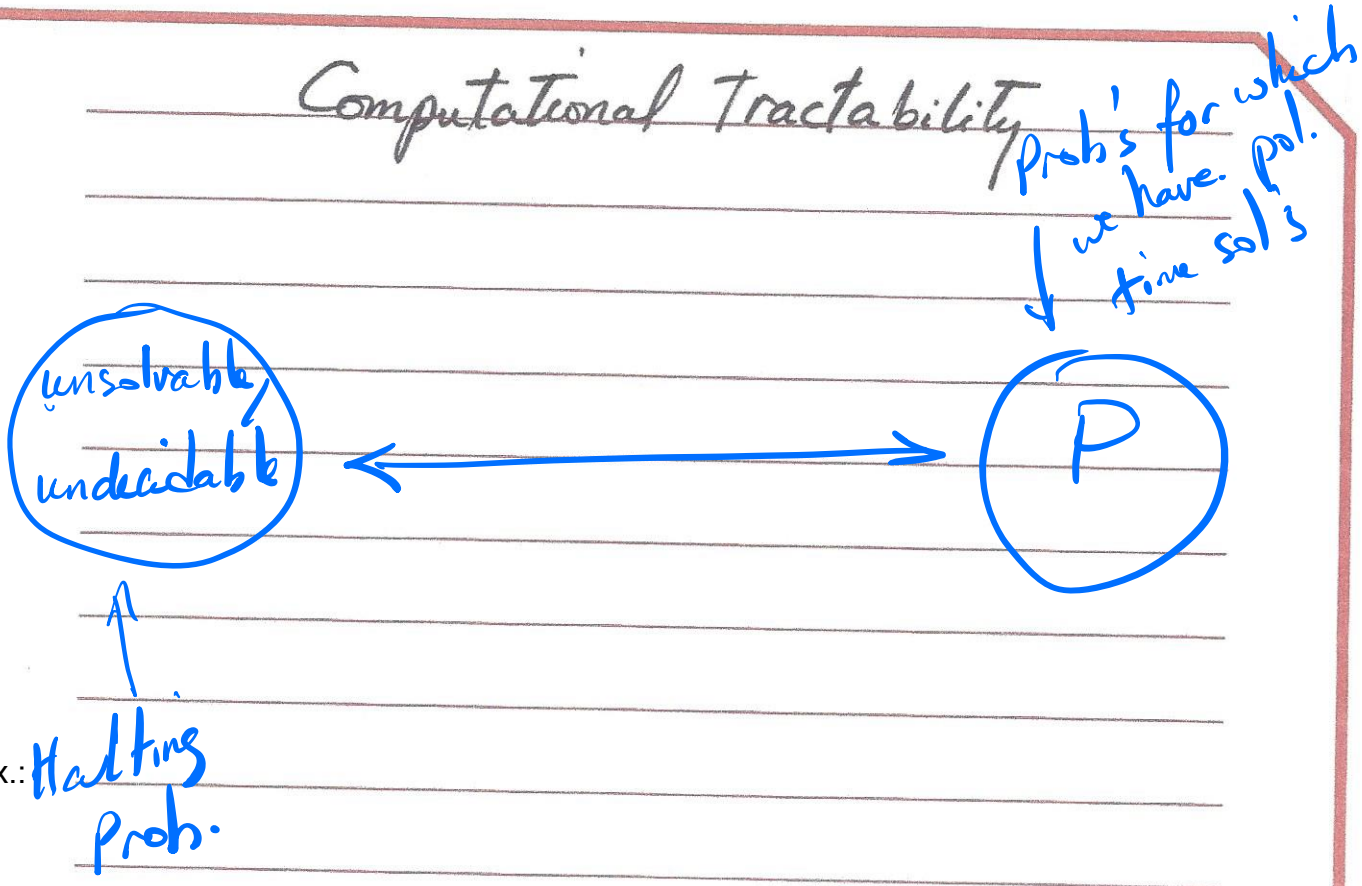
Computational Tractability



Plan: Explore the space of computationally hard problems to arrive at a mathematical characterization of a large class of them.

Technique: Compare relative difficulty of different problems.

Loose definition: If problem X is at least as hard as problem Y, it means that if we could solve X, we could also solve Y.

Formal definition:

$Y \leq_p X$ (Y is polynomial time reducible to X)

if Y can be solved using a polynomial number of standard computational steps plus a polynomial number of calls to a blackbox that solves X.

Suppose $Y \leq_p X$, if X can be solved in polynomial time, then Y can be solved in polynomial time.

Suppose $Y \leq_p X$, if Y cannot be solved in polynomial time, then X cannot be solved in pol time.

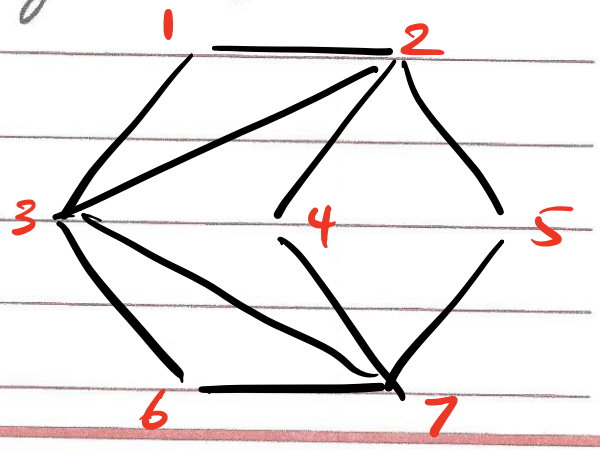
Independent Set

Def. In a graph $G = (V, E)$, we say that a set of nodes $S \subseteq V$ is "independent" if no two nodes in S are joined by an edge.

$\{1, 4, 6\}$, $\{1, 4, 5, 6\}$

$\{3, 4, 5\}$

$\{1\}$



Independent set problem

- Find the largest independent set in graph G.

(opt. version)

- Given a graph G, and a no. k, does G contain an indep set of size at least k?

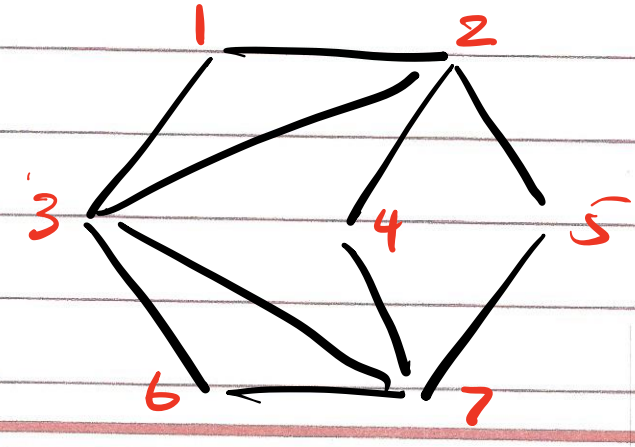
(decision version)

Vertex Cover

Def. Given a graph $G = (V, E)$, we say that a set of nodes $S \subseteq V$ is a vertex cover if every edge in E has at least one end in S.

$\{1, 2, 3, 4, 5, 6, 7\}$

$\{2, 3, 7\}$



Vertex Cover problem

- Find the smallest vertex cover set in G.

(opt. version)

- Given a graph G and a no. k does G contain a vertex cover set of size at most k ?

(decision version)

FACT: Let $G=(V,E)$ be a graph, then S is an independent set if and only if its complement $V-S$ is a vertex cover set.

Proof: A) First suppose that S is an independent set

1- U is in S, and V is not
$\Rightarrow$ V-S $\rightarrow$ will have V and not U

2- V is in S, and U is not
$\Rightarrow$ V-S $\rightarrow$ will have U and not V

3- Neither V nor U is in S
$\Rightarrow$ V-S will have both V & U

B) Suppose V-S is a vertex Cover...

Claim: Indep. set $\leq_p$ vertex cover

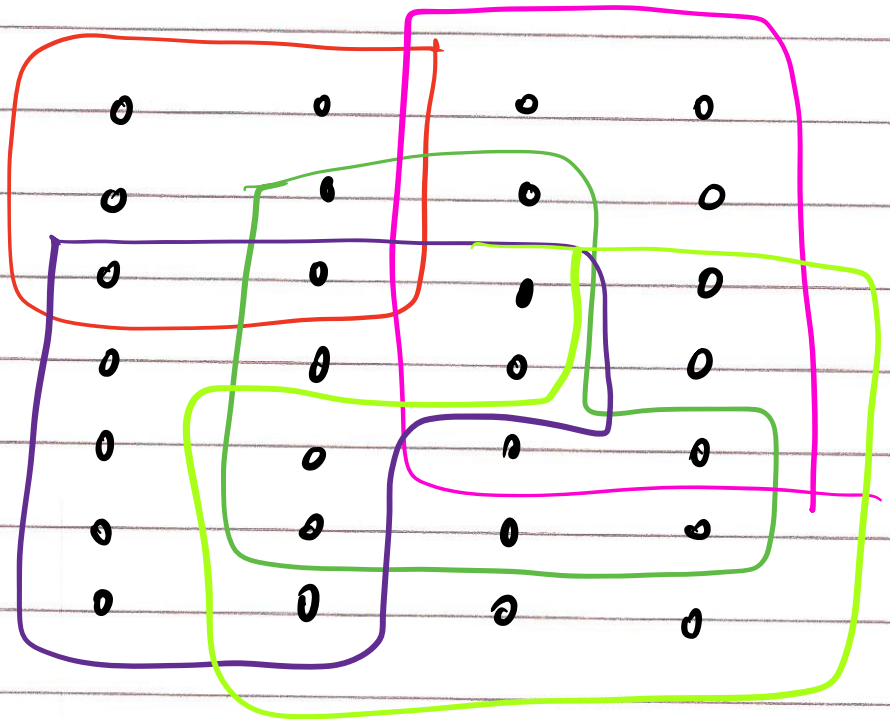
Proof: If we have a blackbox to solve vertex cover, we can decide if G has an independent set of size at least k, by asking the blackbox if G has a vertex cover of size at most $n-k$.

Claim: Vertex Cover $\leq_p$ Indep set

Proof: If we have a blackbox to solve independent set, we can decide if G has a vertex cover set of size at most k, by asking the blackbox if G has an indep. set of size at least $n-k$.

Set Cover Problem

Given a set U of n elements, a collection $S_1, S_2, \ldots S_m$ of subsets of U, and a number k, does there exist a collection of at most k of these sets whose union is equal to all of U.



Claim: Vertex Cover $\leq_p$ Set Cover

$S_1 = \{(1,2),(1,3)\}$

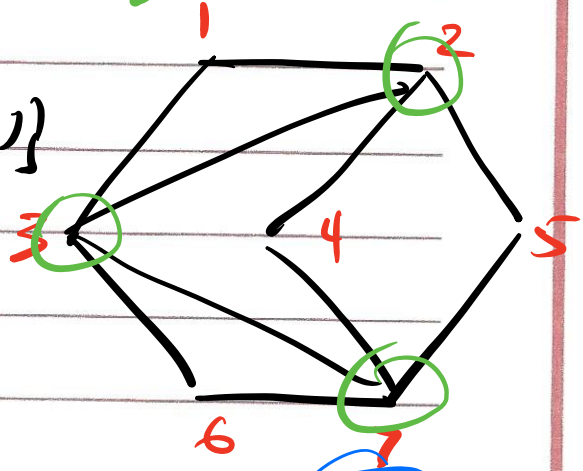
$S_2 = \{(1,2),(2,3),(2,4),(2,5)\}$

$S_3 =$ ' ' '

' ' '

$S_7 =$ ' ' '

k



G, k

Need to show that G has a vertex cover of size k, iff the corresponding set cover instance has k sets whose union ~~is~~ contains ~~equivalent to~~ all edges in G.

Proof:

A) If I have a vertex cover set of size k in G, I can find a collection of k sets whose union contains all edges in G.

B) If I have k sets whose union contains all edges in G, I can find a vertex cover set of size k in G.

Reductions Using Gadgets

- Given n Boolean variables $x_1 \dots x_n$, a clause is a disjunction of terms $t_1 \vee t_2 \vee \dots \vee t_\ell$ where $t_i \in \{x_1 \dots x_n, \bar{x}_1 \dots \bar{x}_n\}$

- A truth assignment for X is an assignment of values 0 or 1 to each x_i:

$(x_1 \vee \bar{x}_2)$ → $x_1 = 0, x_2 = 1$ ✗

$x_1 = 1, x_2 = 1$ ✓

- An assignment satisfies a clause C if it causes C to evaluate to 1.

- An assignment satisfies a collection of clauses if

$C_1 \wedge C_2 \wedge \dots C_k$

evaluates to 1.

ex. $(x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee \bar{x}_3) \wedge (x_2 \vee \bar{x}_3)$

$x_1 = 1 \quad , \quad x_2 = 1 \quad , \quad x_3 = 1$

$x_1 = 0 \quad , \quad x_2 = 0 \quad , \quad x_3 = 0$

Problem Statement: Given a set of clauses $C_1 \ldots C_k$ over a set of variables $X = \{x_1, \ldots x_n\}$ does there exist a satisfying truth assignment?

Satisfiability Problem (SAT)

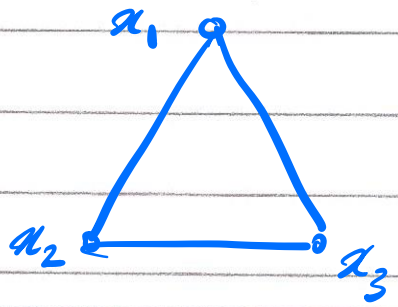
Problem statement: Given a set of clauses $C_1 \ldots C_k$ each of length 3 over a set of variables $X = \{x_1 \ldots x_n\}$ does there exist a satisfying truth assignment?

3SAT

Claim: 3SAT $\leq_p$ Independent Set

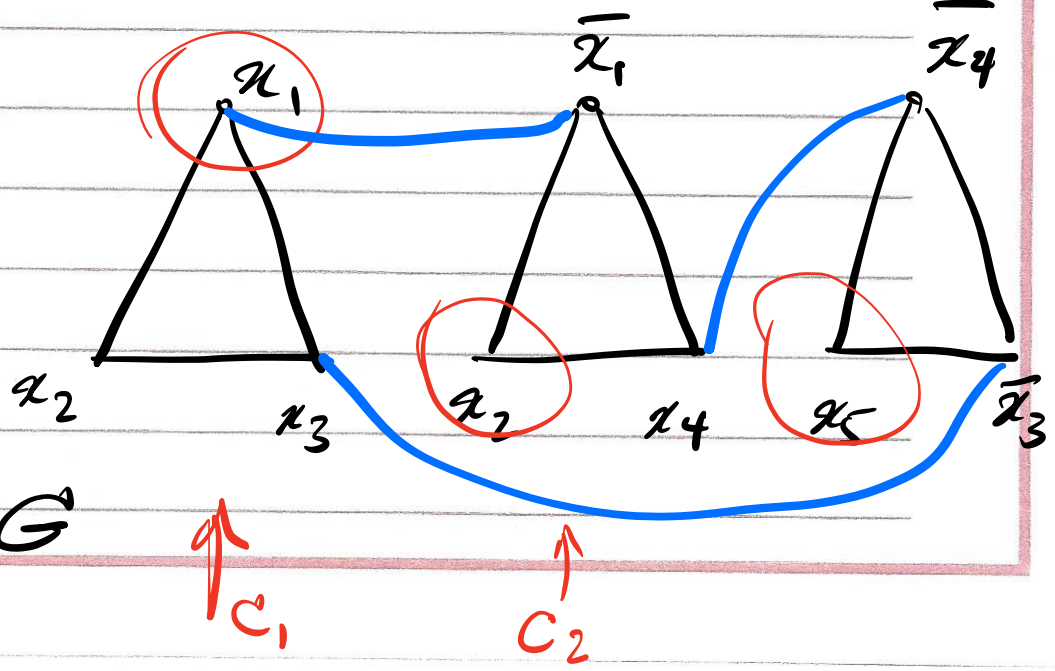
Plan: Given an instance of 3SAT with k clauses, build a graph G that has an indep. set of size k iff the 3SAT instance is satisfiable.

$(x_1 \vee x_2 \vee x_3)$



ex. $C_1 = (x_1 \vee x_2 \vee x_3)$
$C_2 = (\bar{x}_1 \vee x_2 \vee x_4)$
$C_3 = (\bar{x}_4, x_5 \vee \bar{x}_3)$

$\}$ k



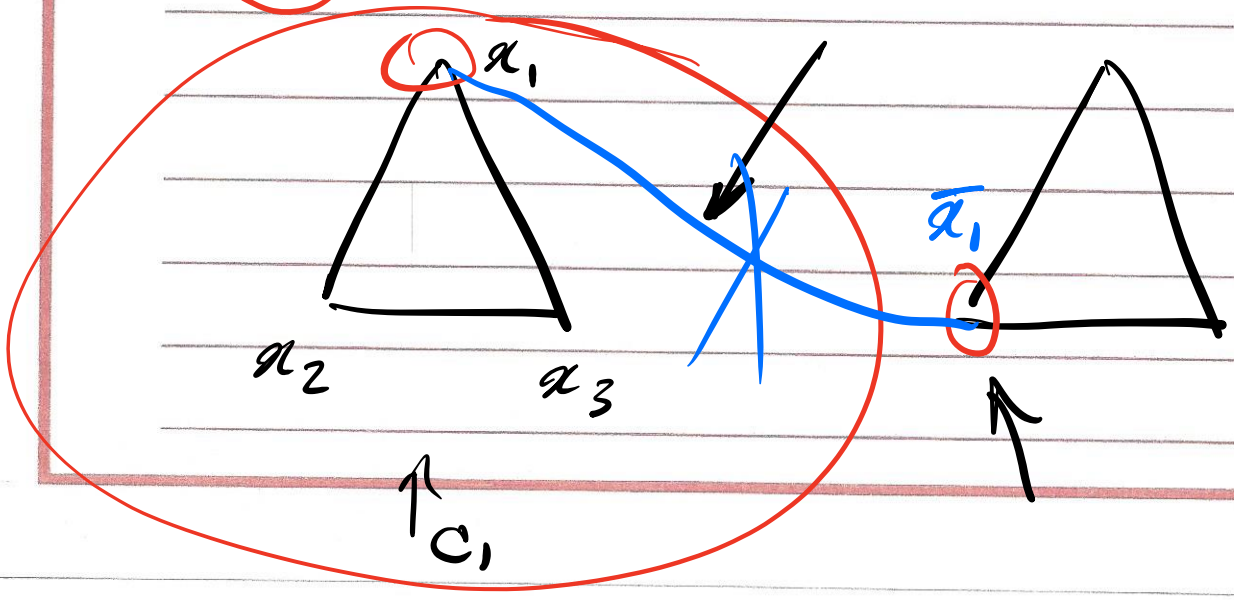
Claim: The 3-SAT instance is satisfiable iff the graph G has an independent set of size k.

Proof: A) If the 3-SAT instance is satisfiable, then there is at least one node label per triangle that evaluates to 1.

Let S be a set containing one such node from each triangle

$C_1 = (x_1 \vee x_2 \vee x_3)$
$C_2 = (\bar{x}_1 \vee x_2 \vee x_4)$
$C_3 = (\bar{x}_4 \vee x_5 \vee \bar{x}_3)$
} k

$x_1 = 1, x_2 = 1, x_3 = 1$
$x_4 = 0, x_5 = 0$



B) Suppose G has an independent set S of size at least k.

if x_i appears as a label in S then set x_i to 1

if $\bar{x}_i$ appears as a label in S then set x_i to 0

if neither x_i nor $\bar{x}_i$ appear as a label in S, then set x_i to either 0 or 1

Efficient Certification

To show efficient certification:

1- Polynomial length certificate

2- Polynomial time certifier

Efficient certification

3-SAT

Certificate t is an assignment of truth values to variables (x_i)

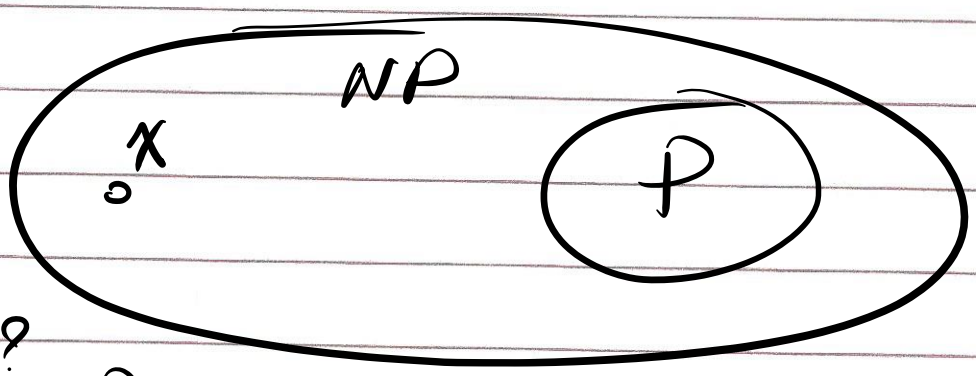
Certifier: evaluate the clauses. If all of them evaluate to 1 then it answers yes.

Indep set

Certificate t is a set of nodes of size at least k in G.

Certifier: check each edge to make sure no edges have both ends in the set
check size of the set $\geq k$
no repeating nodes

Class NP is the set of all problems for which there exists an efficient certifier



$P \stackrel{?}{=} NP$

if $X \in NP$ and for all $Y \in NP$

$Y \leq_p X$, then X is the hardest prob. in NP.

3SAT has been proven to be such a problem

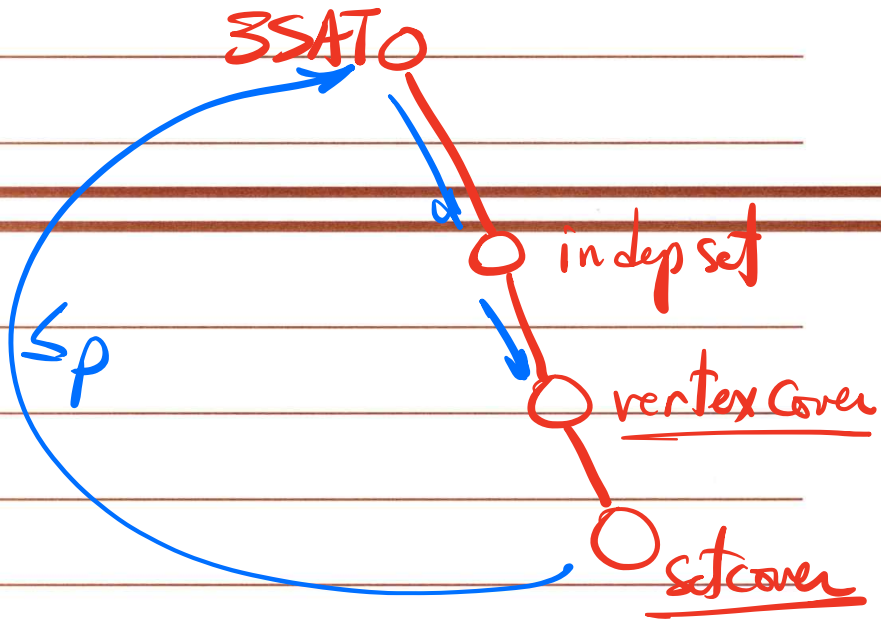
Such a problem is called NP-Complete.

