P/NP

Review Session CSCI 570
Exam 3

Quick Concept Review: Any Doubts?

- Decision Problems?
- P: Decision problems deterministically *solvable in polynomial time*
- NP: Decision problems deterministically *verifiable in polynomial time* (Certificate? Verifier?)
- Polynomial-time Reducible?
- NP-Hard: A problem H is NP-Hard if every problem in NP can be reduced to H in polynomial time.
- NP-Complete? (How to prove that a given decision problem is NP-Complete?)
- Why are reductions useful and how to use them?
- Example: To show that IND-SET is NP-Hard, why is it sufficient to show that 3-SAT (already known to be NP-Complete) can be reduced to IND-SET?

NP- True / False Questions

If INDEPENDENT SET can be reduced to a decision problem X, then X can be reduced to INDEPENDENT SET.

If INDEPENDENT SET can be reduced to a problem X, then X can be reduced to INDEPENDENT SET.

False. If X is in NP, then it would be guaranteed true, but cannot claim without that information.

NP- True / False Questions

All the NP-hard problems are also in NP.

All the NP-hard problems are also in NP.

False. The NP-hard problems that are in NP are called NP-complete but that's only a strict subset of the NP-hard problems.

NP - Long Question 1

The CLIQUE Problem: Given an undirected graph G = (V, E), and a positive integer k, decide whether the graph G contains a ***clique*** of size at least k.

A ***clique*** is a subset of vertices in graph G such that every pair of vertices is connected by an edge.

Prove that the CLIQUE problem is NP-complete.

The CLIQUE Problem: Given an undirected graph G = (V, E), and a positive integer k, decide whether the graph G contains a ***clique*** of size at least k.

A ***clique*** is a subset of vertices in graph G such that every pair of vertices is connected by an edge.

Prove that the CLIQUE problem is NP-complete.

a) CLIQUE is in NP: A subset of vertices V', claimed to be a part of the clique - can be a certificate.
   Verify:
   - The number of vertices are at least k.
   - There is an edge between any two vertices in V'.
   - Runs in polynomial time?

The CLIQUE Problem: Given an undirected graph G = (V, E), and a positive integer k, decide whether the graph G contains a ***clique*** of size at least k.

A ***clique*** is a subset of vertices in graph G such that every pair of vertices is connected by an edge.

Prove that the CLIQUE problem is NP-complete.

b) CLIQUE is NP-Hard

The CLIQUE Problem: Given an undirected graph G = (V, E), and a positive integer k, decide whether the graph G contains a ***clique*** of size at least k.

A ***clique*** is a subset of vertices in graph G such that every pair of vertices is connected by an edge.

Prove that the CLIQUE problem is NP-complete.

b) CLIQUE is NP-Hard

Plan:

- Can reduce from IND-SET or 3SAT

The CLIQUE Problem: Given an undirected graph G = (V, E), and a positive integer k, decide whether the graph G contains a ***clique*** of size at least k.

b) IND-SET to CLIQUE

IND-SET Problem: Given (G,k), is there an independent set of size at least k?

Construction: Create an instance of the CLIQUE problem using (GC, k) - GC being the complement of G (edges reversed)

Proof:

1) If G has an independent-set of size at least k, means that any two vertices in this set do not have an edge in G -> means that they have an edge in GC -> i.e. the same set of vertices form a clique in GC -> proving that GC has a clique of size k.

2) If GC has a clique of size at least k, the same vertices would form an independent set in G.

3SAT to CLIQUE -

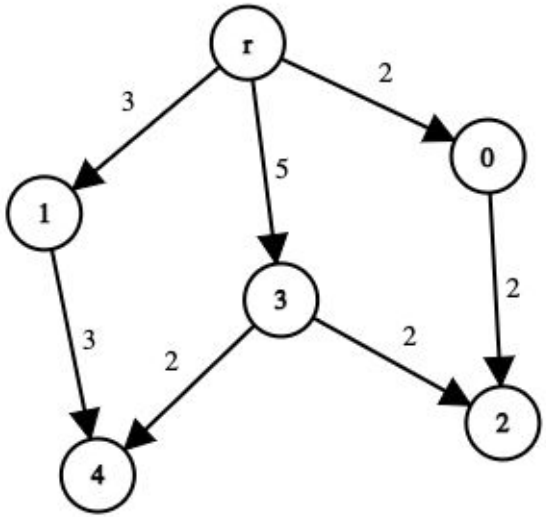
- Key Idea: Literals in the clause → Vertices in the graph. Connect vertices from different clauses that can be consistently set to 1 with each other.
- See Theorem 34.11 in Coreman

NP - Long Question 2

Suppose you move to a new city. The city is defined by a directed graph G=(V,E) and each edge $e \in E$ has a non-negative cost c_e associated to it. Your living place is represented as a node $s \in V$. There is a set of landmarks X (a subset of V), which you want to visit. To plan your roaming around efficiently, you are interested in finding a subgraph G'=(V',E') that contains a path from *s* to each $x \in X$ while minimizing the total edge cost of E'. The decision problem (call it ROAM) is, given a graph G=(V,E) with non-negative costs, vertex subset X of V, a node $s \in V$, and a number k, does there exist a subgraph G'=(V',E') that contains a path from *s* to each $x \in X$, having a total edge cost of E' at most k? Show that ROAM is NP-complete with a reduction from SAT.

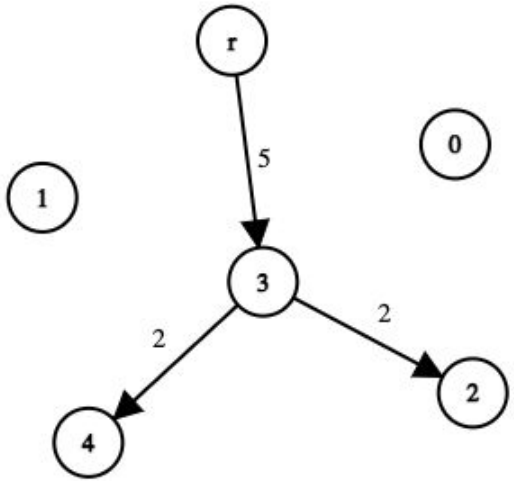
Example

Let’s take the below example with X = {2, 4} and s = r. Is there a way to select edges such that all nodes in X are reachable by s and the total edge cost is at most 9?



Example

The solution to above problem can be seen below. We can see that both nodes 2 and 4 are reachable by r and total edge cost is 5 + 2 + 2 = 9.



Suppose you move to a new city. The city is defined by a directed graph G=(V,E) and each edge $e \in E$ has a non-negative cost c_e associated to it. Your living place is represented as a node $s \in V$. There is a set of landmarks X (a subset of V), which you want to visit. To plan your roaming around efficiently, you are interested in finding a subgraph G'=(V',E') that contains a path from s to each $x \in X$ while minimizing the total edge cost of E'. The decision problem (call it ROAM) is, given a graph G=(V,E) with non-negative costs, vertex subset X of V, a node $s \in V$, and a number k, does there exist a subgraph G'=(V',E') that contains a path from s to each $x \in X$, having a total edge cost of E' at most k? Show that ROAM is NP-complete with a reduction from 3SAT.

a) Show that ROAM is in NP

Solution: We show that a subgraph G' as a certificate. Certifier checks that all the nodes in X are reachable - can be done using DFS, thus in poly-time. Then check the total edge weight is within k - doable in linear (hence poly-) time.