Transitivity

if $Z \leq_p Y$ and $Y \leq_p X$

then $Z \leq_p X$

$3SAT \leq_p$ indep set $\leq_p$ vertex Cover $\leq_p$ Set Cover



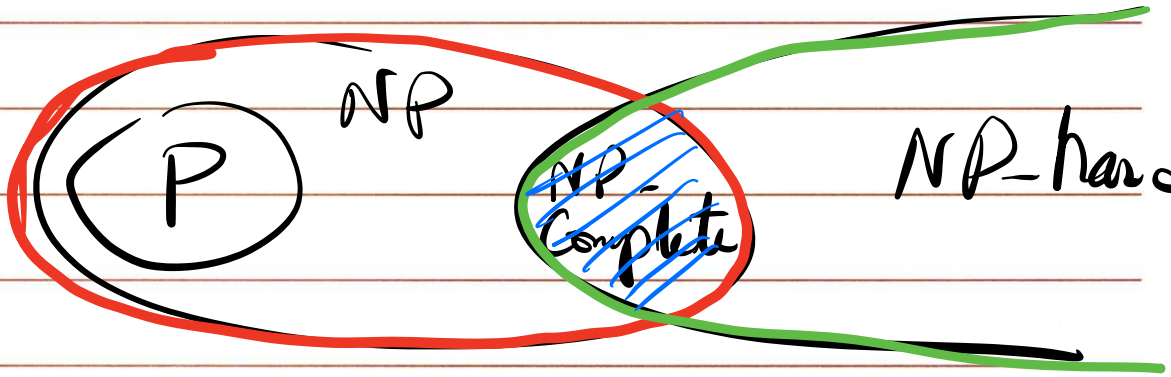
Basic strategy to prove a prob. X is NP-Complete

1- Prove $X \in NP$

2- Choose a problem Y that is known to be NP-Complete

3- Prove that $Y \leq_p X$

NP-hard is the set of problems that are at least as hard as NP-Complete problems.



Discussion 10

1. Given the SAT problem from lecture for a Boolean expression in Conjunctive Normal Form with any number of clauses and any number of literals in each clause. For example,

$$(X_1 \vee \neg X_3) \wedge (X_1 \vee \neg X_2 \vee X_4 \vee X_5) \wedge \ldots$$

Prove that SAT is polynomial time reducible to the 3-SAT problem (in which each clause contains at most 3 literals.)

2. The *Set Packing* problem is as follows. We are given m sets $S_1, S_2, \ldots S_m$ and an integer k. Our goal is to select k of the m sets such that no selected pair have any elements in common. Prove that this problem is **NP**-complete.

3. The *Steiner Tree* problem is as follows. Given an undirected graph $G=(V,E)$ with nonnegative edge costs and whose vertices are partitioned into two sets, R and S, find a tree $T \subseteq G$ such that for every v in R, v is in T with total cost at most C. That is, the tree that contains every vertex in R (and possibly some in S) with a total edge cost of at most C.
Prove that this problem is **NP**-complete.

1. Given the SAT problem from lecture for a Boolean expression in Conjunctive Normal Form with any number of clauses and any number of literals in each clause. For example,

$(X_1 \vee \neg X_3) \wedge (X_1 \vee \neg X_2 \vee X_4 \vee X_5) \wedge \ldots$

Prove that SAT is polynomial time reducible to the 3-SAT problem (in which each clause contains at most 3 literals.)

SAT

(x_1)

$(x_1 \vee x_2)$

$(x_1 \vee x_2 \vee x_3)$

$(x_1 \vee x_2 \vee x_3 \vee x_4)$

3SAT

$(x_1 \vee x_1 \vee x_1)$

$(x_1 \vee x_2 \vee x_1)$

$(x_1 \vee x_2 \vee x_3)$

$(x_1 \vee x_2 \vee s_1) \wedge$

$(\bar{s}_1 \vee x_3 \vee x_4)$

$(x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5)$

$(x_1 \vee x_2 \vee s_1) \wedge$

$(\bar{s}_1 \vee x_3 \vee s_2) \wedge$

$(\bar{s}_2 \vee x_4 \vee x_5)$

2. The *Set Packing* problem is as follows. We are given m sets $S_1, S_2, \ldots S_m$ and an integer k. Our goal is to select k of the m sets such that no selected pair have any elements in common. Prove that this problem is **NP**-complete.

1- Show Set Packing is in NP

Certificate: List of k sets that have no elements in common.

Certifier: check that the intersection between each pair of sets in the list is Null &

there are k sets in the certificate

2- Choose indep set.

3- Show indepset $\leq_p$ Setpacking

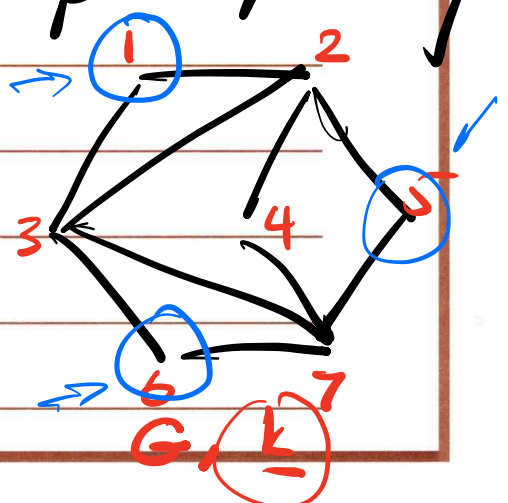
$S_1 = \{(1,2),(1,3)\}$

$S_2 = \{(1,2),(2,3),(2,4),(2,5)\}$

$S_3 =$

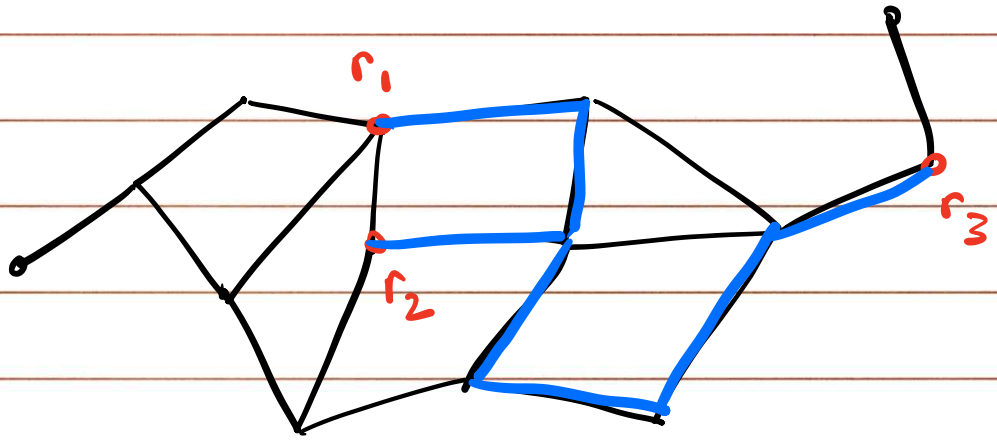
S_6

$S_7 =$



k

3. The *Steiner Tree* problem is as follows. Given an undirected graph $G=(V,E)$ with nonnegative edge costs and whose vertices are partitioned into two sets, R and S, find a tree $T \subseteq G$ such that for every v in R, v is in T with total cost at most C. That is, the tree that contains every vertex in R (and possibly some in S) with a total edge cost of at most C.
Prove that this problem is **NP**-complete.



1- Show Steiner Tree $\in$ NP

a- Certificate : Tree spanning T across all nodes in set R (and maybe some in S) w/ Cost $\leq C$

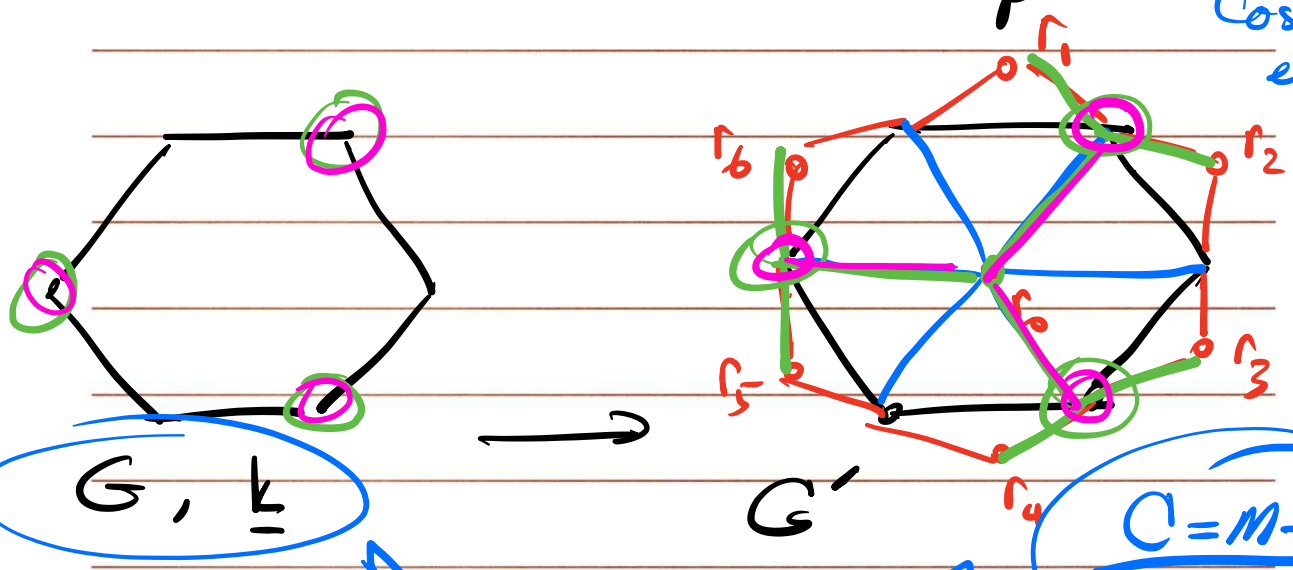
b- Certifier :

- check that T is a tree
- " " Cost of T $\leq C$
- " " T covers all nodes in R.

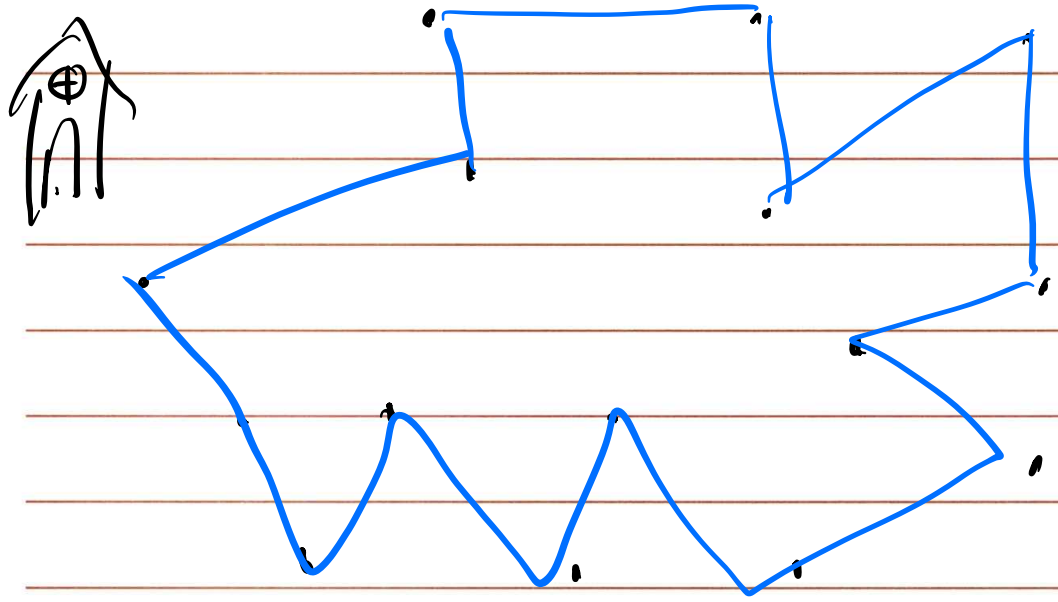
2- Choose vertex Cover

3- Show Vertex Cover $\leq_p$ Steiner Tree

Cost of all edges in G' = 1



Traveling Salesman Problem (TSP)
&
Hamiltonian Cycle



Problem Statement

Given the set of distances, order n cities in a tour $V_{i_1}, V_{i_2}, \ldots V_{i_n}$ with $i_1 = 1$, so it minimizes

$$\sum d(V_{i_j}, V_{i_{j+1}}) + d(V_{i_n}, V_{i_1})$$

Decision version of TSP:

Given a set of distances on n cities and a bound D, is there a tour of length/cost at most D?

Def. A cycle C in G is a Hamiltonian Cycle, if it visits each vertex exactly once.

Problem Statement:

Given an undirected graph G, is there a Hamiltonian cycle in G?

Show that the Hamiltonian Cycle Problem is NP-complete

1- Show HC is in NP

a. Certificate: ordered list of the vertices on the HC.

a. Certifier:

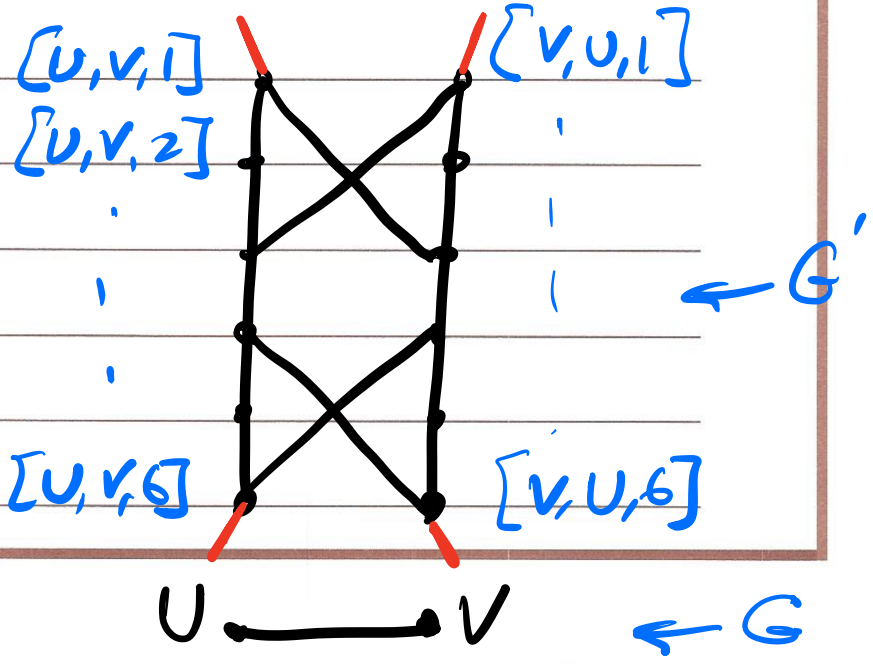
- Check all vertices are covered
- No duplicate nodes
- Check that all adjacent nodes in the list are connected by an edge.
- Check that last & first nodes are connected by an edge.

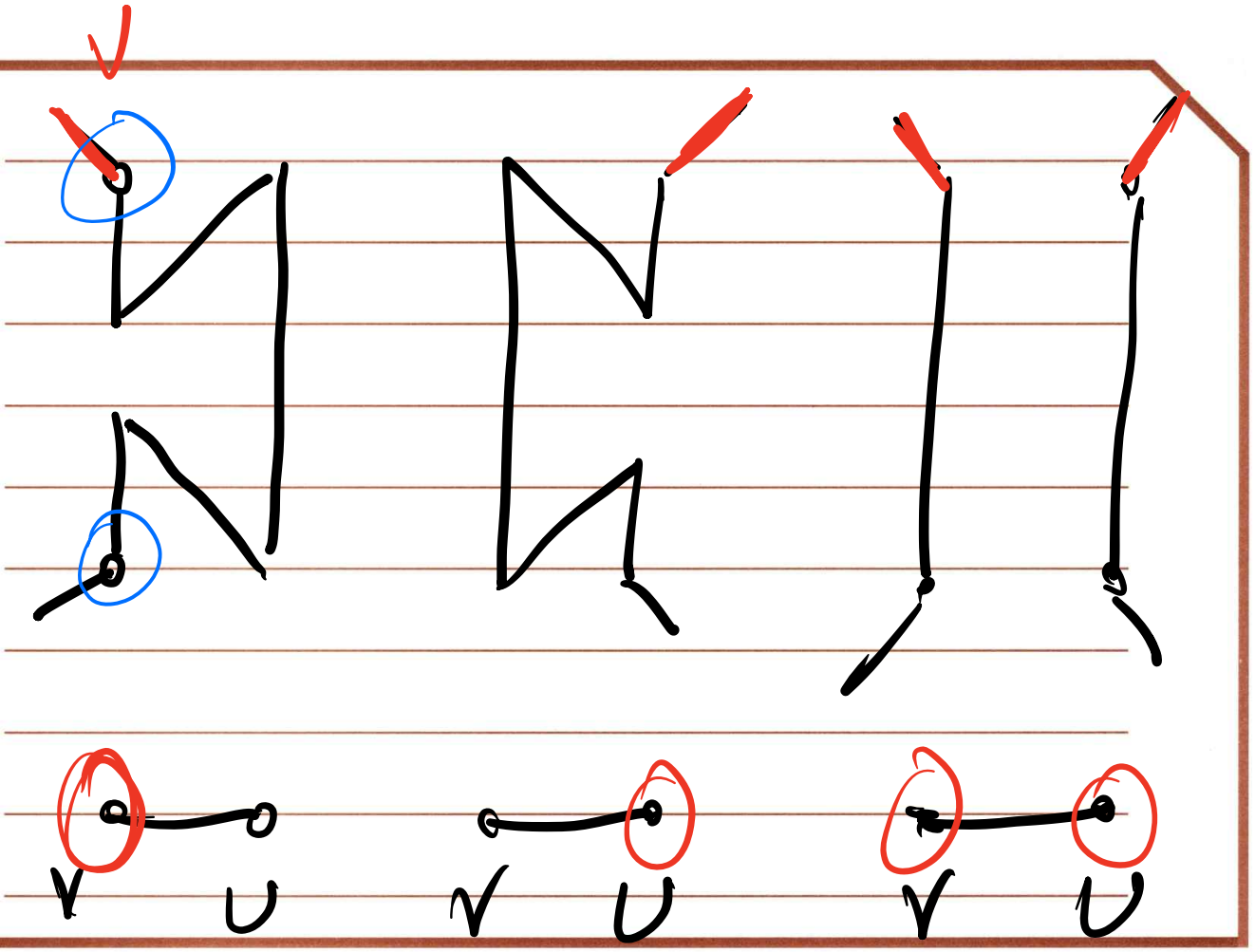
2- Choose Vertex Cover

Plan: Given an undirected graph $G = (V, E)$ and an integer k, we construct $G' = (V', E')$ that has a Hamiltonian Cycle iff G has a vertex cover of size at most k.

Construction of G'

For each edge (v, u) in G, G' will have one gadget W_{vu} with following node labeling:



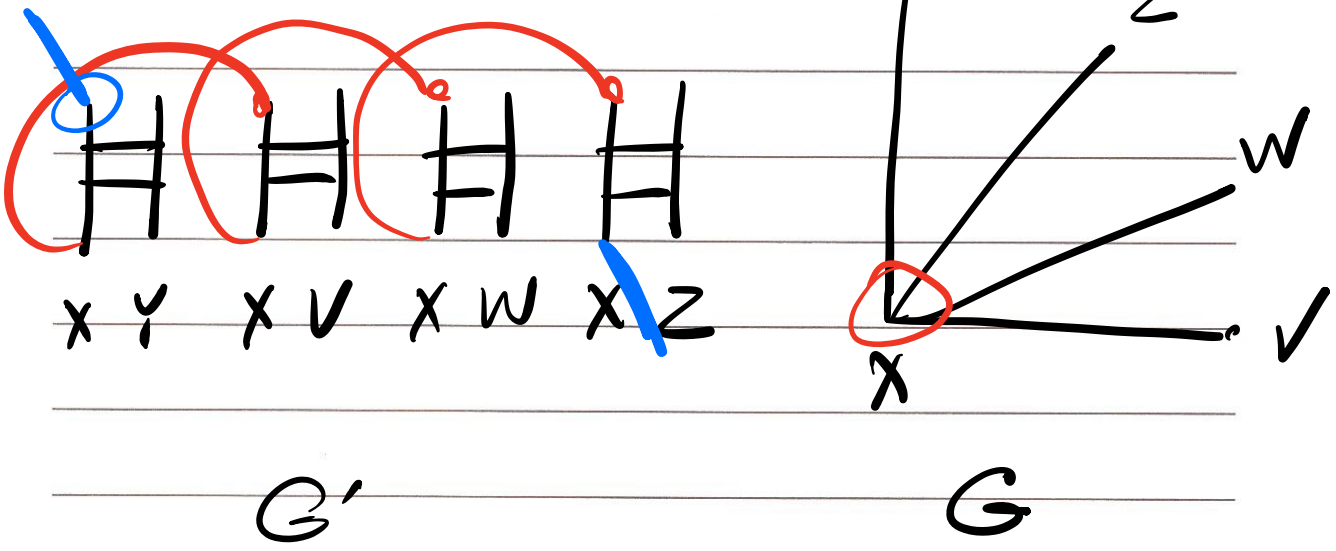

v
u
v
u
v
u

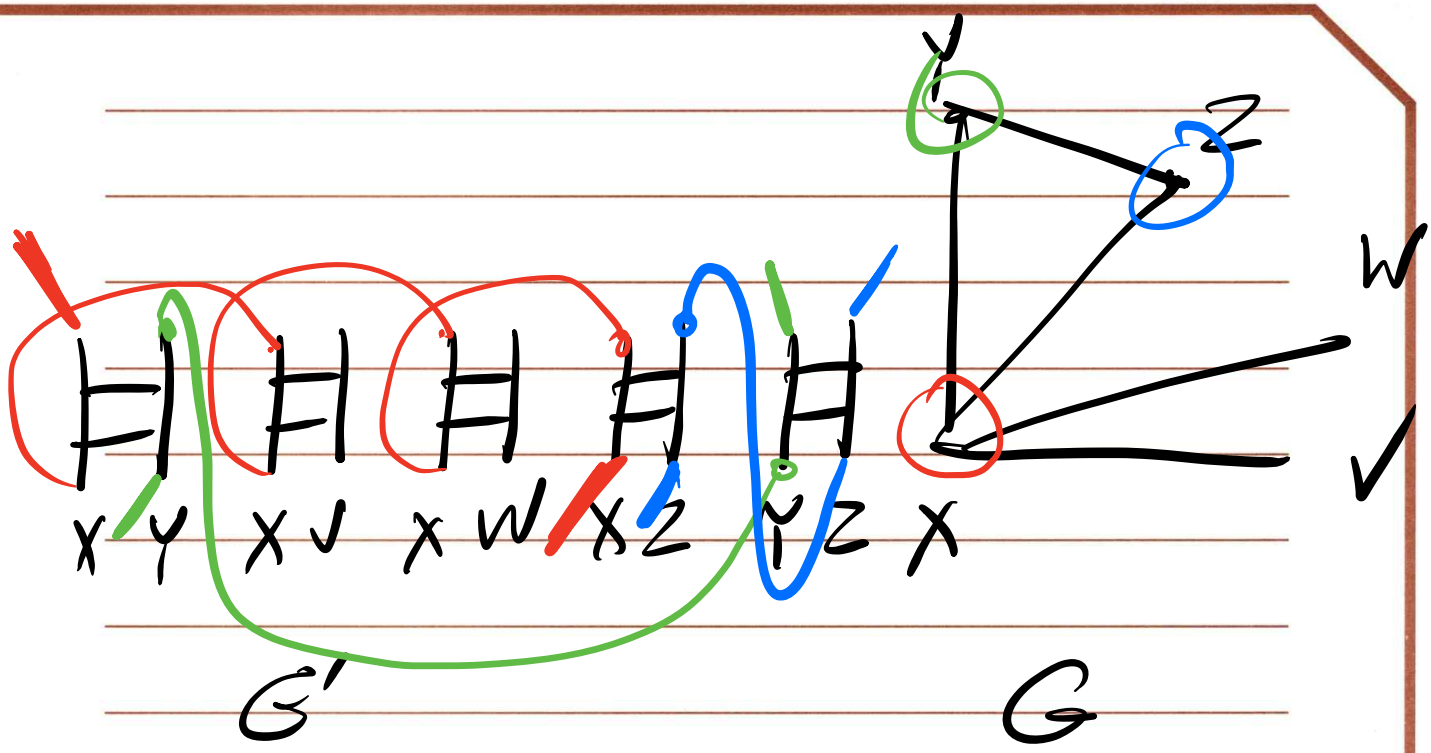
Other vertices in G'

- Selector vertices: There are k selector vertices in G', $S_1, \ldots, S_k$

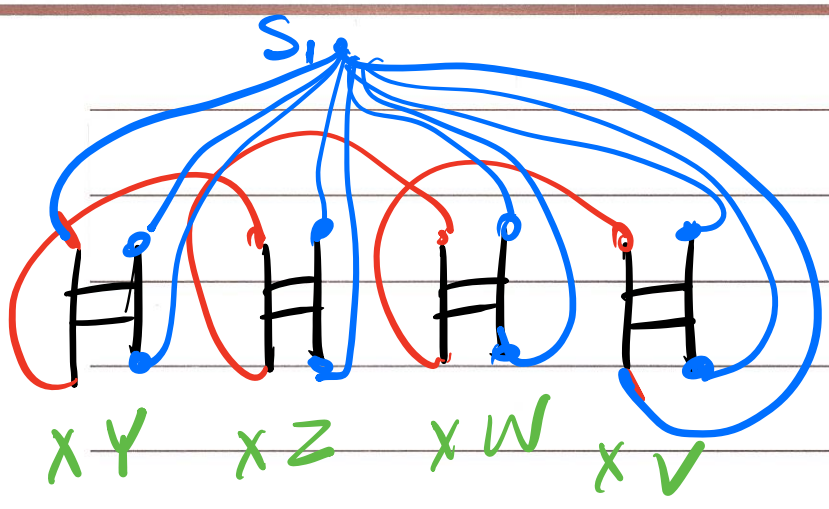
Other edges in G'

1. For each vertex $v \in V$ we add edges to join pairs of gadgets in order to form a path going through all the gadgets corresponding to edges incident on v in G.