Suppose you move to a new city. The city is defined by a directed graph G=(V,E) and each edge $e \in E$ has a non-negative cost c_e associated to it. Your living place is represented as a node $s \in V$. There is a set of landmarks X (a subset of V), which you want to visit. To plan your roaming around efficiently, you are interested in finding a subgraph G'=(V',E') that contains a path from s to each $x \in X$ while minimizing the total edge cost of E'. The decision problem (call it ROAM) is, given a graph G=(V,E) with non-negative costs, vertex subset X of V, a node $s \in V$, and a number k, does there exist a subgraph G'=(V',E') that contains a path from s to each $x \in X$, having a total edge cost of E' at most k? Show that ROAM is NP-complete with a reduction from 3SAT.

Hints for construction

1) 'Satisfy ALL clauses' VS 'Reach ALL nodes in X'
2) Variables serve as ways to satisfy clauses VS nodes/edges in G serve as ways to satisfy the reachability constraints of X,

(b) Reduction from SAT: Construct

A new node r for living place.

Nodes x_i and $\sim x_i$ for each literal, all connected to r.

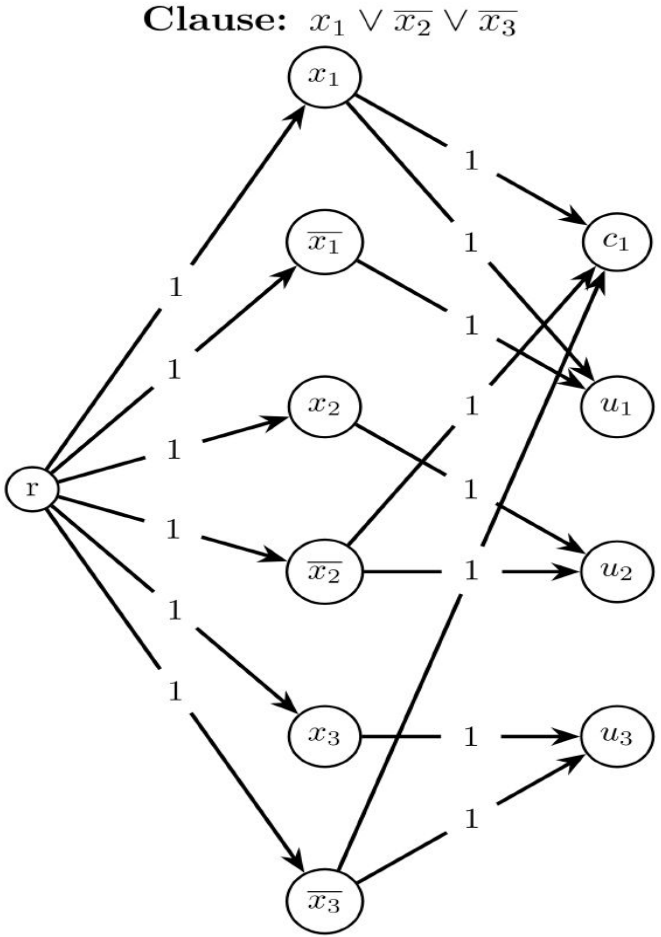
Nodes c_i for each clause, connected to its literals.

Nodes u_i for each variable, connected to x_i & $\sim x_i$

All edges have weight 1.

Subset X contains all u_i and c_i

K = 2n + m (as used in the claim next)
(n = #vars, m = #clauses)



(b) Construction:

A new node r for living place.

Nodes x_i and $\sim x_i$ for each variable, all connected to r.

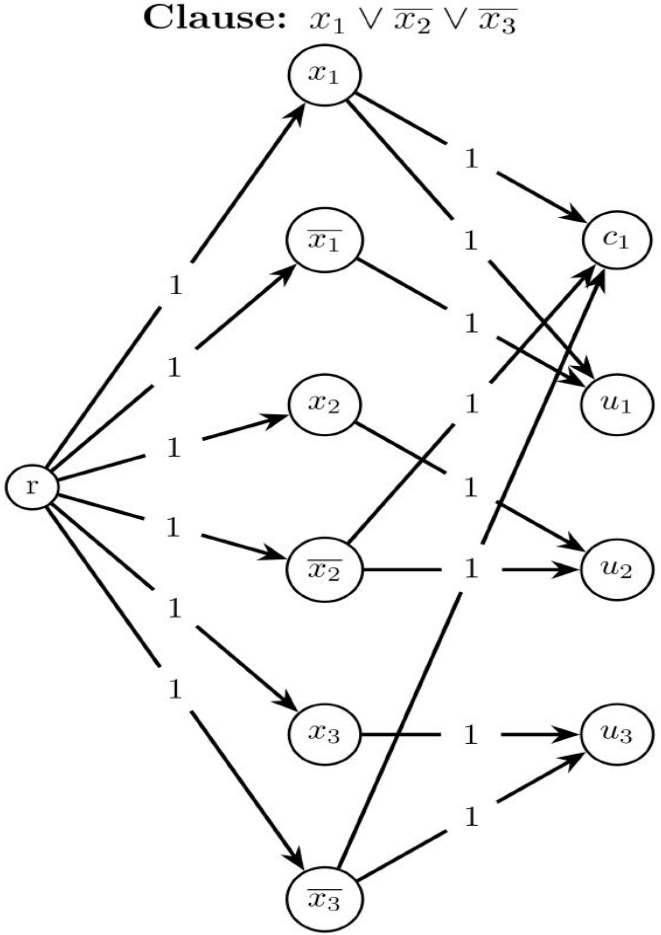
Nodes c_i for each clause, connected to corresponding literals.

Nodes u_i for each variable connected to x_i and $\sim x_i$

All edges have weight 1.

Subset X contains all u_i and c_i

(c) Claim: Given 3-SAT instance (i.e. the 3CNF formula) has a solution (i.e. a satisfying assignment) **if and only if** the constructed graph has a subgraph G' containing paths from r to each x in X, with total edge weight at most 2n + m

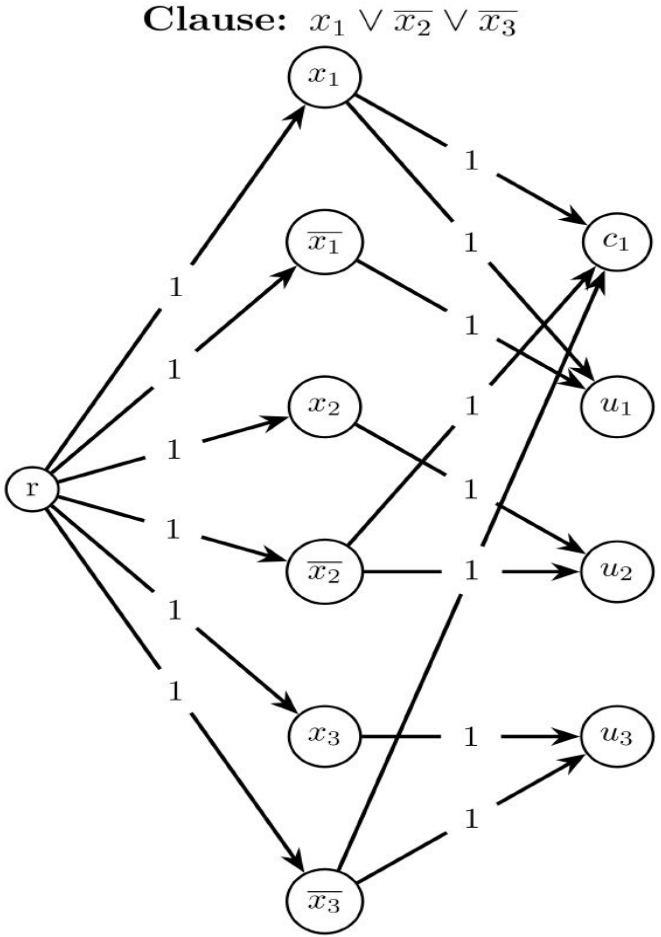


(d) SAT has a satisfying assignment => Constructed graph has a subgraph with routes from r to each x in X with weight at most 2n + m

Proof: Consider the satisfying assignment. For any x_i, let L_i be the literal that is true (i.e. x_i or $\sim x_i$). Construct the subgraph by including

1) The nodes corresponding to each L_i (alongwith the source node r & all the destination nodes u_i and c_j)

2) All the edges r -> L_i and L_i -> u_i so that u_i is reachable from r (thus, 2n such edges). Further, since the subgraph is constructed from a satisfying assignment, some literal in each clause is true and we add the edge from the corresponding literal node to the clause node (thus, m such edges) so that all c_j are reachable.