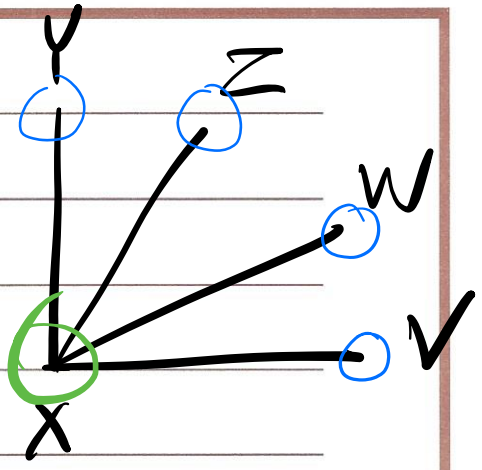



Y
Z
W
X
Y
X
J
X
W
X
Z
Y
Z
X
G'
G

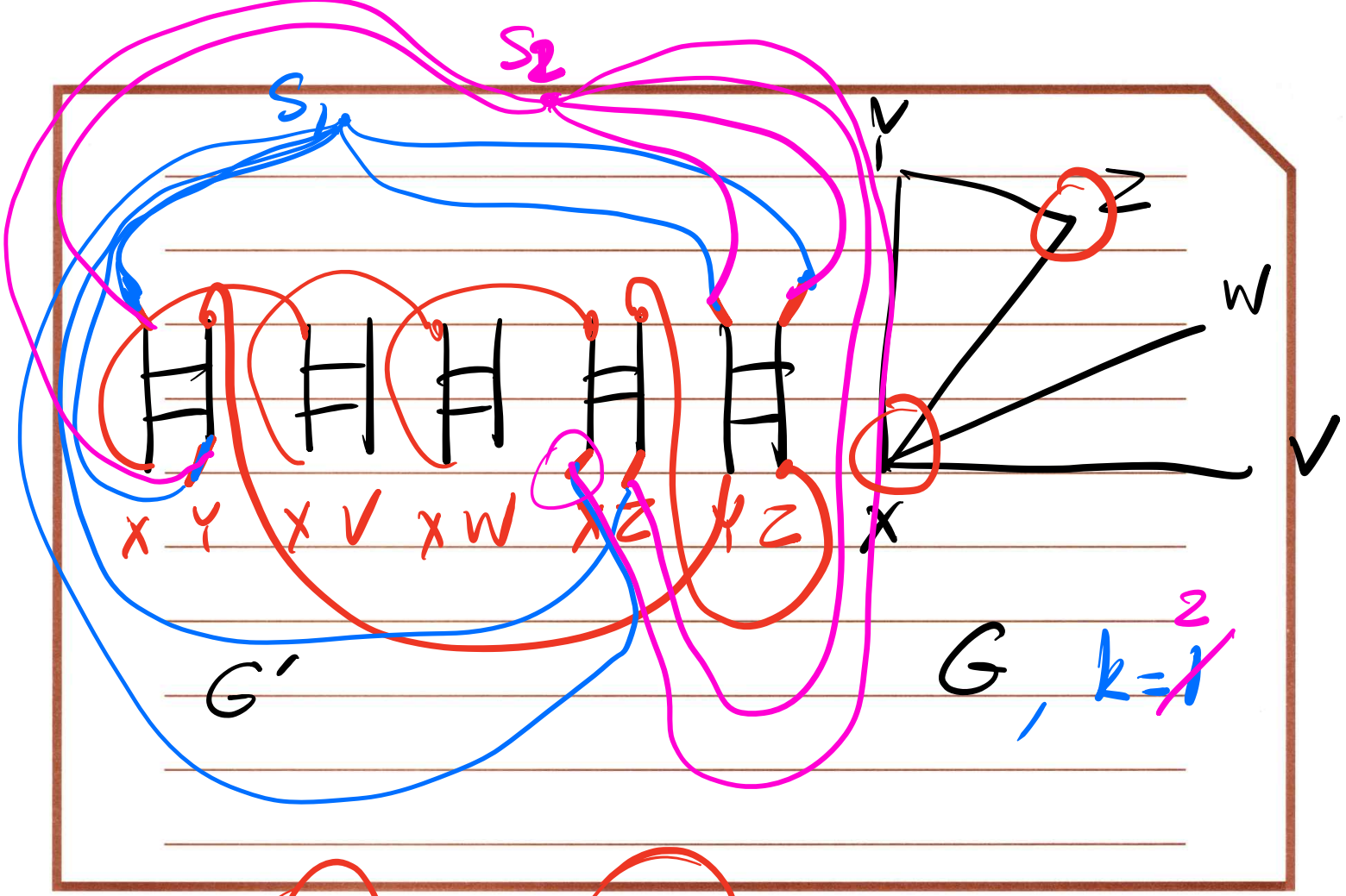
2- Final set of edges in G' join the first vertex $[x, Y, 1]$ and last vertex $[x, Y_{(deg(x))}, 6]$ of each of these paths to each of the selector vertices.

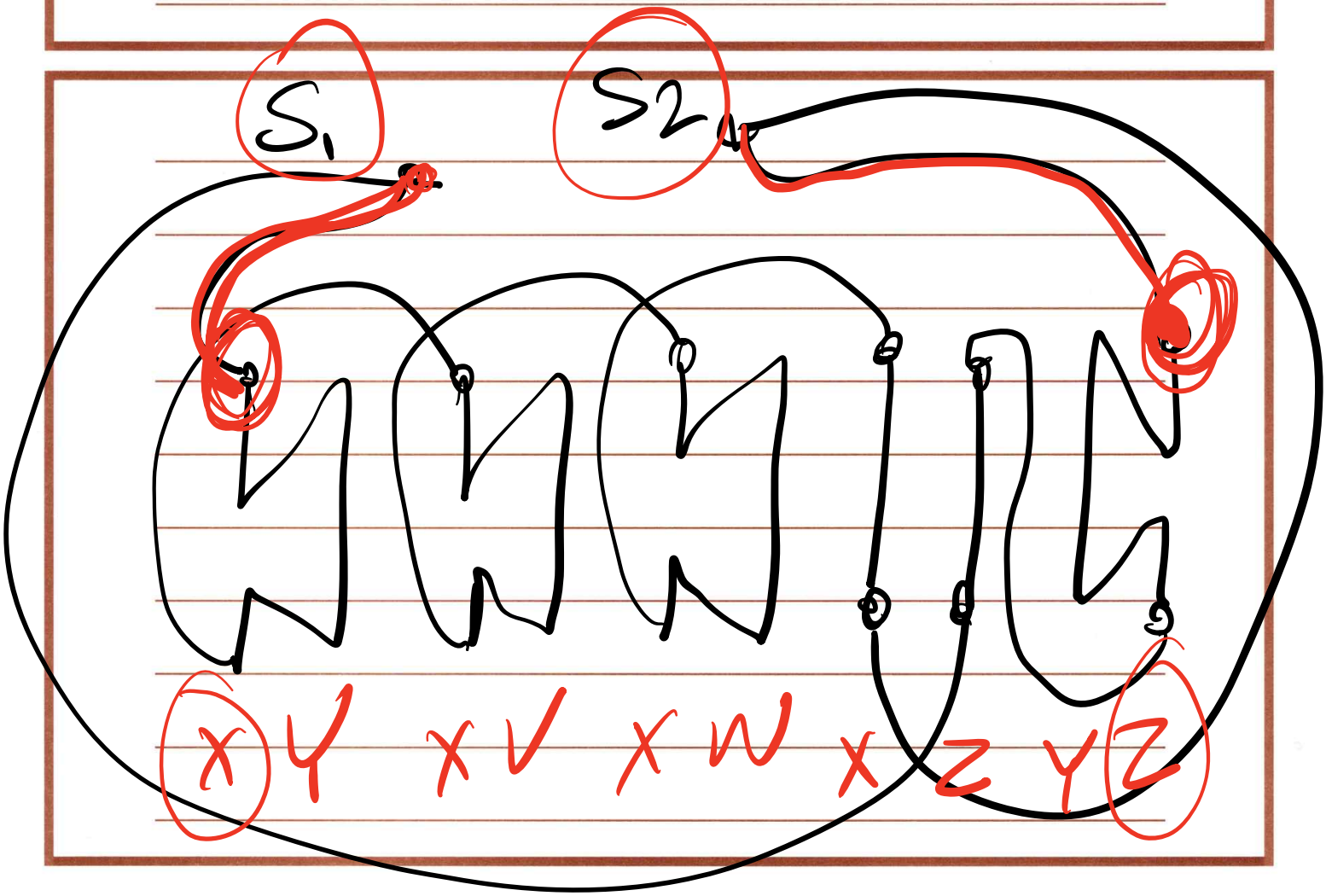


G'



G, k=1

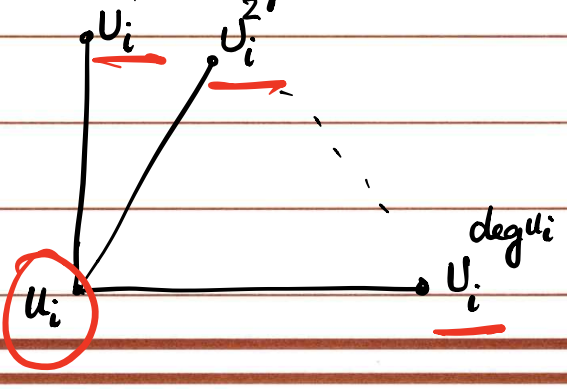

S1
S2
V
Z
W
V
X
x y
x v
x w
x z
y z
G'
G
k=2


S1
S2
x y
x v
x w
x z
y z

Proof: A) Suppose that $G = (V, E)$ has a vertex cover of size $\leq k$. Let the vertex cover set be

$$S = \{u_1, u_2, \ldots, u_k\}$$

We will identify neighbors of u_i as shown here:



Form a Ham. Cycle in G' by following the nodes in G in this order:

start at s_1 and go to

$[u_1, u_1^1, 1]$ ---- $[u_1, u_1^1, 6]$

$[u_1, u_1^2, 1]$ ---- $[u_1, u_1^2, 6]$

$\vdots$

$[u_1, u_1^{\deg u_1}, 1]$ ---- $[u_1, u_1^{\deg u_1}, 6]$

Path corresponding to edges that are incident on U_2

Then go to S_2 and follow the nodes

$[U_2, U_2', 1]$ - - - - $[U_2, U_2', 6]$

$[U_2, U_2^2, 1]$ - - - - $[U_2, U_2^2, 6]$

$\vdots$

$[U_2, U_2^{\deg U_2}, 1]$ - - - $[U_2, U_2^{\deg U_2}, 6]$

Then go to S_3 - - - -

$\vdots$

$\vdots$

$[U_k, U_k', 1]$ - - - $[U_k, U_k', 6]$

$[U_k, U_k^2, 1]$ - - - $[U_k, U_k^2, 6]$

$\vdots$

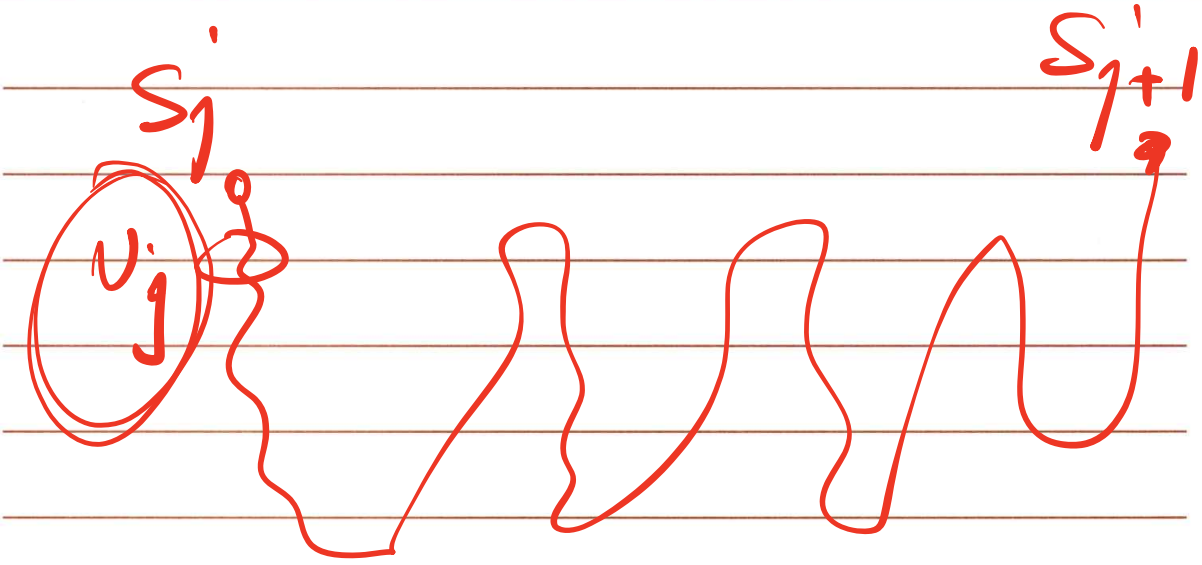
$[U_k, U_k^{\deg U_k}, 1]$ - - - $[U_k, U_k^{\deg U_k}, 6]$

Then return back to S_1.

B) Suppose G' has a Hamiltonian cycle C, then the set

$$S = \{ u_j \in V : (s_j, [u_j, u_j', 1]) \in C \text{ for some } 1 \leq j \leq k \}$$

will be a vertex cover set in G.



We Prove that TSP is NP-Complete

1. Show that TSP $\in$ NP

Certificate: Tour of cost $\leq C$

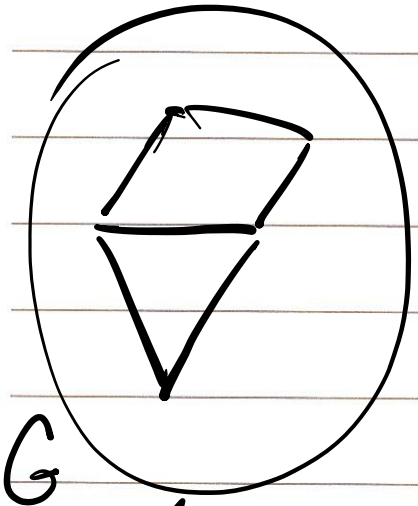
Certifier: Everything we did for HC certifier +

check total cost of the tour $\leq C$ ✓

2- Choose an NP-Complete problem:

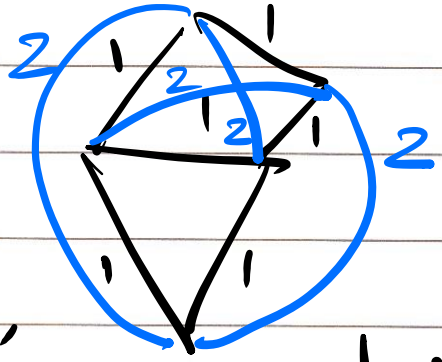
Hamiltonian Cycle.

3- Prove that Ham. Cycle $\leq_p$ TSP



G

no. of nodes in G



G'

Is there a tour of cost $\leq n$ in G'?

Proof:

a) If we have a HC in G

$\Rightarrow$ I can create a tour of cost n in G'

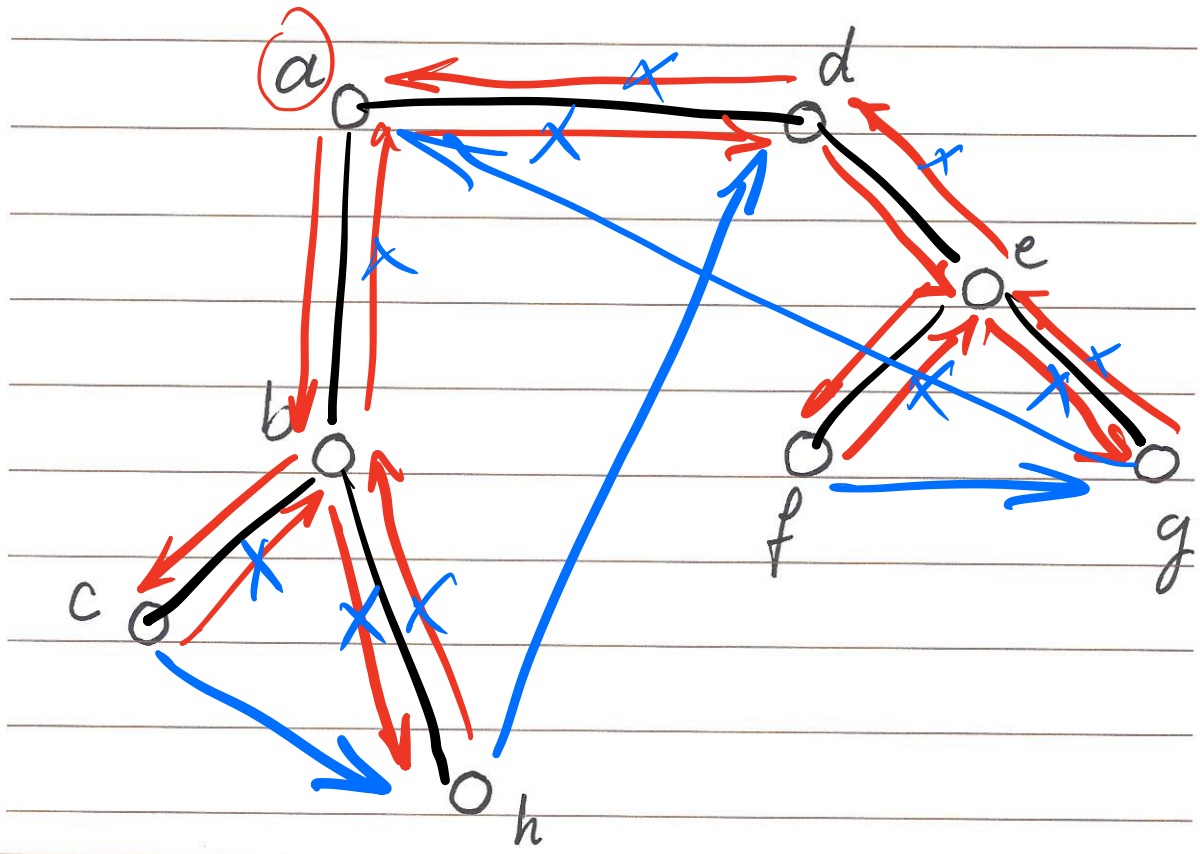
b) If we have a tour of cost n in G'

$\Rightarrow$

Our Collection of NP-complete problems:

3SAT, Ind. Sep. Set, Vertex Cover, Set Cover, Ham. Cycle, TSP, 0-1 Knapsack, Subset Sum, Graph 3-Coloring

Traveling Salesman Problem
(w/ Triangle Inequalities)



Cost of initial tour = 2 * Cost of MST

(Cost of MST $\leq$ Cost of opt tour

Cost of our approx. sol. $\leq$ 2 * Cost of MST

Cost of our sol. $\leq$ 2 * Cost of opt. sol.

This is a 2-approximation alg.

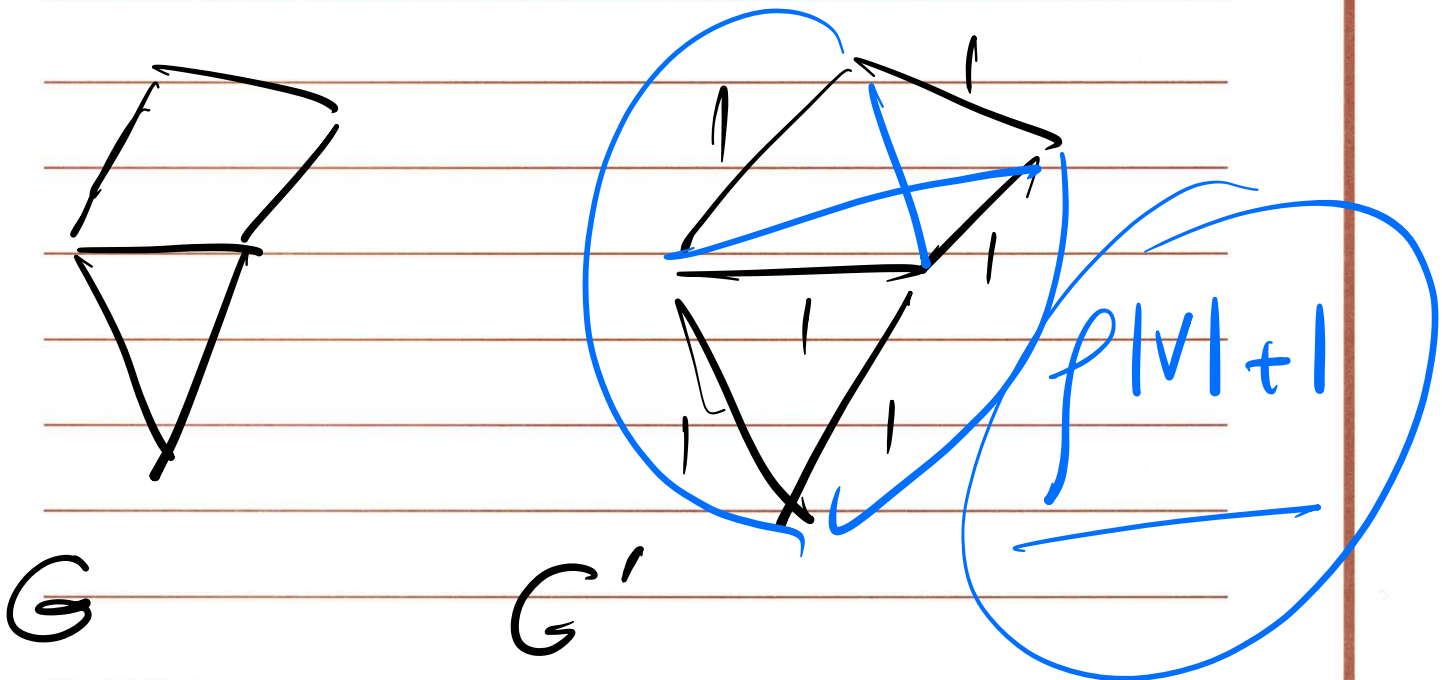
General TSP

Theorem: if $P \neq NP$, then for any constant $\rho \geq 1$, there is no polynomial time approximation algorithm with approximation ratio ρ for the general TSP

Plan: We will assume that such an approximation algorithm exists. We will then use it to solve the HC problem.

Given an instance of the HC problem on graph G, we will construct G' as follows:

- G' has the same set nodes as in G
- G' is a fully connected graph.
- Edges in G' that are also in G have a cost of 1.
- Other edges in G' have a cost of $f|V|+1$.



a) If there is a HC in G

$\Rightarrow$ cost of opt. tour in $G' = |V|$

approx. alg. should find a tour of cost $\leq \rho(|V|)$

b) if there is a tour of cost $\leq \rho|V|$ in G'

$\Rightarrow$ G has a HC!

Discussion 11

1. In the *Min-Cost Fast Path* problem, we are given a directed graph $G=(V,E)$ along with positive integer times t_e and positive costs c_e on each edge. The goal is to determine if there is a path P from s to t such that the total time on the path is at most T and the total cost is at most C (both T and C are parameters to the problem). Prove that this problem is **NP**-complete.

2. We saw in lecture that finding a Hamiltonian Cycle in a graph is **NP**-complete. Show that finding a Hamiltonian Path -- a path that visits each vertex exactly once, and isn't required to return to its starting point -- is also **NP**-complete.

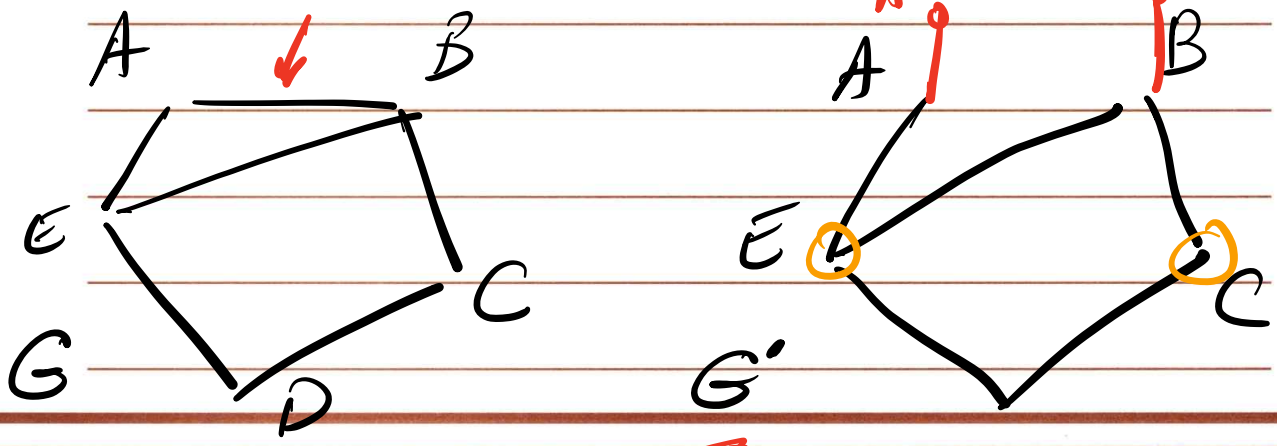
3. Some **NP**-complete problems are polynomial-time solvable on special types of graphs, such as bipartite graphs. Others are still **NP**-complete.
Show that the problem of finding a Hamiltonian Cycle in a bipartite graph is still **NP**-complete.

2. We saw in lecture that finding a Hamiltonian Cycle in a graph is **NP**-complete. Show that finding a Hamiltonian Path -- a path that visits each vertex exactly once, and isn't required to return to its starting point -- is also **NP**-complete.

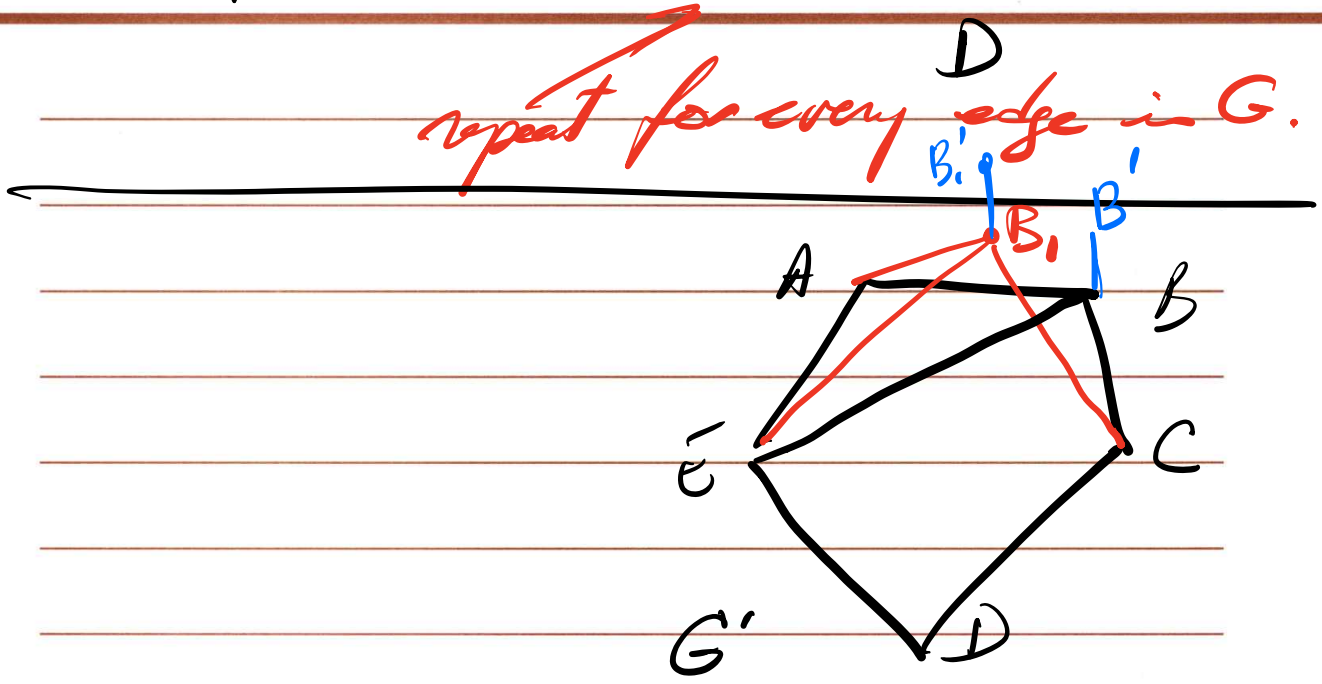
1- Skip

2- Choose Ham. Cycle.

3- Show HC $\leq_p$ Ham. Path



repeat for every edge in G.



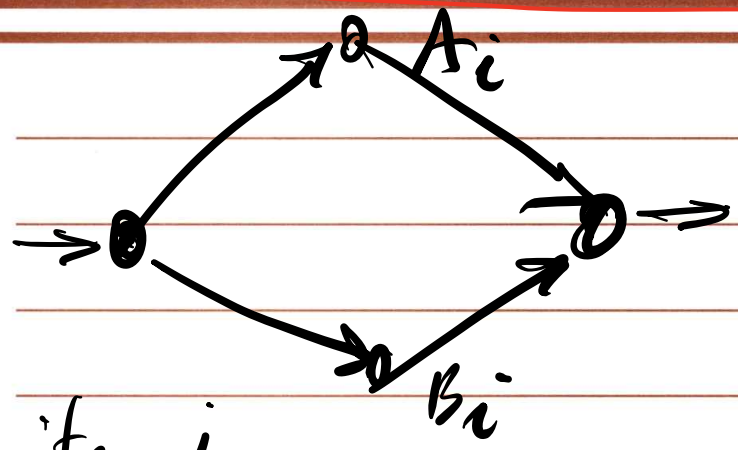
1. In the *Min-Cost Fast Path* problem, we are given a directed graph $G=(V,E)$ along with positive integer times t_e and positive costs c_e on each edge. The goal is to determine if there is a path P from s to t such that the total time on the path is at most T and the total cost is at most C (both T and C are parameters to the problem). Prove that this problem is **NP**-complete.

1- Skip

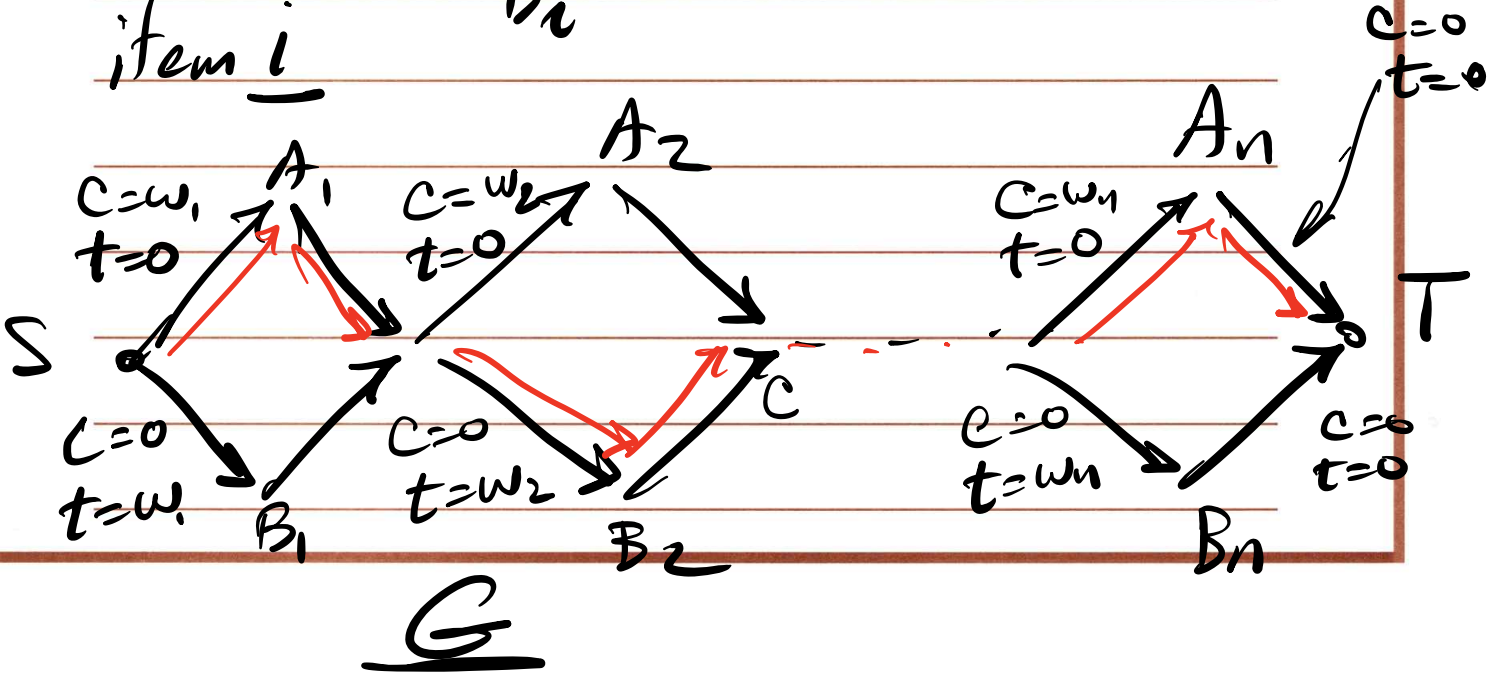
2- Choose Subset sum

3- Show Subset sum $\leq_p$ MCFP

Given a set of n items with weights w_i is there a subset of them with total weight $\leq W$ & total $v \geq M$

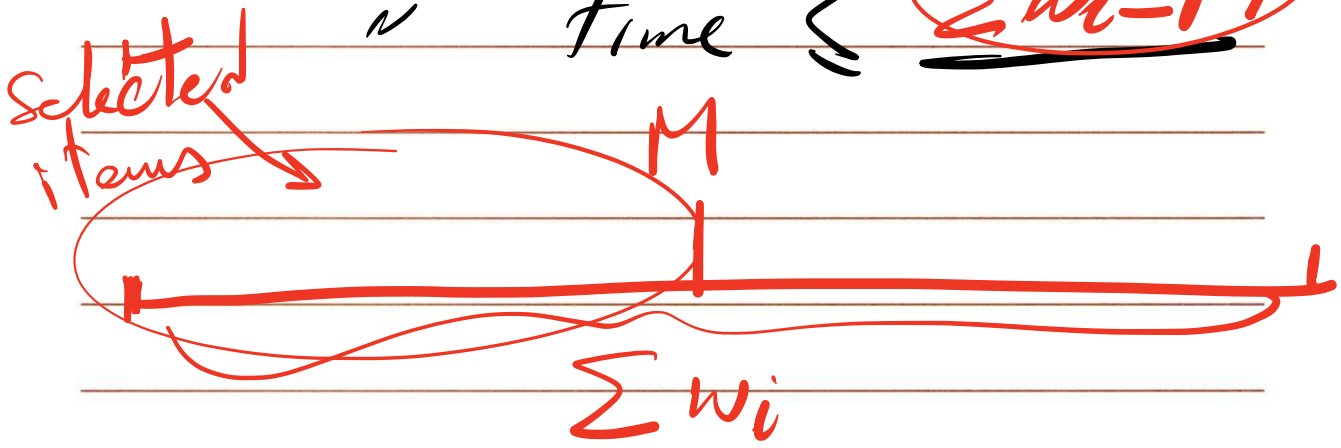


item i



G

Is there a MCFP from s to t
w/ total cost $\leq W$
" time $\leq \sum w_i - M$

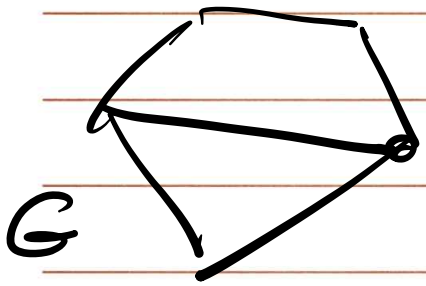


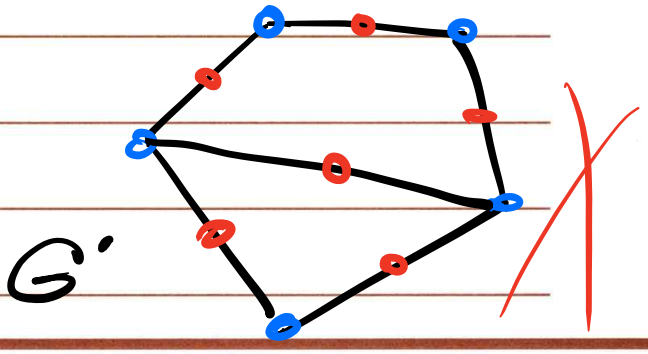
3. Some **NP**-complete problems are polynomial-time solvable on special types of graphs, such as bipartite graphs. Others are still **NP**-complete.
Show that the problem of finding a Hamiltonian Cycle in a bipartite graph is still **NP**-complete.

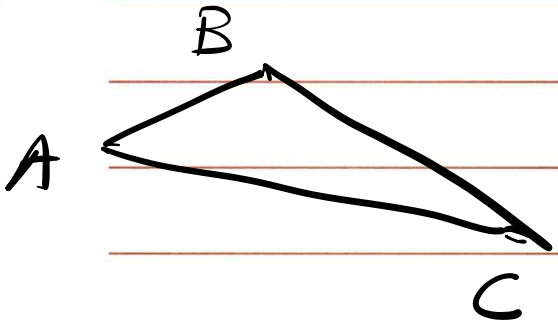
1- Skip

2- Choose HC

3- Show $HC \leq_p HC$ in a bipartite graph

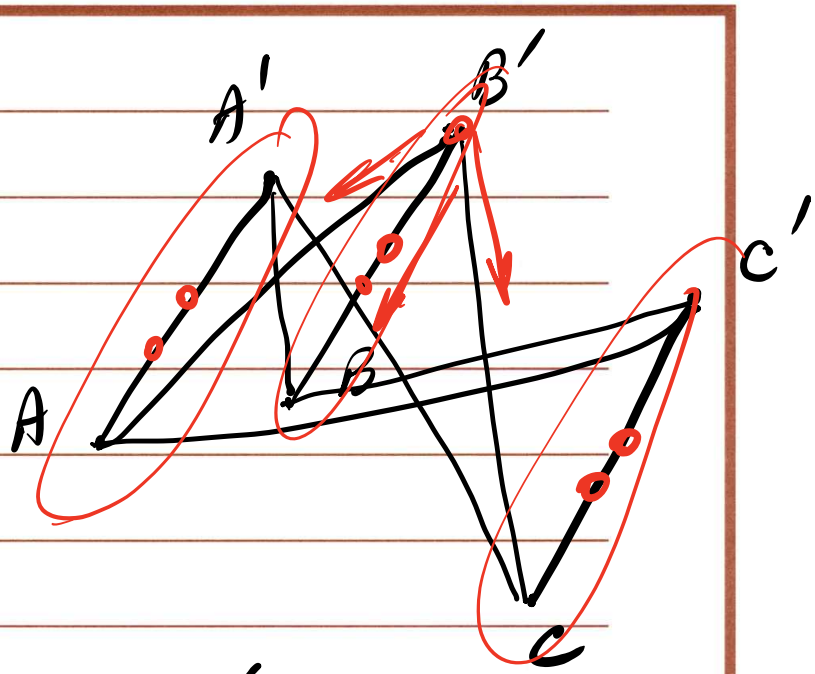






G

ABCA



G'

AA'BB'CC'A

CC'AA'BB'C

C A B C

Q

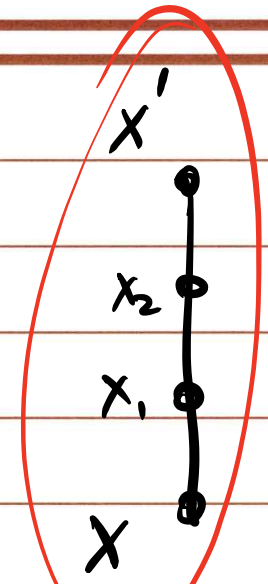
$AB'CA'BC'A$

$AA'BB'CC'A$

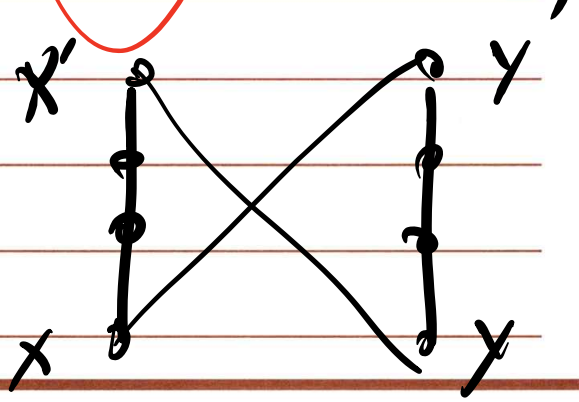
A B C

B'

A node in G (X) →



An edge in G (XY) →



Load Balancing Problem

(Approximation Example)

Input:

m resources with equal processing power

n jobs, where job j takes t_j to process

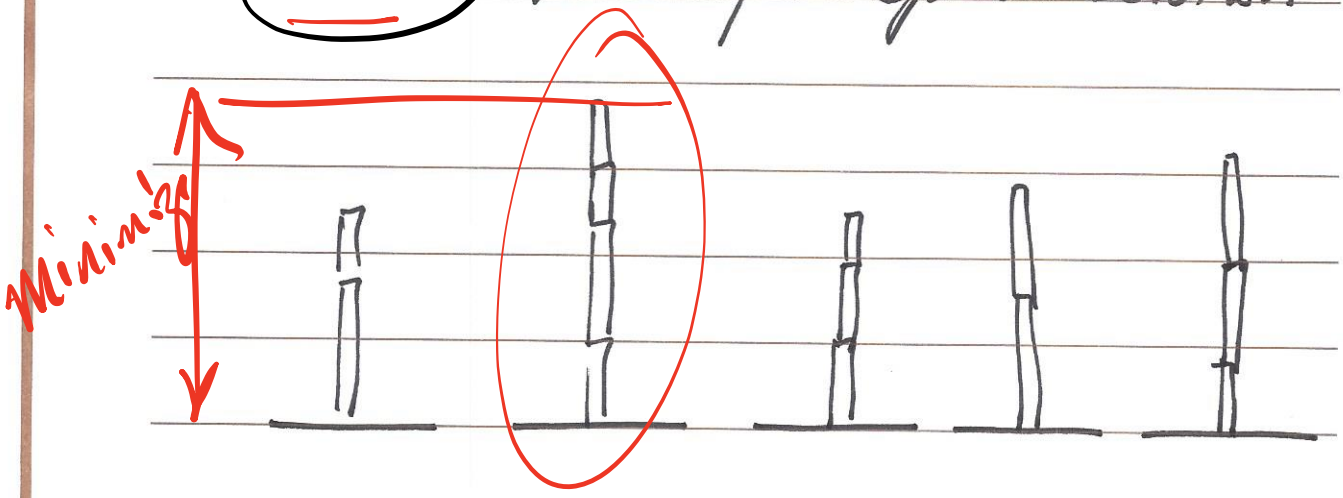
objective:

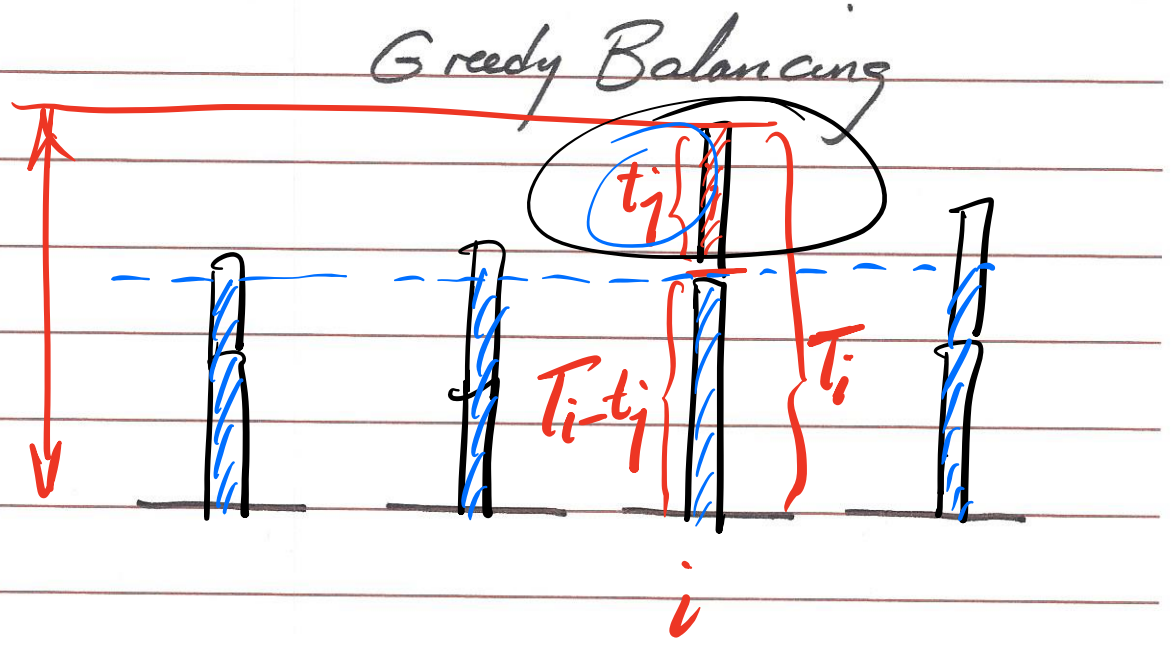
Assignment of jobs to resources such that the maximum load on any machine is minimized.