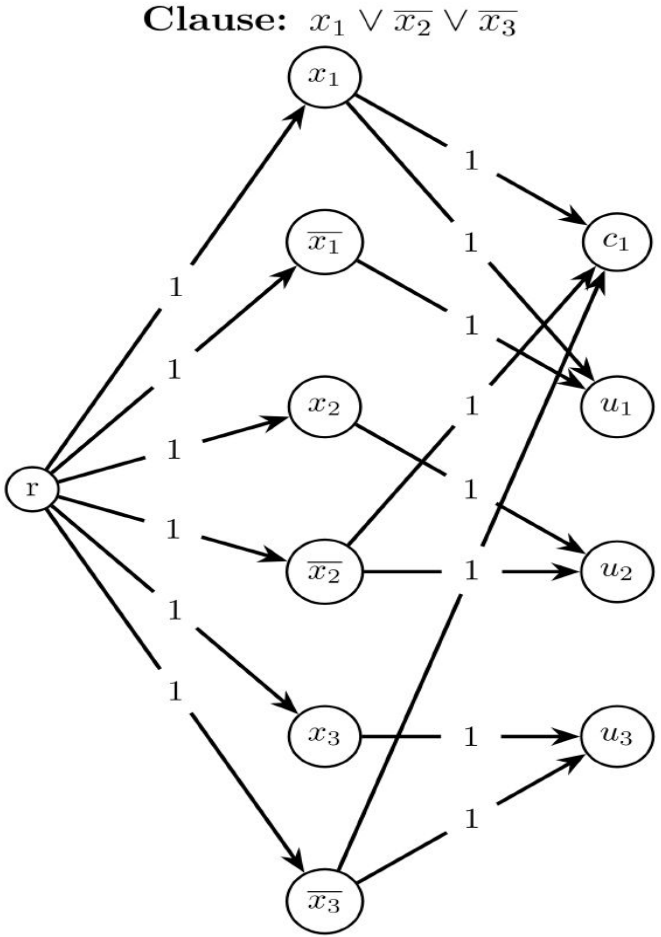


(e) Constructed graph has a subgraph spanning X with weight at most 2n + m => SAT has a satisfying assignment

Proof: Consider the spanning subgraph H.

1) H must have the node L_i and edges r -> L_i and L_i -> u_i for either $L_i = x_i$ or $L_i = \sim x_i$ so that u_i is reachable from r (thus, 2n such edges, and n nodes).
2) Further, for each clause, there must be an edge in H from the literal node to the clause node (thus, m such edges) so that it's reachable from r. Additionally if this literal is not already included from 1), there must be an edge from r to this literal to make the clause node reachable.

Since H has a total edge weight at most 2n + m, it must have precisely the n nodes and 2n+m edges mentioned above. Then, we obtain an assignment by setting each x_i to true/false so as to make the corresponding L_i (that is included in H) true. This is a complete assignment since H includes a node for each variable as shown in 1). Further this is a satisfying assignment since for each clause, one of the literals (which makes it reachable from r in H) becomes true, thus satisfying the clause.



NP - Long Question 3

Given an undirected connected graph $G = (V, E)$ in which a certain number of tokens $t(v) \geq 1$ 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.

Given an undirected connected graph $G = (V, E)$ in which a certain number of tokens $t(v) \geq 1$ 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.

A) Show that the above problem is in NP.