Notation:

T_i : load on machine/resource i

T^* : value of the optimal solution





$$T^* \geq t_j$$

$$T^* \geq T_i - t_j$$

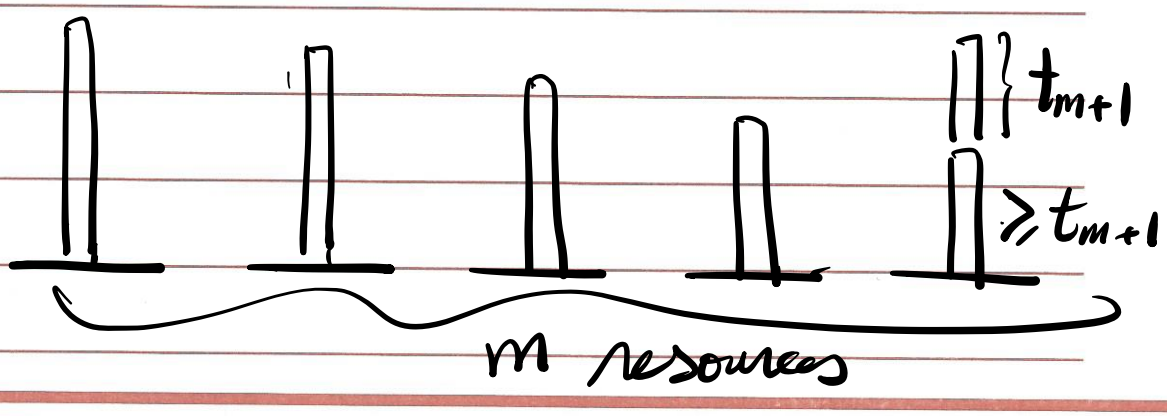
$$2T^* \geq T_i$$

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

We have a 2-approximation

Improved approximation to greedy balancing

Initially sort jobs in decreasing order of length then use same greedy balancing



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

$t_j \leq t_{m+1} \rightarrow t_j \leq \frac{1}{2} T^*$

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

$T_i - t_j \leq T^*$

$T_i \leq 1.5\, T^*$

1.5-approximation!

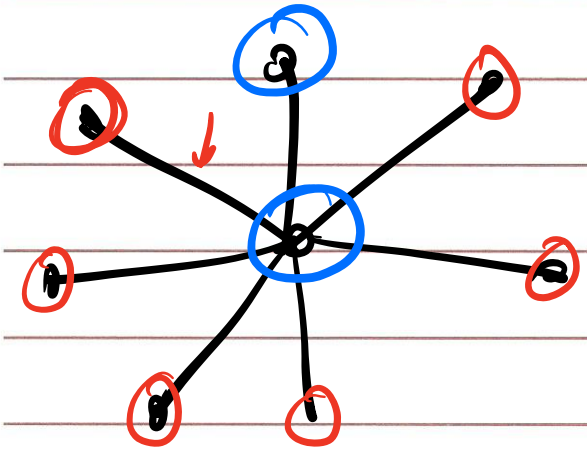
Vertex Cover Problem
(Approximation Example)

Problem Statement: Find the smallest vertex cover in graph G.

A 2-approximation algorithm to the vertex Cover problem:

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

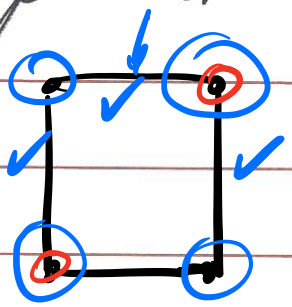
Why is this a 2-approximation?



Question: Since Independent Set $\leq_p$ Vertex Cover,

Can I use our 2-approximation algorithm for vertex cover to find a .5-approximation to independent set?

NO!



Theorem: Unless P=NP, there is no $\frac{1}{n^{1-\epsilon}}$ approximation for the Maximum Independent Set problem for any $\epsilon > 0$ where n is the no. of nodes in the graph.

Question: Since Vertex Cover $\leq_p$ Set Cover

Can I use a 2-approximation algorithm for Set Cover to find a 2-approximation to Vertex Cover? Yes!

MAX-3SAT

Given a set of clauses of length 3, find a truth assignment that satisfies the largest number of clauses.

A .5-approximation to the MAX-3SAT problem:

- Set everything to true
  If less than 50% of clauses evaluate to true, then
- Set everything to False

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

System of Linear Equations

$$[A][X] = [B]$$

Linear Programming

$$[A][X] \leq [B]$$

Objective function: $[C^T][X]$

Goal: Maximize the objective function subject to the above constraints

Linear Programming Standard Form

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

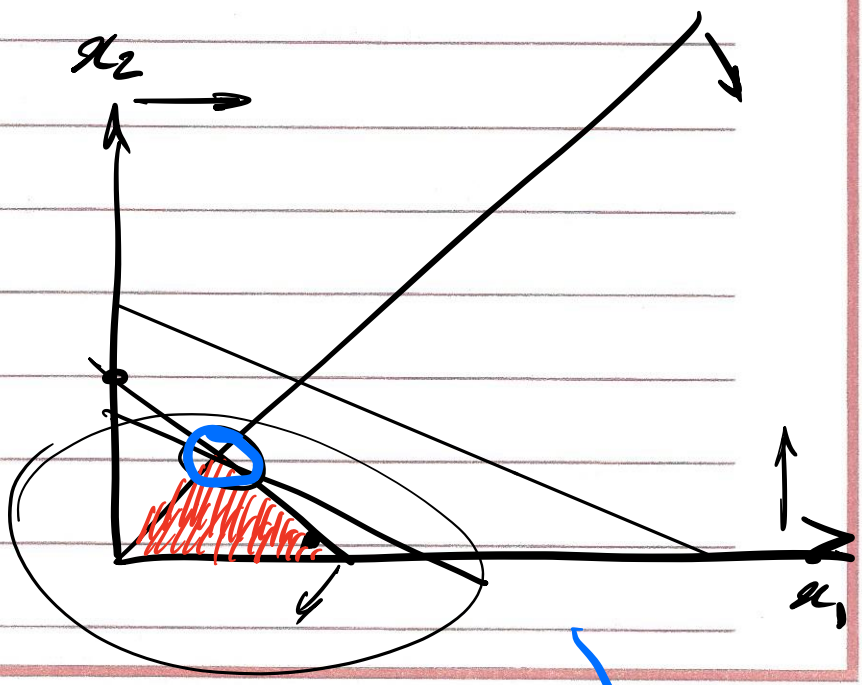
Ex.: $x_1 - x_2 \geq 0$

$x_1 \geq 0$

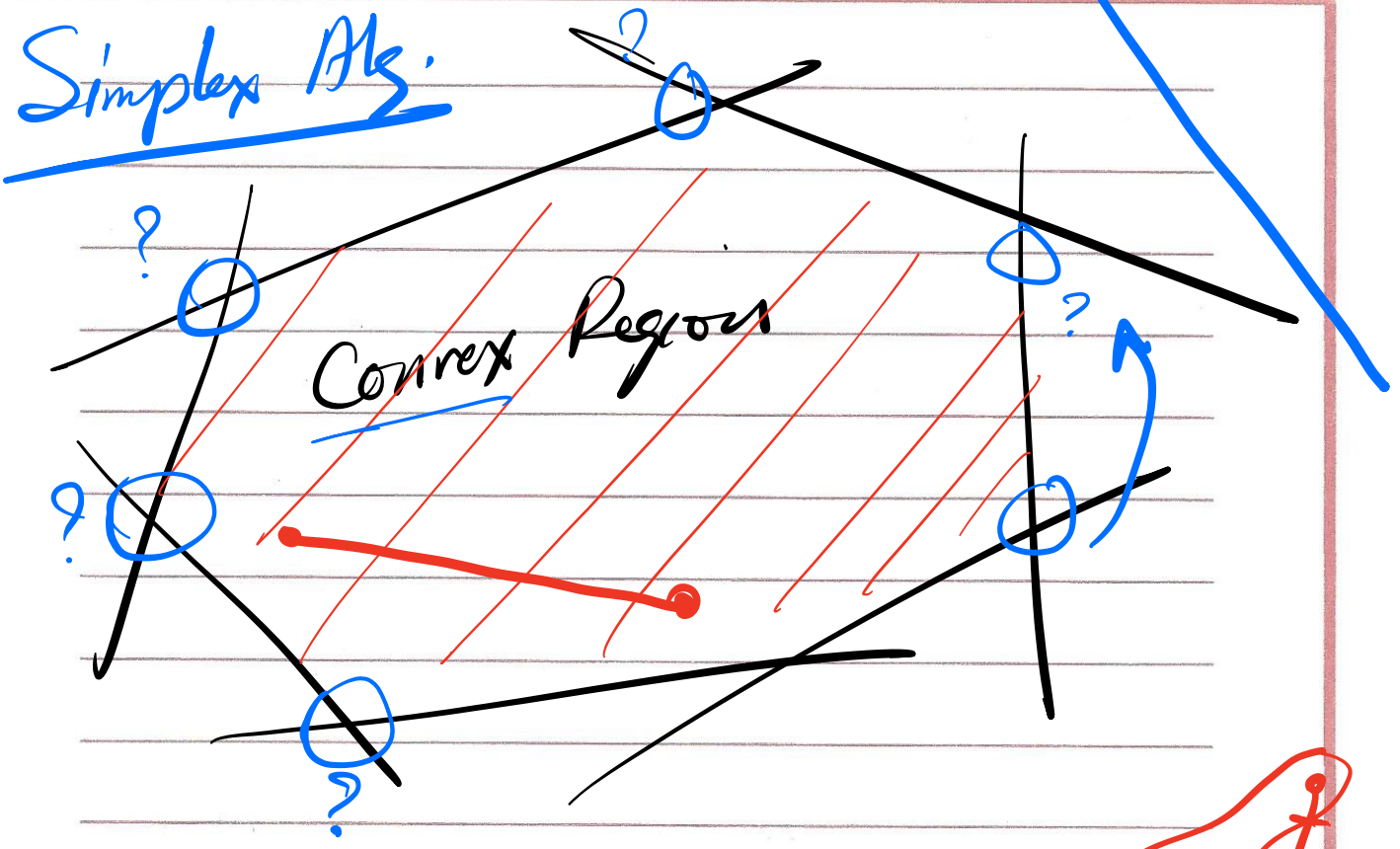
$x_2 \geq 0$

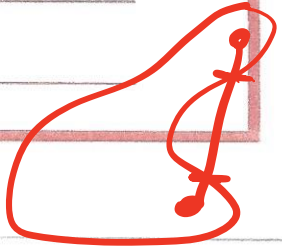
$x_1 + x_2 \leq 4$

Maximize $x_1 + 2x_2$



Simplex Alg.





Weighted Vertex Cover Problem

For $G=(V,E)$, $S \subseteq V$ is a vertex cover set such that each edge has at least one end in S

Also, $w_i \geq 0$ for each $i \in V$

So, the total weight of the set $= W(S) = \sum_{i \in S} w_i$

Objective: Minimize $W(S)$

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

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

$x_i + x_j \geq 1$

x_i x_j

Minimize $\sum w_i x_i$

Subject to:

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

$x_i \in \{0,1\}$? ✗

This is Not a linear program

This is integer linear programming

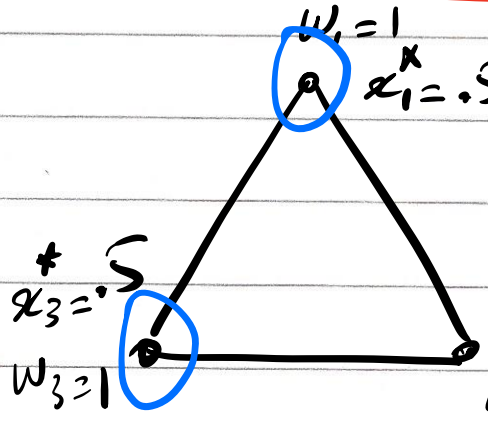
- Linear Programming
  Continuous Variables
- Integer Programming
  Discrete variables
- Mixed Integer Programming
  Both Cont. & Discrete
- Non-Linear Programming
  Non Linear Constraints or Objective Functions

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

$$w_{LP} = \sum_i w_i x_i^*$$

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

$$w_{LP} \leq w(S')$$



$w_{LP} = 1 \times .5 + 1 \times .5 + 1 \times .5$
$= 1.5$

$w(S') = 2$

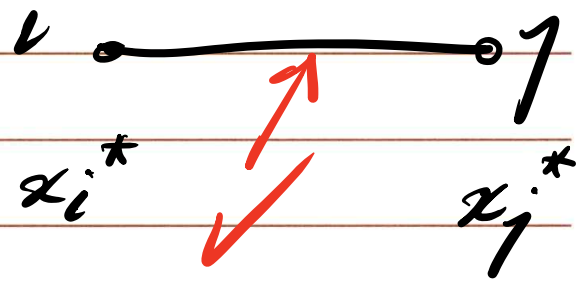
$w(S) = 1 \times 1 + 1 \times 1 + 1 \times 1 = 3$

$x_i^* = 0 \Rightarrow i \notin S$

$x_i^* = 1 \Rightarrow i \in S$

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

$x_i^* + x_j^* \geq 1$



Say S is our approx. sol.

$w(S) \leq 2 \cdot w_{LP}$

$w_{LP} \leq w(S')$

$w(S) \leq 2 \cdot w(S')$

$\Rightarrow$ 2-approximation

Maxflow Problem

Variables are flows over edges

objective function: Maximize $\sum_{e \text{ out of } S} f(e)$

Subject to

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

$\sum_{e \text{ into } v} f(e) - \sum_{e \text{ out of } v} f(e) = 0$ for $v \in V$ except for s & t

$A = B$

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

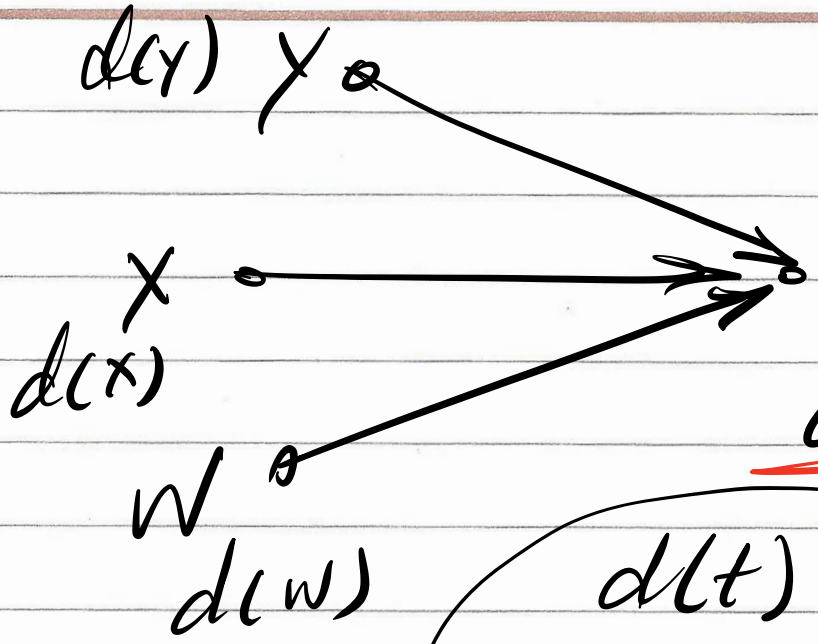
$\sum_{e \text{ into } v} f(e) \geq ?$

$\leq ?$

Shortest Path using LP

Problem Statement:
Find shortest path from s to t.

Shortest distance from s to v is d(v) for each v.



$d(t) \leq d(y) + C_{yt}$

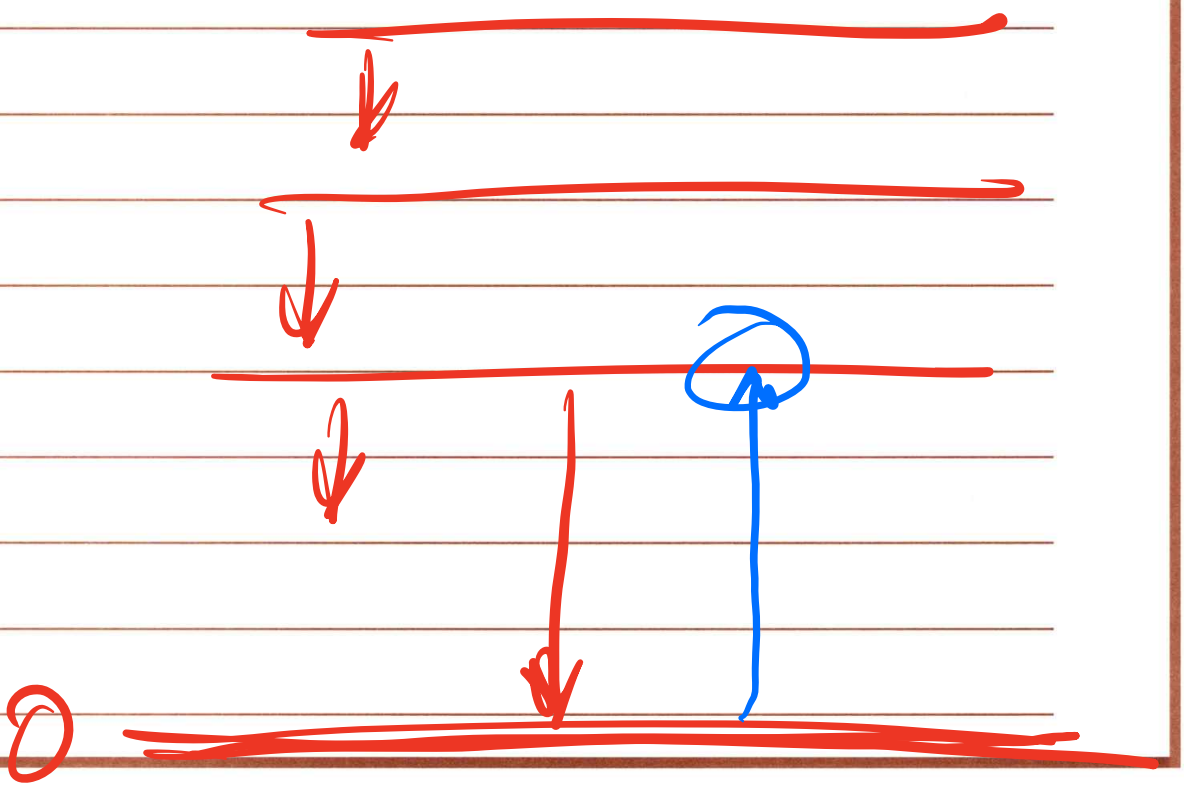
$d(t) \leq d(x) + C_{xt}$

$d(t) \leq d(w) + C_{wt}$

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

Objective function:

Maximize ~~Minimize~~ $d(t)$



Discussion 12

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

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

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

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

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

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

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

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

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

4. Recall the 0/1 knapsack problem:

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

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

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

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

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

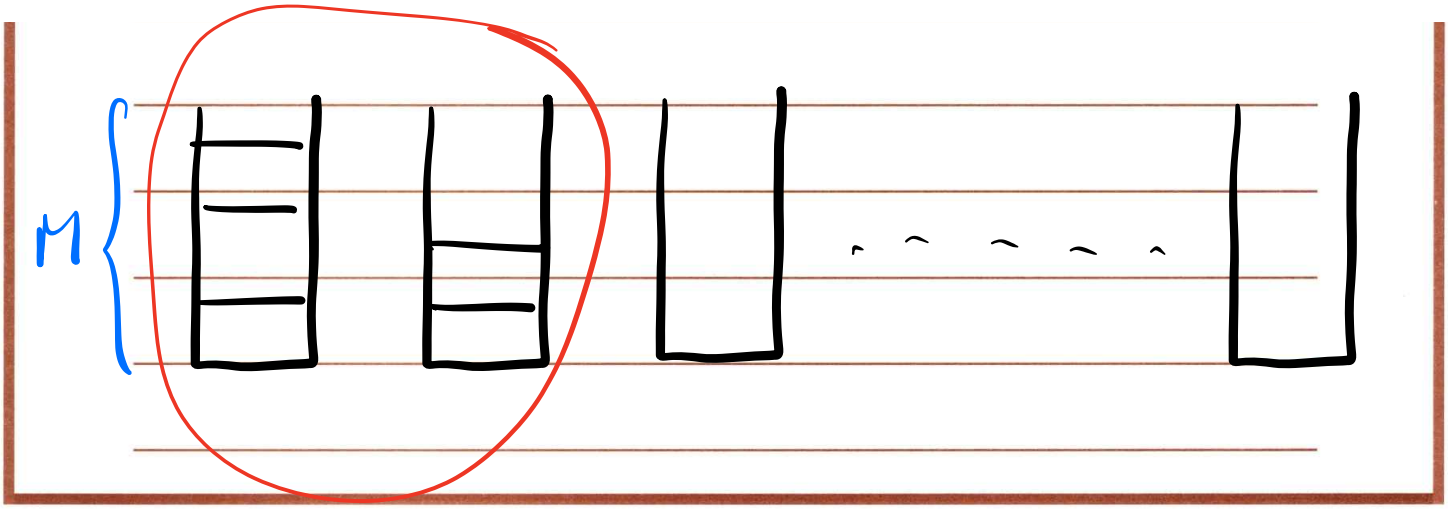
While we can add more items to the knapsack, pick items in this order.

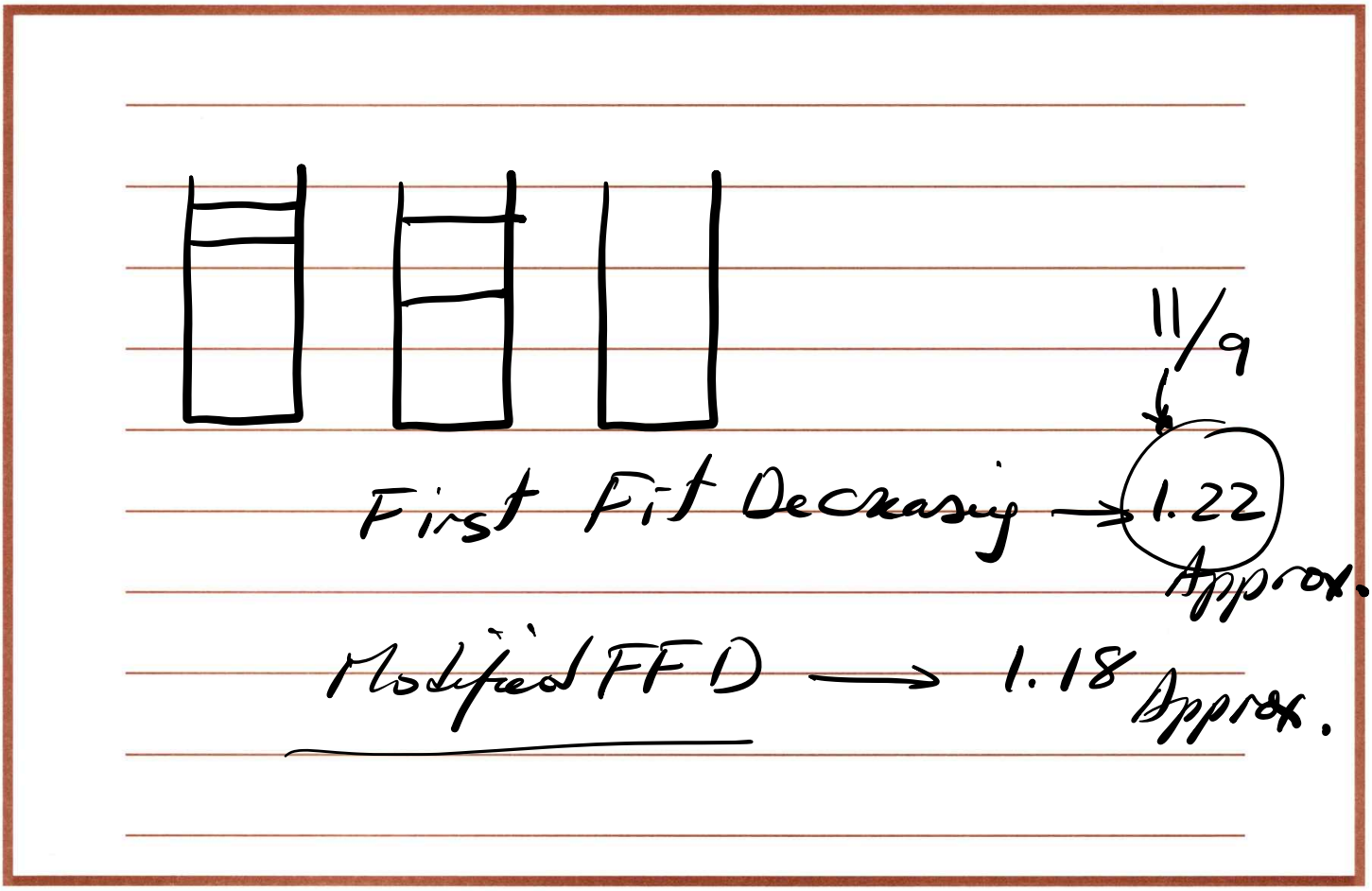
Show that this approximation algorithm has no nonzero constant approximation ratio.

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

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

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



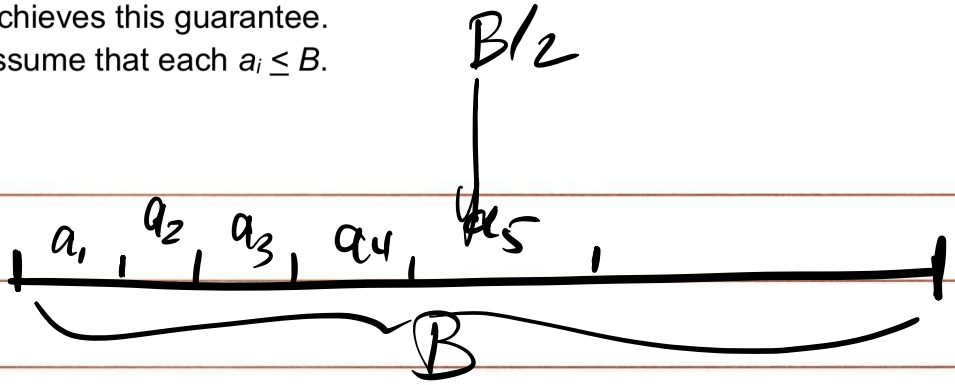


2. Suppose you are given a set of positive integers A: $a_1\ a_2\ \ldots\ a_n$ and a positive integer B. A subset $S \subseteq A$ is called *feasible* if the sum of the numbers in S does not exceed B.

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

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

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



Either choose $a_1 \ldots a_k$ if $\sum_{i}^{k} a_i \geq B/2$

or if a_k does not fit into the remaining cap. and we have not reached $B/2$ then use a_k itself.

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

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

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

Let M_1, M_2, M_3 denote the cubic meters of mat's 1, 2, 3.

Maximize $1000 * M_1 + 1200 M_2 + 12000 M_3$

Subject to:

$$M_1 + M_2 + M_3 \leq 60$$

$$2 * M_1 + M_2 + 3 M_3 \leq 100$$

$$0 \leq M_1 \leq 40$$

$$0 \leq M_2 \leq 30$$

$$0 \leq M_3 \leq 20$$

4. Recall the 0/1 knapsack problem:

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

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

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

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

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

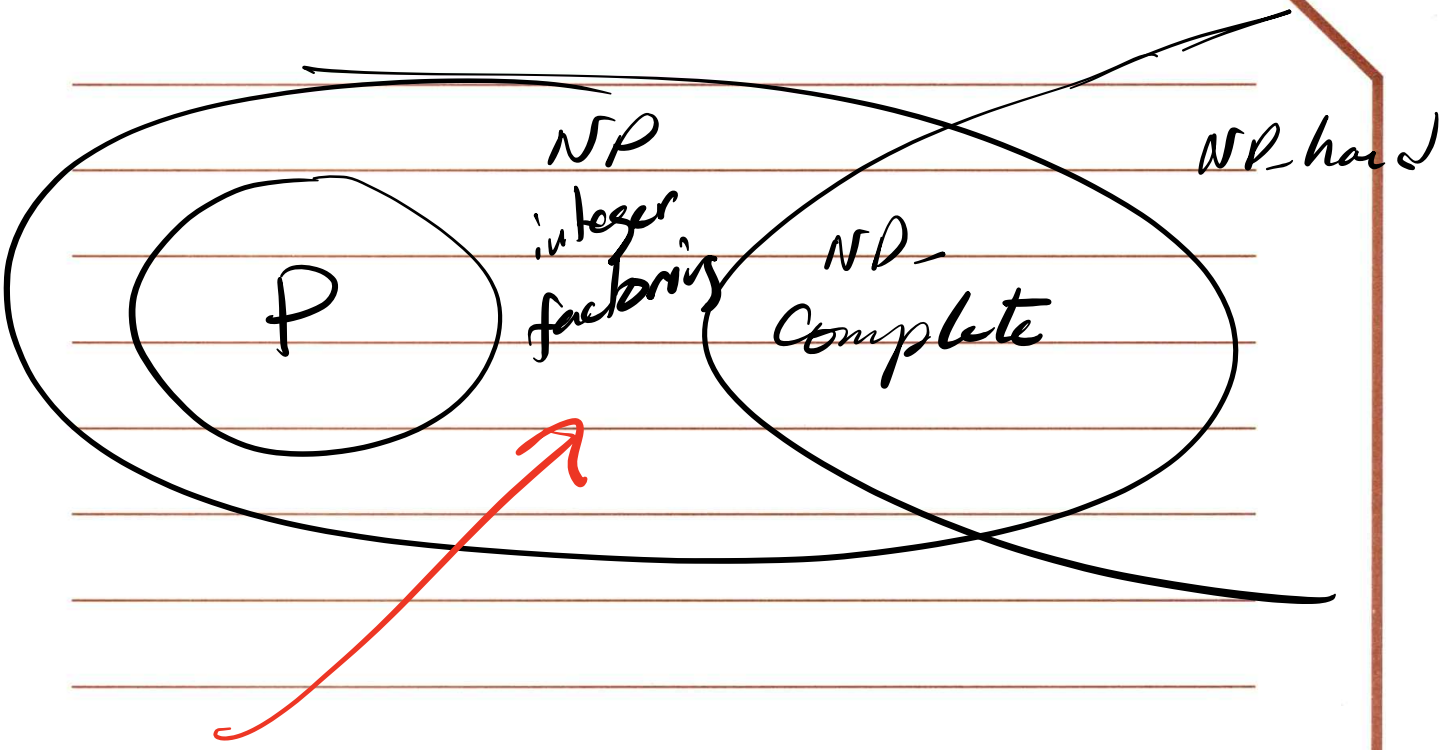
While we can add more items to the knapsack, pick items in this order.

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

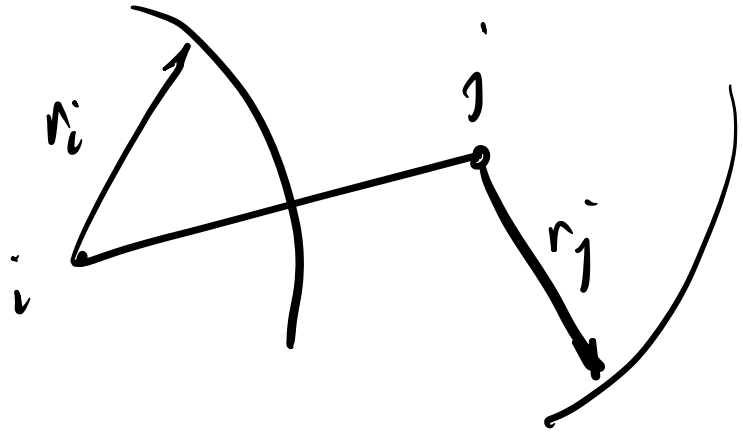
In other words, if the value of the optimal solution is P*, prove that there is no constant ρ ($1 > \rho > 0$), where we can guarantee our greedy algorithm to achieve an approximate solution with total profit ρ P*.

item 1: weight 1. value 2

item 2: weight W value W


NP
integer factoring
NP-hard
P
NP-Complete

A set of n space stations need your help in building a radar system to track spaceships traveling between them. The i^{th} space station is located in 3D space at coordinates (x_i, y_i, z_i). The space stations never move. Each space station i will have a radar with power r_i, where r_i is to be determined. You want to figure how powerful to make each space station's radar transmitter, so that whenever any spaceship travels in a straight line from one space station to another, it will always be in radar range of either the first space station (its origin) or the second space station (its destination). A radar with power r is capable of tracking spaceships anywhere in the sphere with radius r centered at itself. Thus, a spaceship is within radar range through its trip from space station i to space station j if every point along the line from (x_i, y_i, z_i) to (x_j, y_j, z_j) falls within either the sphere of radius r_i centered at (x_i, y_i, z_i) or the sphere of radius r_j centered at (x_j, y_j, z_j). The cost of each radar transmitter is proportional to its power, and you want to minimize the total cost of all of the radar transmitters. You are given all of the (x_1, y_1, z_1), ... ,(x_n, y_n, z_n) values, and your job is to choose values for $r_1,...,r_n$. Express this problem as a linear program.



Variables: r_i's

Objective function: Minimize $\sum r_i$'s

Subject to $r_i + r_j \geq d_{ij}$ for all i & j

Randomized Algorithms

Two general types of randomizations

1- Algorithm relies on random nature of input

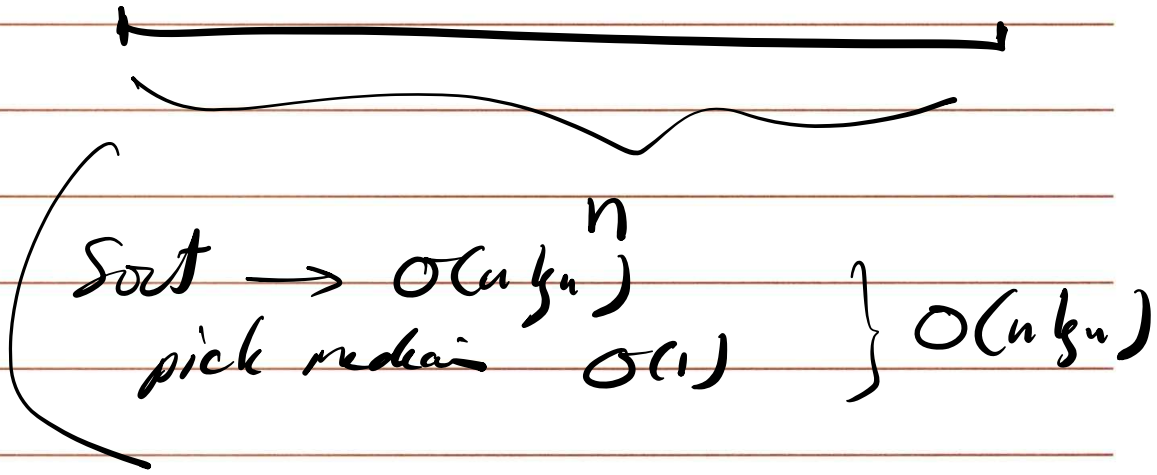
2- Algorithm behaves randomly

Two types of Results

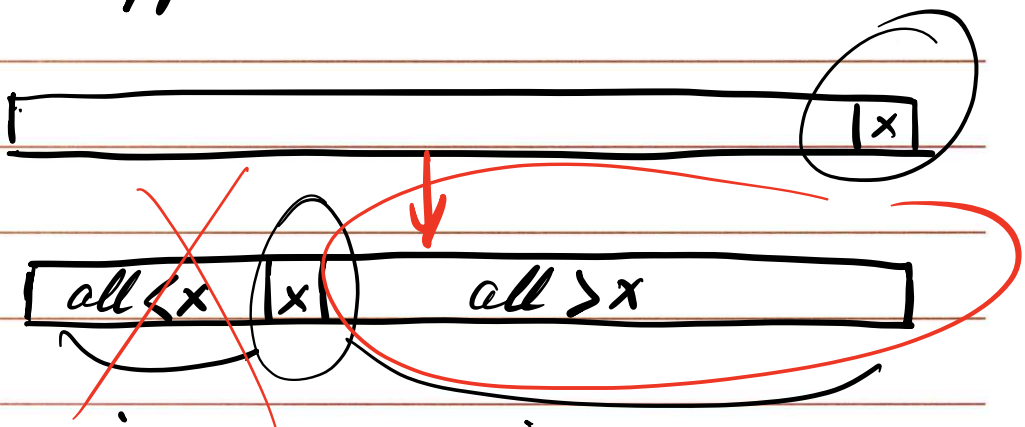
1- Some randomized algorithms always produce correct results. Randomization only affects the run time.

2- Some randomized algorithms may produce incorrect solutions, but we try to bound the probability of such an incorrect solutions.

Ex 1: **Problem Statement:**
Find the median in an unsorted array.



Another Approach



If looking for the median, we then only need to look into either the right half or the left half. (Not necessarily equal halves)

Solution:

Use the random nature of data/input to find the median:

Divide: Pick a pivot value (last term in the array) and split the array in two.
Terms in left half are all less than the pivot value
Terms in right half are all greater than the pivot value.

Conquer:

if x is the median term we are looking for return it
else if the median is in the left half then explore the left half recursively
else explore the right half recursively
end if

A Simplistic Analysis of Run time

Assume array size is only reduced by 10% each time.

$$T(n) = cn + (9/10)cn + (9/10)^2 cn + \cdots$$

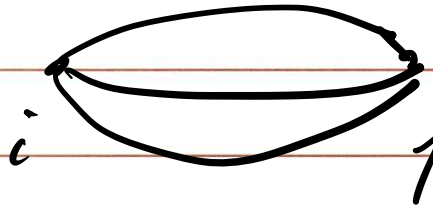
$$= O(n)$$ ← Expected Run Time

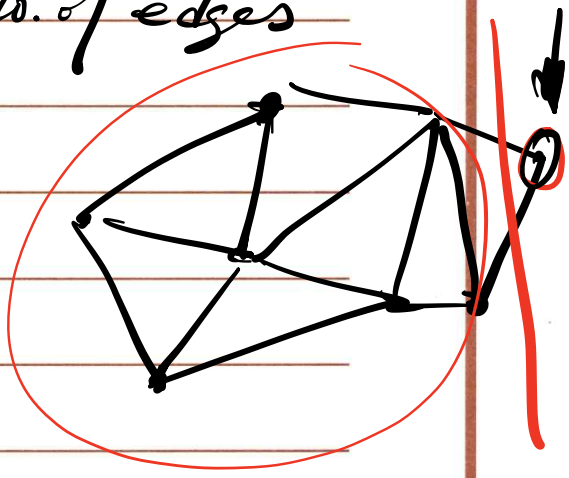
$$x + \alpha x + \alpha^2 x + \cdots$$

$$\alpha < 1 \Rightarrow \sum = O(x)$$

Ex. 2: Global Min-Cut Problem

Given a connected unweighted multigraph, partition vertices into two disjoint sets with the minimum no. of edges connecting the two set.





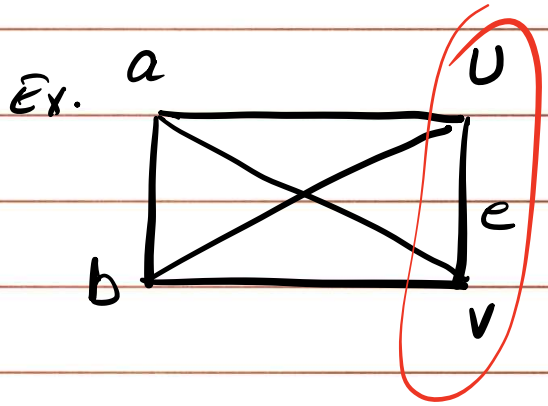
Deterministic solution

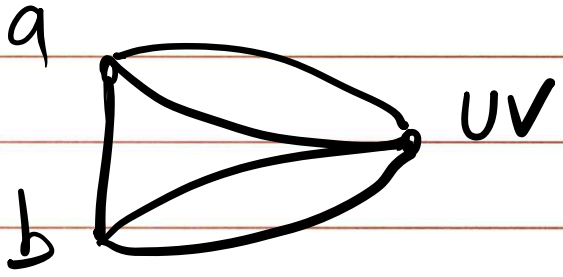
- Pick any vertex as a source
- For each other vertex in V
  Run Max Flow and find min cut.
- Return smallest Min Cut.

Randomize Solutions Using Edge Contractions

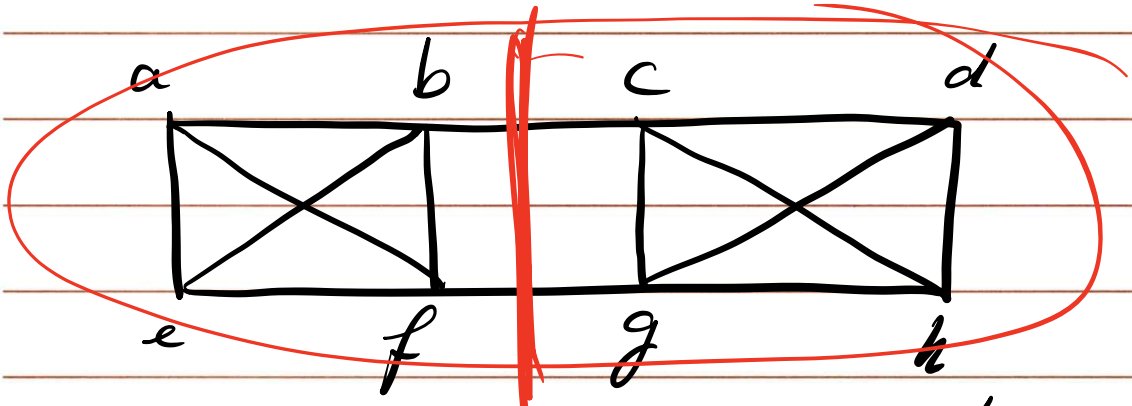
Edge Contraction: Contraction of an edge e with endpoints U & V is the replacement of U and V with a single vertex such that edges incident to the new vertex are the edges (other than e) that were incident on U or V.

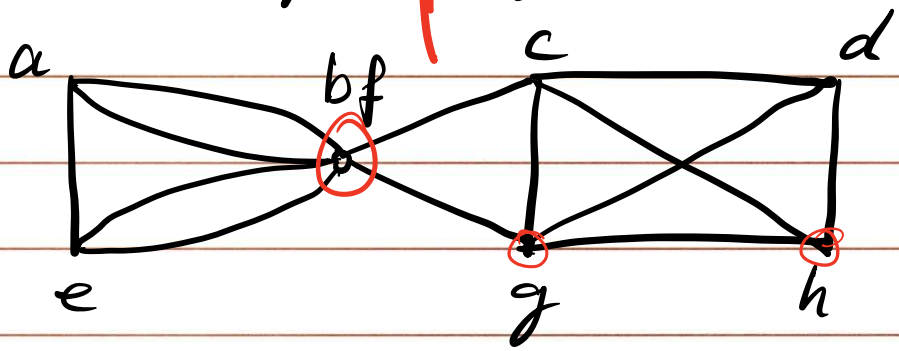
Ex.



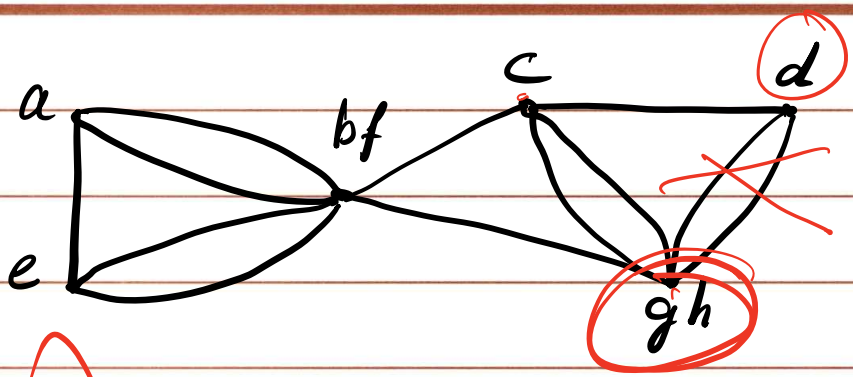


Find Global Min-Cut in the graph below:

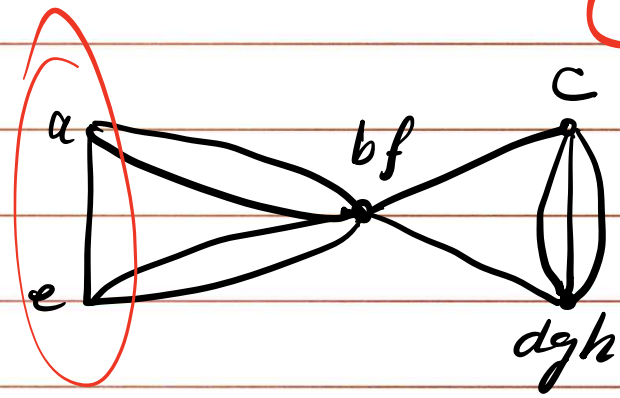




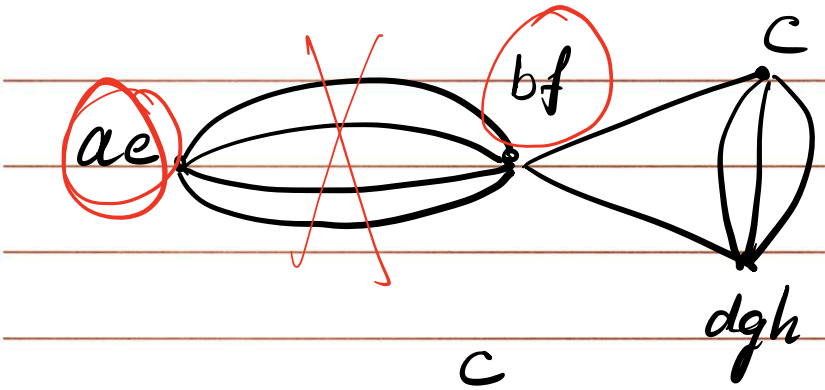
Contract edge bf



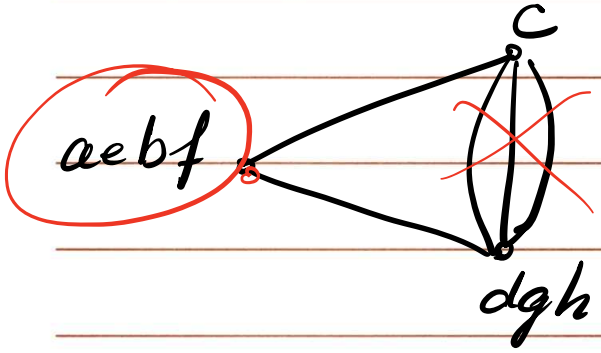
Contract gh



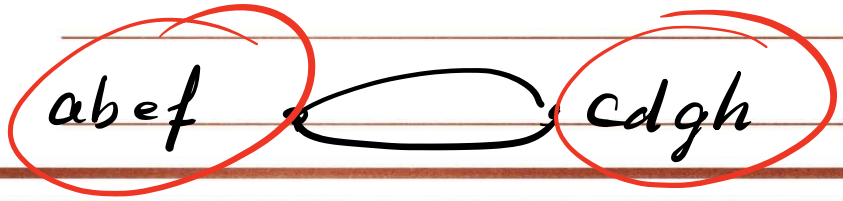
Contract d(gh)



Contract ae



Contract (ae)(bf)



Contract c(dgh)

Randomized Algorithm

While there are more than 2 nodes left
pick an edge at random
and contract the edge
(Remove any self loops)
end while

For the two remaining nodes U_1 & U_2
Set V_1 = {nodes that went into U_1}
V_2 = { " " " " U_2}

Analysis

Assume that a graph has a single min cut C. If we never contract edges from that min cut then the algorithm will end up with the correct Min Cut.

How likely is this?

Say size of min cut is k.
So each node must have at least degree k.

$\Rightarrow$ Total no. of edges in $G \geq kn/2$

So, the probability that the 1st contracted edge is Not on our min cut is at least $1 - \frac{k}{kn/2}$

The probability that all contracted edges are Not on our min cut is at least

$$\left(1-\frac{2}{n}\right)\cdot\left(1-\frac{2}{n-1}\right)\cdot\left(1-\frac{2}{n-2}\right)\cdots\left(1-\frac{2}{3}\right)$$

$$=\Omega\left(1/n^2\right)$$

The probability of the algorithm failing N times is

$$\left(1-1/n^2\right)^N$$

If we choose $N=100n^2$ we can make it close to zero! Since $\left(1-1/n^2\right)^{100n^2}\leq e^{-100}$

$$\left(\lim_{x\to\infty}\left(1-1/x\right)^{ax}=e^{-ax}\right)$$

Discussion 10

1. Given the SAT problem from lecture for a Boolean expression in Conjunctive Normal Form with any number of clauses and any number of literals in each clause. For example,

$$(X_1 \vee \neg X_3) \wedge (X_1 \vee \neg X_2 \vee X_4 \vee X_5) \wedge \ldots$$

Prove that SAT is polynomial time reducible to the 3-SAT problem (in which each clause contains at most 3 literals.)

Solution: We will turn each clause of size k in the SAT problem into one or more clauses of size 3 as follows:

Clause size	SAT	3SAT
1	(X_1)	$(X_1 \vee X_1 \vee X_1)$
2	$(X_1 \vee X_2)$	$(X_1 \vee X2 \vee X_1)$
3	$(X_1 \vee X2 \vee X_3)$	$(X_1 \vee X2 \vee X_3)$
4	$(X_1 \vee X2 \vee X_3 \vee X_4)$	$(X_1 \vee X2 \vee S_1) \wedge (\neg S_1 \vee X_3 \vee X_4)$
5	$(X_1 \vee X2 \vee X_3 \vee X_4 \vee X_5)$	$(X_1 \vee X2 \vee S_1) \wedge (\neg S_1 \vee X_3 \vee S_2) \wedge (\neg S_2 \vee X_4 \vee X_5)$
…		

In general, we can turn any clause of size k>3 into k-2 clauses of size 3 using k-3 dummy variables to "chain" the clauses together using conjunction as shown in above examples.

Proof: We need to show that the clause of size k is satisfiable iff the chain of clauses of size 3 is satisfiable. To show this:
A – if we have a satisfying truth assignment in the clause of size k, we can find a satisfying truth assignment in the chain of clauses of size 3, because the satisfying truth assignment in the clause of size k requires at least one of the terms in the clause to be true. This will cause one of the (k-2) clauses of size 3 in the chain to evaluate to 1, we can then use the k-3 dummy variables to set the remaining clauses to be true and therefore find a satisfying truth assignment for the chain.

B – if we have a satisfying truth assignment in the chain of clauses of size 3, it must be that at least one of the X_i variables is true (because the dummy variables on their own can only set k-3 clauses to true). And a satisfying truth assignment in the clause of size k requires only one of the X_i variable to be true. So this will be a satisfying truth assignment.

2. The *Set Packing* problem is as follows. We are given *m* sets $S_1, S_2, \ldots S_m$ and an integer *k*. Our goal is to select *k* of the *m* sets such that no selected pair have any elements in common. Prove that this problem is **NP**-complete.

Solution:

1- Prove that Set Packing is in NP
   Certificate: a subset of k sets (out of the m sets given) that have no elements in common
   Certifier: Can easily check in polynomial time that
   a- There are k sets in the certificate
   b- The k sets have no elements in common
   A and b can be easily done in polynomial time. → Set Packing ϵ NP
2- Choose independent set for our reduction
3- Will show that Indep. Set $\leq_p$ Set Packing
   We will start with an instance of the Indep Set problem (Is there an indep set of size at least k in G) and will construct a set of sets such that there are k of them that have no elements in common iff we have an independent set of size k in G.

Construction of sets: For each node i in G we will create a set S_i. The elements of S_i will consist of the edges incident on i in G.

Proof of correctness for the reduction step:
A – If we have an indep set of size k in G, we can use that to find k sets that have no common elements. The reason is that since the k nodes in G are independent they do not share any edges, therefore the sets corresponding to these k nodes will not have any elements in common since these elements correspond to the edges incident on the k nodes in G.

B – If we have k sets that have no elements in common, we can find an indep set of size k in G. The reason is that since these sets do not have any elements in common, the corresponding nodes in G will have no edges in common or in other words they will be independent, and will form an indep set of size k.

3. The *Steiner Tree* problem is as follows. Given an undirected graph *G=(V,E)* with nonnegative edge costs and whose vertices are partitioned into two sets, *R* and *S*, find a tree $T \subseteq G$ such that for every *v* in *R*, *v* is in *T* with total cost at most *C*. That is, the tree that contains every vertex in *R* (and possibly some in *S*) with a total edge cost of at most *C*.
Prove that this problem is **NP**-complete.

Solution:

1- Prove that the Steiner Tree Problem is in NP
   Certificate: a tree of cost at most C that covers all nodes in R
   Certifier: Can easily check in polynomial time that

a- Tree covers on nodes in R (run BFS on the tree)
b- The total cost of the tree is at most C

A and b can be easily done in polynomial time. → Steiner Tree Problem ϵ NP

2- Choose vertex cover for our reduction
3- Will show that Vertex Cover $\leq_p$ Steiner Tree Problem
We will start with an instance of the vertex cover problem (Is there an vertex cover of size at most k in G) and will construct G' such that G has a vertex cover of size at most k iff G' has a Steiner Tree of cost at most m+k.

Construction of G': G' will have the same set of nodes and edges in G plus a number of new nodes and edges. The nodes in G' that exist in G will belong to the set S. We now introduce a new set of nodes in G':

- We will add one node (r_e) per edge e in G' (adding m nodes in this process). All these nodes will belong to the set R
- We will connect each node to the two ends of the corresponding edge in G'
- We will add one more node r_0 in G' and will connect r_0 to all the nodes in the set S in G'
- All edges in G' will have a cost of 1

Proof of correctness for the reduction step:

A- If we have a vertex cover of size k in G, we can produce a Steiner tree of cost m+k in G'. This can be done by using the following edges in the Steiner Tree
   a. k edges that connect r_0 to the node in G' corresponding to those k nodes that form a vertex cover of size k in G.
   b. m edges that connect each of the r_e nodes in G' to the node that covers in e in G.

   This will result in a tree of cost m+k that covers all nodes in the set R.

B- If we have a Steiner tree of cost m+k in G' we can produce a vertex cover of size k in G. This can be done by putting all the nodes in the set S that are part of the Steiner tree in the vertex cover set. (m of the edges will be needed to connect r_e nodes to one of the S nodes. The other k edges in the tree will be connecting these S nodes (that are part of the tree) to other S nodes on the tree (for example through node r_0). Since these S nodes have direct connections to all nodes in the set R, then the nodes corresponding to these S nodes in G will form a vertex cover of size k.)

Discussion 11

1. In the *Min-Cost Fast Path* problem, we are given a directed graph *G=(V,E)* along with positive integer times t_e and positive costs c_e on each edge. The goal is to determine if there is a path *P* from *s* to *t* such that the total time on the path is at most *T* and the total cost is at most *C* (both *T* and *C* are parameters to the problem). Prove that this problem is **NP**-complete.

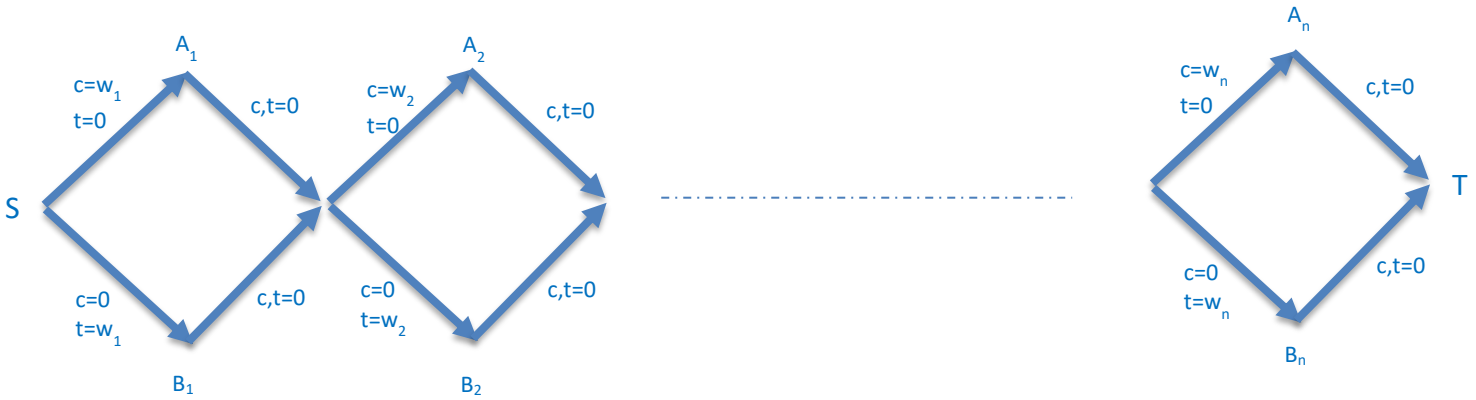
Solution:

1- Prove that Min-Cost Fast Path is in NP
   Certificate: an s-t path with total cost ≤C and total time ≤T
   Certifier: Can easily check in polynomial time that
   a- Set of edges given are in fact a path from s-t
   b- Total time is ≤T and total cost is ≤C
   a and b can be easily done in polynomial time. → Min-Cost Fast Path ϵ NP
2- Choose Subset Sum for our reduction
3- Will show that Subset Sum $\leq_p$ Min-Cost Fast Path

   Background: Decision version of the Subset Sum problem asks whether given n items where item i has weight w_i, if there is a subset of them whose total weight is less than W and greater than M.

   Plan: We build a graph G such that it has an s-t path with total cost ≤W and total time ≤Σw_i – M iff there is a subset of items whose total weight is between M and W.

   We use gadgets to represent each item. Each gadget will offer two paths through the gadget. If the s-t path in G goes through the A node of the gadget we can interpret that as item being selected as part of the set. If the s-t path goes through the B node of the gadget we interpret that as the item not being selected as part of the set. We string up the gadgets and set time te and costs ce to edges as follows:



Proof:

A- If we are given a set of items with total weight between M and W, we can find a path from S to T with total cost of at most W and total time of at most $\Sigma w_i - M$ by choosing the path through each gadget based on whether the item is part of the set (Go through the A node) or not (go through the B node). If we go through the A node for object i, the object contributes w_i to the cost of the path and if we go through the B node, the object contributes w_i to the total time for the path. So the total cost for the path will be the total weight of the objects selected which we know if $\leq W$ and the total time of the path is total weight of the objects that are not selected which we know is $\Sigma w_i - M$ (because we know the total weight of the objects selected is $\geq M$)

B- If we are given a path from S to T which has a total cost of at most W and a total time of at most $\Sigma w_i - M$, we can find a set of objects with total weight between M and W. The S-T path can easily select the objects that belong to the set. If the path goes through the A node for an object, we place that object in the set, otherwise not. Since the total cost of the path is at most W then the total weight of the objects selected will be at most W and since the total time for the path is at most $\Sigma w_i - M$, the total weight of the objects not selected will be $\Sigma w_i - M$, which means that the total weight of objects selected will be at least M.

2. We saw in lecture that finding a Hamiltonian Cycle in a graph is **NP**-complete. Show that finding a Hamiltonian Path -- a path that visits each vertex exactly once, and isn't required to return to its starting point -- is also **NP**-complete.

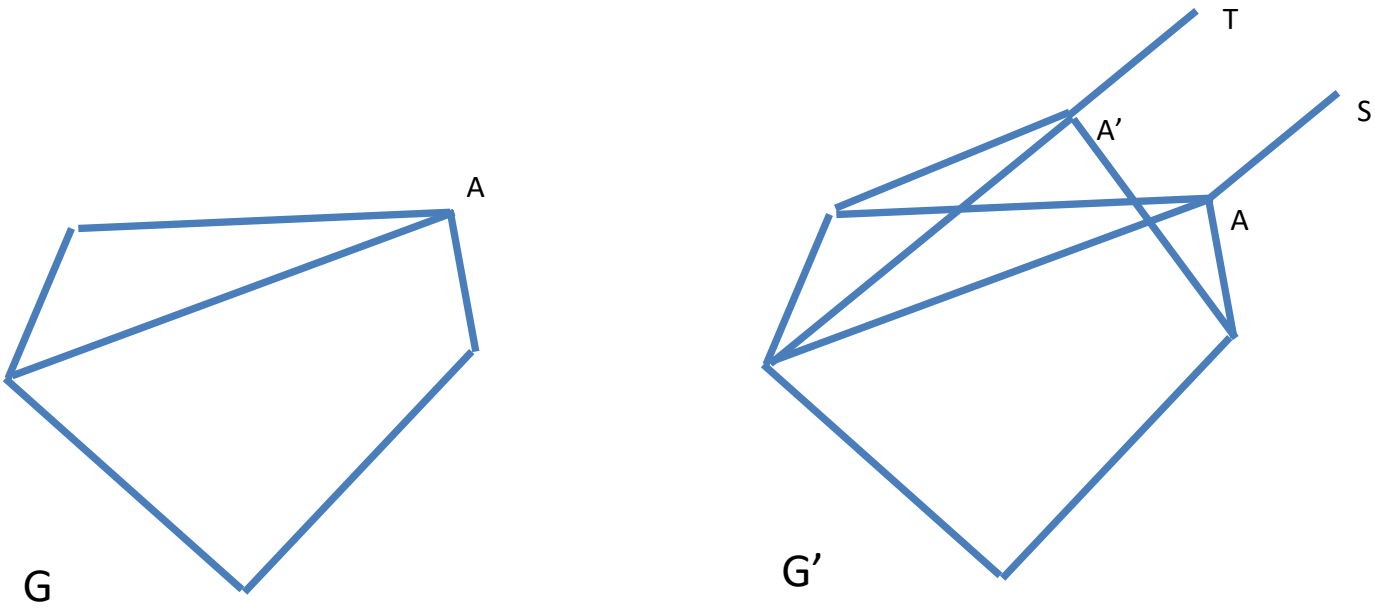
Solution:

4- Prove that Hamiltonian Path is in NP
   Certificate: an ordering of all nodes in G that forms a Hamiltonian Path
   Certifier: Can easily check in polynomial time that
   a- There is an edge between each pair of adjacent vertices in the given order
   b- All nodes in G are visited by the path
   a and b can be easily done in polynomial time. → Hamiltonian Path ϵ NP
5- Choose Hamiltonian Cycle for our reduction
6- Will show that Hamiltonian Cycle $\leq_p$ Hamiltonian Path

Plan: Given graph G—an instance of the Hamiltonian Cycle problem, we will construct G' such that G' has a Hamiltonian Path iff G has a Hamiltonian Cycle.

Construction of G'. We will split one of the nodes in G, say node A. Nodes A and A' will have the same connections as the original node A in G. We will then add two node nodes S and T and connect one with A and the other with A'.

Now G' has a Hamiltonian Path from S to T iff there is a Hamiltonian Cycle in G.



Proof:

A- If we are given a Hamiltonian Path in G', since S and T have a degree of 1, and can only be the beginning or the end of the path, the path must go from S to T. Ignoring the two new edges SA and TA', this path will give us a Hamiltonian Cycle in G since A and A' are the same node in G, i.e. the path will start and end at the same node (A).

B- If we are given a Hamiltonian Cycle in G, we can create a Hamiltonian Path in G' by splitting the Cycle at node A and creating a path from A' to A. We can then form a Hamiltonian Path in G' by starting at S going to A, following the Hamiltonian Cycle to A' and end the Path at T.

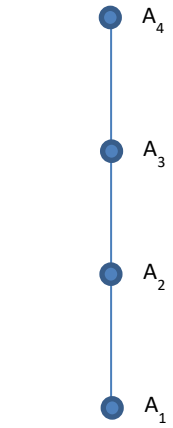
3. Some **NP**-complete problems are polynomial-time solvable on special types of graphs, such as bipartite graphs. Others are still **NP**-complete.
Show that the problem of finding a Hamiltonian Cycle in a bipartite graph is still **NP**-complete.

Solution:

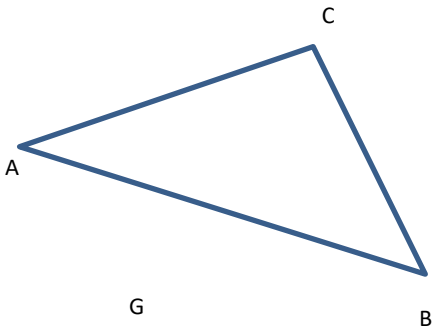
7- Prove that Hamiltonian Cycle in a bipartite graph is in NP
   Certificate: an ordering of all nodes in G that forms a Hamiltonian Cycle
   Certifier: Can easily check in polynomial time that
   a- There is an edge between each pair of adjacent vertices in the given order
   b- All nodes in G are visited by the path
   c- There is an edge between the last node in the order and the first node
   a, b and c can be easily done in polynomial time. → Hamiltonian Cycle in a bipartite graph ϵ NP
8- Choose Hamiltonian Cycle for our reduction
9- Will show that Hamiltonian Cycle $\leq_p$ Hamiltonian Cycle in a bipartite graph

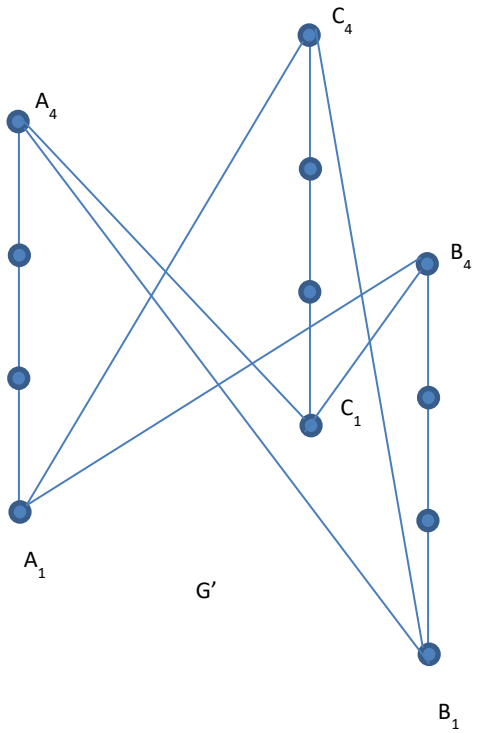
Plan: Given graph G—an instance of the Hamiltonian Cycle problem, we will construct G' such that G' is bipartite and has a Hamiltonian Cycle iff G has a Hamiltonian Cycle.

Construction of G': For each node A in G we will use a gadget with four nodes as shown below. If A and B are connected in G, we connect nodes A_1 and B_4 and nodes B_1 and A_4 in G'.



Gadget in G' representing the node A in G





G' is bipartite since we place all nodes V_1 and V_3 into the set X and nodes V_2 and V_4 into the set Y, all edges in G' go between sets X and Y. And G' has a Hamiltonian Cycle iff G has a Hamiltonian Cycle.

Proof:

A- If we are given a Hamiltonian Cycle in G' it must be of the form $V_1 V_2 V_3 V_4 U_1 U_2 U_3 U_4 \ldots V_1$ since there is no other way for the Hamiltonian Cycle to go through the nodes of each gadget. We can then use the same sequence of nodes V, U, …., V to form a Hamiltonian Cycle in G, since if there is a connection between V_4 and U_1 in G', there must be an edge between V and U in G.

B- If we are given a Hamiltonian Cycle in G that goes through nodes V, U, …., V we can form a Hamiltonian Cycle in G' by going through the gadgets corresponding to nodes V, U, …., V i.e. nodes $V_1 V_2 V_3 V_4 U_1 U_2 U_3 U_4 \ldots V_1$ since if there is a connection between nodes V and U, there must be a connection between nodes V_4 and U_1 in G'.

CSCI 570 - Fall 2021 - HW 10

Due November 18th

Graded

1. (20 pts) Consider the partial satisfiability problem, denoted as 3-Sat(α). We are given a collection of k clauses, each of which contains exactly three literals, and we are asked to determine whether there is an assignment of true/false values to the literals such that at least αk clauses will be true. Note that 3-Sat(1) is exactly the 3-SAT problem from lecture.

   Prove that 3-Sat(15/16)is NP-complete.

   Hint: If x, y, and z are literals, there are eight possible clauses containing them: (x ∨ y ∨ z), (!x ∨ y ∨ z), (x ∨ !y ∨ z), (x ∨ y ∨ !z), (!x ∨ !y ∨ z), (!x ∨ y ∨ !z), (x ∨ !y ∨ !z), (!x ∨ !y ∨ !z)

   Solution:

   To prove it's in NP: given a truth value assignment, we can count how many clauses are satisfied and compare it to 15k / 16.

   To prove it's NP-hard:

   We will show that 3-SAT ≤p 3-SAT(15/16). For each set of 8 original clauses, create 8 new clauses using 3 new variables, and construct the clauses considering the collection of all possible clauses on the 3 new variables. If the number of clauses is a multiple of 8, then we are done: any assignment will satisfy 7/8 of the new clauses, so we must satisfy all of the original clauses in a valid solution to satisfy exactly 15/16 of the clauses. If the number of clauses is not a multiple of 8, we will satisfy more than 15/16 of the clauses, but if even one of the original clauses is not satisfied, we will have satisfied less than 15/16 of the clauses.

   So, 3-Sat(15/16) is in the intersection of NP and NP-hard which is the class NP-Complete

2. (20 pts) Consider modified SAT problem, SAT' in which given a CNF formula having m clauses in n variables $x_1, x_2, \ldots, x_n$, the output is YES if there is an assignment to the variables such that exactly m − 2 clauses are satisfied, and NO otherwise. Prove that SAT' is also NP-Complete.

   Solution:

To show that SAT' is NP-Complete,

First we will show that SAT' ∈ NP:

Given the assignment values as certificate, we can evaluate the SAT' instance and verify if it is satisfied. This is same as the SAT-verification. Moreover, we can count the number of satisfied clauses and check if it is equal to m - 2 in linear time.

Next, we show that SAT ≤p SAT':

Construction: Add four more clauses x_1, x_2, $\neg x_1$, $\neg x_2$ to the original SAT instance.

Claim: CNF formula obtained for SAT', F' has an assignment which satisfies SAT' iff CNF formula of SAT, F has an assignment which satisfies SAT.

=>) if F has an assignment which satisfies SAT, then F' has an assignment which satisfies SAT'

Proof: If an assignment $x_1 \ldots x_n$ satisfies F, then it satisfies exactly two of the four extra clauses, giving exactly m + 2, which is nothing but m' – 2 satisfied clauses for the F'.

<=) if F' has an assignment which satisfies SAT', then F has an assignment which satisfies SAT

Proof: By construction, the only unsatisfied clauses for F must be one of x_1 or $\neg x_1$ and one of x_2 or $\neg x_2$, so all the original m clauses are satisfied.

3. (20 pts) Given a graph G=(V,E) and two integers k, m, the *Dense Subgraph Problem* is to find a subset V' of V, whose size is at most k and are connected by at least m edges. Prove that the *Dense Subgraph Problem is* NP-Complete.

   Solution:

   Proving this problem NP is trivial, So here we show how to prove it's NP-Completeness.

   We prove that the Independent set problem ≤P Dense Subgraph Problem.

   Given a graph G(V, E) and an integer k, an independent set decision problem outputs yes, if the graph contains an independent set of size k. For an arbitrary graph G=(V, E) of n vertices, we first get the complementary graph G_c of G.

   A clique is a subset of vertices of an undirected graph G such that every two distinct vertices in the clique are adjacent; that is, its induced subgraph is

complete. We also know that an independent set in G is a clique in G_c (the complement graph of G) and vice versa.

Then we set m to k*(k-1)/2 and test with the dense subgraph problem.

Claim: There exists an independent set of size k in G (equivalently, a clique in G_c of size k), iff there exists a subgraph of G_c with at most k vertices and at least m = k*(k-1)/2 edges.

=>) If there exists a clique in G_c of size at least k, then there exists a subgraph of G_c with at most k vertices and at least k*(k-1)/2 edges.

If there is a clique of size at least k then there is a clique of size exactly k. Moreover, by definition, a clique of size k would have k * (k-1)/2 edges.

<=) If there exists a subgraph of G_c with at most k vertices and at least k*(k-1)/2 edges, then there is a clique of size at least k.

For a subgraph to have k * (k-1) edges, implies there are k vertices. So this subset with k vertices forms a clique in G_c of size k.

Ungraded

4. (20 pts) (Modified from Textbook 8.16)Consider the problem of reasoning about the identity of a set from the size of its intersections with other sets. You are given a finite set U of size n, and a collection $A_1 \ldots A_m$ of subsets of U. You are also given numbers $c_1 \ldots c_m$, and numbers $d_1 \ldots d_m$. The question is:

   Does there exist a set $X \subset U$ so that for each i = 1…m, the cardinality of $X \cap A$ is larger than c_i but smaller than d_i? We will call this an instance of the *Intersection Inference Problem, with* input U, $\{A_i\}$, and $\{c_i\}$, $\{d_i\}$. Prove that Intersection Inference is NP-complete.

   Solution:

   To show this problem is NP-Complete, we first prove it's NP.

   Given the set $X \subset U$, we can enumerate all Bi in the collection, calculate the intersection with X, and compare its cardinality with c_i and d_i. So this problem is NP.

   Then, we prove that the vertex cover $\leq_P$ Intersection Inference.

   For an arbitrary graph G = (V, E) of n vertices, we let V to be set U, and construct n+1 subsets $A_1 \ldots A_{n+1}$ by: A_i (1<=i<=n) consists of all adjacent nodes of $v_i \in V$, and A_{n+1}= U. Then We set $c_1 \ldots c_n$ to 1, $d_1 \ldots d_n$ to infinity, c_{n+1} to 0, d_{n+1} to k. The solution of the intersection inference problem with this inputs is exactly the vertex cover of G of size no more than k.

5. (20 pts) (Textbook 8.28) The following is a version of the independent Set Problem. you are given a graph G = (V, E) and an integer k. For this problem, we will call a set $I \subset V$ *strongly independent if,* for any two nodes v, u ∈ I, the edge (v, u) does not belong to E, and there is also no path of two edges from u to v, that is, there is no node w such that both (u, w) ∈ E and (w, v) ∈ E. The Strongly independent Set Problem is to decide whether G has a strongly independent set of size at least k.

   Prove that the Strongly independent Set Problem is NP-complete.

   Solution:

   To show this problem is NP-Complete, we first prove it's NP.

   Given a vertex subset V' $\subseteq V$, we can check in polynomial time if any pair of vertices u, v ∈ V' are either connected by an edge in E for by another node w ∈ V. Thus we can determine if a vertex subset if a strong independent set in polynomial time.

   Then, We prove that Independent Set $\leq_P$ Strong Independent Set.

   For an arbitrary graph G = (V,E), we split every edge in E into two nodes and connect them by a new node w_e (that means, an edge (u,v) in E becomes two edges (u,w_e) and (v,w_e). Then we connect all w_e nodes pairwise. Mathematically, we generate a new graph G+=(V+, E+) where V+ = V $\cup$ {w_e | e ∈ E}, and E+ = {(u,w_e), (v,w_e) | e=(u, v), e ∈ E} $\cup$ {(w_e ,$w_{e'}$) | e, e' ∈ E}. To determine if there's an independent set in G of size at least k, we can check if there's any strong independent set in G+ with the same condition. Then:

   1.If there is not such set: It's easy to prove that any independent set of G is also a strong independent set of G+, so there's also no independent set of size at least k in G.

   2. If there is such set: Let V+' be that set. Since V+' is a strong independent set of G+, if any node in x ∈ {w_e | e ∈ E} is in V+', then V+' can only contain x (because all other nodes in V+ are all not further than 2 to x). In this case, |V+'| = 1, and it's trivial to find an independent set of size 1 in G. On the other hand, if V+' don't intersect with {w_e | e ∈ E}, then V+' $\subseteq$ V, making it an

independent set of V. In both cases, there's an independent set in G of size at least k.

CSCI 570 - Fall 2021 - HW 11

Graded Problems

Problem 1

[**15 points**] Given an undirected graph with positive edge weights, the BIG-HAM- CYCLE problem is to decide if it contains a Hamiltonian cycle C such that the sum of weights of edges in C is at least half of the total sum of weights of edges in the graph. Show that finding BIG-HAM-CYCLE in a graph is NP-Complete.

Solution: The certifier takes an undirected graph (the BHC instance) and a sequence of edges as input. It verifies that the sequence of edges form a Hamiltonian cycle and that the total weight of the cycle is at least half of the total weight of the edges in the graph. Thus BIG-HAM-CYCLE is in NP.
We claim that Hamiltonian Cycle is polynomial time reducible to BIG-HAM-CYCLE. To see this, given an instance of problem Hamiltonian cycle in an undirected graph $G = (V,\ E)$, pick an edge e and set its weight to $|E|$ and assign the rest of the edges weight of 1. When this weighted graph is fed into the BIG-HAM-CYCLE decider blackbox, it returns "yes" if and only if G has a Hamiltonian cycle containing the edge e. By repeating the above once for every edge e in the graph G, we can decide if the graph has a Hamiltonian cycle.

Rubric:

- 5 points for proving the problem is in NP.
- 3 points for reducing from Hamiltonian Cycle.
- 7 points for running BIG-HAM-CYCLE on new weighted graph.

Problem 2

[**15 points**] Show that vertex cover remains NP-Complete even if the instances are restricted to graphs with only even degree vertices.

Solution: The certifier takes an undirected graph (the VC instance) and a set of vertices as input. It verifies the union of all edges belong to the each

vertex cover all the original graph's edges. Thus problem is in NP.(the same certifier to the original vertex cover problem)
We claim that vertex cover with only even degree vertices remains NP-Complete by showing it is polynomial time reducible from the original vertex cover problem. Let $(G = (V, E), k)$ to be an input instance of Vertex Cover. Because each edge in E contributes a count of 1 to the degree of each of the vertices with which it connects, the sum of the degrees of the vertices is exactly $2|E|$, an even number. Hence, there is an even number of vertices in G that have odd degrees. Let U be the subset of vertices with odd degrees in G.
Construct a new instance $\overline{(G} = (V_0, E_0), k + 2)$ of Vertex Cover, where $V_0 = V \cup \{x, y, z\}$ and $E_0 = E \cup \{(x, y), (y, z), (z, x)\} \cup \{(x, v)|v \subset U\}$. That is, we make a triangle with three new vertices, and then connect one of them (say x) to all the vertices in U. The degree of every vertex in V_0 is even. Since a vertex cover for a triangle is of (minimum) size 2, it is clear that V_0 has a vertex cover of size k + 2 if and only if G has a vertex cover of size k.
Hence, vertex cover with only even degree vertices is NP Complete.

Rubric:

- 5 points for proving the problem is in NP.
- 2 points for reducing from vertex cover.
- 8 points for correctly reconstructing the new graph.

Problem 3

[**15 points**] Given an undirected connected graph G= (V, E) in which a certain number of tokens $t(v) \geqslant 1$ placed on each vertex v. You will now play the following game. You pick a vertex u that contains at least two tokens, remove two tokens from u and add one token to any one of adjacent vertices. The objective of the game is to perform a sequence of moves such that you are left with exactly one token in the whole graph. You are not allowed to pick a vertex with 0 or 1 token. Prove that the problem of finding such a sequence of moves is NP-complete by reduction from Hamiltonian Path.

Solution:
Construction: given a HP in G, we construct G' as follows. Traverse a HP in G and placed 2 tokens on the starting vertex and one token on each other vertex in the path.

Claim: G has a HP iff G' has a winning sequence.
- by construction before the last move we will end up with a single vertex having two tokens on it. Making the last move, we will have exactly one token on the board.
-since there is only one vertex with 2 tokens, we will start right there playing the game. Each next move is forced. When we finish the game, we get a sequence

moves which represents a HP.

Rubric:

- Didn't prove it is in NP: -5
- Didn't prove it is NP-hard: -10
- Assigned two tokens on one random vertex instead of the starting vertex of Hamiltonian path: -2 (Since it is a reduction from Hamiltonian path, not from Hamiltonian cycle, 2 tokens should be assigned on the starting vertex)
- Assigned wrong number of tokens on one vertex: -3
- Assigned wrong number of tokens on two vertices: -6
- Assigned wrong number of tokens on three or more vertices(considered as not a valid reduction from Hamiltonian path): -7
- Not a valid reduction from Hamiltonian path: -7

Ungraded Problems

Problem 1

You are given a directed graph G = (V, E) with weights we on its edges. The weights can be negative or positive. The Zero-Weight-Cycle Problem is to decide if there is a simple cycle in G so that the sum of the edge weights on this cycle is exactly 0. Prove that this problem is NP-complete.

Solution:
see https://en.wikipedia.org/wiki/Zero-weight_cycle_problem

Problem 2

The graph five-coloring problem is stated as follows: Determine if the vertices of G can be colored using 5 colors such that no two adjacent vertices share the same color.
Prove that the five-coloring problem is NP-complete.
Hint: You can assume that graph 3-coloring is NP-complete.

Solution:
Show that 5-coloring is in NP:
Certificate: a color solution for the network, i.e., each node has a color.
Certifier:

i. Check for each edge (u,v), the color of node u is different from the color of node v.
ii. Check at most 5 colors are used.

Show that 5-coloring is NP-hard: prove that 3-coloring $\leqslant$p 5-coloring.
Graph construction:
Given an arbitrary graph G. Construct G' by adding 2 new nodes u and v to G. Connect u and v to all nodes that existed in G, and to each other.
G' can be colored with 5 colors iff G can be colored with 3 colors.

i. If there is valid 3-color solution for G, say using colors 1,2,3, we want to show there is a valid 5-coloring solution to G'. We can color G' using five colors by assigning colors to G according to the 3-color solution, and then color node u and v by additional two different colors. In this case, node u and v have different colors from all the other nodes in G', and together with the 3-coloring solution in G, we use at most 5 colors to color G'.

ii. If there is a valid 5-coloring solution for G', we want to show there is a valid 3-coloring solution in G. In G', since node u and v connect to all the other nodes in G and to each other, the 5-coloring solution must assign two different colors to node u and v, say colors 4 and 5. Then the remaining three colors 1,2,3 are used to color the remaining graph G and form a valid 3-color solution.

Solving 5-coloring to G' is as hard as solving 3-color to G, then 5-coloring problem is at least as hard as 3-coloring, i.e., 3-coloring $\leqslant$p 5-coloring.

CSCI 570 - Fall 2021 - HW 12

Due November

Graded Problems

Problem 1

800 students in the "Analysis of Algorithms" class in 2021 Fall take the exams onsite. The university provided 9 classrooms for exam use, each classroom can contain C_i (capacity) students. The safety level of a classroom is proportional to $\alpha_i(C_i - S_i)$, where α_i is the area size of the classroom and S_i is the actual number of students taking the exams in the classroom. Due to the pandemic, we want to maximize the total safety level of all the classroom. Besides, to guarantee students have a comfort environment, the number of students in a classroom should not exceed half of the capacity of each classroom.

Express the problem as a linear programming problem. You **DO NOT** need to solve it.

Solution

Our variables are S_i. Our objective function is:

$$maximize \sum_{i=1}^{9} \alpha_i(C_i - S_i)$$

subject to

$$S_i \geq 0, for i = 1, ..., 9$$

$$S_i \leq \frac{1}{2}C_i, for i = 1, ..., 9$$

$$\sum_{i=1}^{9} S_i = 800$$

Problem 2

A triangle is a set of three vertices, every two of which is connected by an edge. Consider the following minimization problem that we refer to as triangle

removal. The input is a graph $G(V, E)$. The goal is to select a minimum subset of edges whose removal from the graph gives a new graph with no triangles.

Design a strongly polynomial time algorithm that provides a factor 3 approximation for triangle removal.

Solution

We design a greedy algorithm for reaching a 3-approximation. Start with the graph G. In each step find a triangle (this can be done by checking each (u, v) edge, and checking all other nodes, e.g., x to find out if $(u, x) \in E(x, v) \in E$). Add all edges (u, v), (v, x), and (x, u) to the solution set F (and remove them from E), and continue until G contains no more triangles.

Time Complexity Finding a triangle require $O(mn)$, the removing process is performed at most m times. So the overall time complexity is $O(m^2n)$

3-approximation Consider the maximum number of edge disjoint triangles to be $\alpha(G)$ and the optimal answer of the question to be $\beta(G)$. We know that $\alpha(G) \leq \beta(G)$ because at least of edge from each disjoint triangle should be removed.
Note that by design of our algorithm, the set of triangles found in subsequent iterations are all edge disjoint. If there's κ of them, this must be $\leq$ the maximum, i.e. $\alpha(G)$. The set of edge disjoint triangles that we have found at the end of this process (κ) is a lower bound on the size of maximum set of disjoint triangles. Therefore $\kappa \leq \alpha(g) \leq \beta(G)$. (We have proved $\alpha(g) \leq \beta(G)$ in the previous paragraph.) Our solution set F has size $|F| = 3\kappa \leq 3\alpha(G) \leq 3\beta(G)$. Therefore, this greedy procedure is a 3-approximation algorithm.

Problem 3

A company makes three products and has 4 available manufacturing plants. The production time (in minutes) per unit produced varies from plant to plant as shown below:

		Manufacturing Plant			
		1	2	3	4
Product	1	5	7	4	10
	2	6	12	8	15
	3	13	14	9	17

Similarly the profit ($) contribution per unit varies from plant to plant as below:

		Manufacturing Plant			
		1	2	3	4
Product	1	10	8	6	9
	2	18	20	15	17
	3	15	16	13	17

If, one week, there are 35 working hours available at each manufacturing plant how much of each product should be produced given that we need at least 100 units of product 1, 150 units of product 2 and 100 units of product 3. Formulate this problem as a linear program. You do not have to solve the resulting LP.

Solution

Variables At first sight we are trying to decide how much of each product to make. However on closer inspection it is clear that we need to decide how much of each product to make at each plant. Hence let x_{ij} = amount of product i ($i = 1, 2, 3$) made at plant j ($j = 1, 2, 3, 4$) per week. Although (strictly) all the x_{ij} variables should be integer they are likely to be quite large and so we let them take fractional values and ignore any fractional parts in the numerical solution. Note too that the question explicitly asks us to formulate the problem as an LP rather than as an IP.

Constraints We first formulate each constraint in words and then in a mathematical way. Limit on the number of minutes available each week for each workstation

$$5x_{11} + 6x_{21} + 13x_{31} \leq 35(60)$$
$$7x_{12} + 12x_{22} + 14x_{32} \leq 35(60)$$
$$4x_{13} + 8x_{23} + 9x_{33} \leq 35(60)$$
$$10x_{14} + 15x_{24} + 17x_{34} \leq 35(60)$$

Lower limit on the total amount of each product produced

$$x_{11} + x_{12} + x_{13} + x_{14} \geq 100$$
$$x_{21} + x_{22} + x_{23} + x_{24} \geq 150$$
$$x_{31} + x_{32} + x_{33} + x_{34} \geq 100$$

All variables are greater than equal to zero.

Objective Maximize total profit. Maximize

$$10x_{11}+8x_{12}+6x_{13}+9x_{14}+18x_{21}+20x_{22}+15x_{23}+17x_{24}+15x_{31}+16x_{32}+13x_{33}+17x_{34}$$

Ungraded Problems

Problem 1

Suppose you have a knapsack with maximum weight W and maximum volume V. We have n dividable objects. Each object i has value m_i, weights w_i and takes v_i volume. Now we want to maximize the total value in this knapsack, and at the same time We want to use up all the space in the knapsack. Formulate this problem as a linear programming problem. You DO NOT have to solve the resulting LP.

Solution

Let's denote the quantity of each object by a_i. Our objective function is:

$$maximize \sum_{i=1}^{n} a_i m_i$$

subject to

$$\sum_{i=1}^{n} a_i w_i \leq W$$

$$0 \leq a_i \leq 1, for i = 1, ..., n$$

$$\sum_{i=1}^{n} a_i v_i = V$$

Problem 2

Given a graph G and two vertex sets A and B, let $E(A, B)$ denote the set of edges with one endpoint in A and one endpoint in B. The **Max Equal Cut** problem is defined as follows

Instance Graph $G(V, E)$, $V = \{1, 2, ..., 2n\}$.

Question Find a partition of V into two n-vertex sets A and B, maximizing the size of $E(A, B)$.

Provide a factor 1/2-approximation algorithm for solving the **Max Equal Cut** problem.

Solution

Start with empty sets A, B, and perform n iterations:
In iteration i, pick vertices $2i - 1$ and $2i$, and place one of them in A and the other in B, according to which choice maximizes $|E(A, B)|$.

(i.e., if $|E(A \cup \{2i-1\}, B \cup \{2i\})| \geq |E(A \cup \{2i\}, B \cup \{2i-1\})|$ then add $2i$ - 1 to A and $2i$ to B, else add $2i$ to A and $2i$ - 1 to B.)

In a particular iteration, when we have cut (A, B) and we add u and v. Suppose u has N_{Au}, N_{Bu} neighbours in A, B respectively and suppose v has N_{Av}, N_{Bv} neighbours in A, B respectively. Then, adding u to A and v to B adds N_{Bu} + N_{Av} edges to the cut, whereas doing the other way round adds N_{Bv} + N_{Au} edges to the cut. Since the sum of these two options is nothing but the total number of edges being added to this partial subgraph, the bigger of the two must be at least half the total number of edges being added to this partial subgraph. Since this is true for each iteration, at the end when all the nodes (and edges) are added, our algorithm is bound to add at least half of the total $\|E\|$ edges. Naturally since the max equal cut capacity OPT $\leq \|E\|$, our solution is 1/2-approximation.