Solution: The sequence of moves will act as a certificate. The certifier can perform the sequence of moves and then go through all the nodes in the graph to check if only one token exists in the entire graph. Since, this can be done in linear time, the problem is in NP.

Given an undirected connected graph $G = (V, E)$ in which a certain number of tokens $t(v) \geq 1$ 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.

Hints for construction:

1) Sequence of moves vs path in a graph.
2) Can leverage token assignment to ensure that each node is visited exactly once.

Given an undirected connected graph $G = (V, E)$ in which a certain number of tokens $t(v) \geq 1$ 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.

Construction:

Let's reduce Hamiltonian path to the above problem.

We call the above problem V times, once considering each node u as the starting point. The graph G' is the same graph G and we place 2 tokens on the starting node and 1 token on all the remaining nodes.

If any of this problem with starting point u has a solution, then there is a hamiltonian path starting at node u in G.

Given an undirected connected graph $G = (V, E)$ in which a certain number of tokens $t(v) \geq 1$ 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.

Forward Claim:

If there is a Hamiltonian path in the graph, then there will be a valid sequence of moves for at least one call to the above problem. Let's say the hamiltonian path in G is the sequence of nodes x1, x2, ….. xn. Then, when we use the above problem considering x1 as a starting point, we can go through the list of nodes in the hamiltonian path. We know that x1 has two tokens and rest of the nodes have 1 token. When we consider x1 as a starting point, we delete its two tokens and add a token to x2, so now x2 has two tokens. Once a node gets visited, there will be 0 tokens in that node and will have added a token to the next node in the sequence of moves(making its tokens 2). Since the path visits every node, we have removed the one token on each of the node. When we visit last vertex, we remove its tokens and add one token to a random neighbour node and hence, only one token is remaining in the entire graph.

Given an undirected connected graph $G = (V, E)$ in which a certain number of tokens $t(v) \geq 1$ 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.

Backward Claim:

If there is a valid sequence of move for any of the V calls to the above problem, then there is a Hamiltonian path in G. Let's say there is a valid sequence of moves x1, x2, …. xn, xn + 1, when we consider the problem with only x1 having two tokens. We can see that the only possible starting point is x1 as there is no other node with two tokens. Also, when we push a token to node x2, our next move needs to be removing two tokens from x2, as there is no other node with two tokens. Once we have visited a node, we can't visit it again as the node now has 0 tokens and pushing a token to it makes the number of tokens 1. Hence, x1, x2, …. xn are unique nodes. We also need to visit each node at least once so that there is no more than 1 token remaining in the entire graph. This shows that the path x1, x2, ….. xn has visited each node exactly once. Since we can push token only to a neighbour node, the sequence x1, x2, ….., xn will be a valid path and hence, will be a valid hamiltonian path.

Linear Programming

Review Session CSCI 570
Exam 3

Minimum Weighted Set Cover

Given a set U of n elements, and m subsets of U denoted by S_j, $j \in 1, 2, \ldots, m$, and a weight w_j for each subset, find a k-approximation of the minimum-weighted set cover.

You are given that the union of all S_j's equals U and each element in U belongs to at most k subsets. Weight of a set cover is the sum of weights of the subsets included in the set cover.

Use Integer Linear Programming to solve this problem.

Steps of the solution

1. Formulate the problem as an Integer Linear Programming (ILP) problem
2. Relax the constraints to convert to a Linear Programming (LP) problem
3. Use rounding to get an approximate solution
4. Show that the approximate solution is a valid set cover
5. Prove that the approximate solution is at most k times the optimal solution

Step 1: Formulate as ILP

Suppose U = {u_1, u_2, … , u_n}

Define a decision variable x_i for each subset S_i, i = 1 to m

x_i = 1 if S_i is included in the set cover, else 0

For a collection of subsets to be a set cover, each element should be present in its union.

$$\sum_{u_i \in S_j} x_j \geq 1 \quad \forall\, u_i \in U$$

Step 1: Formulate as ILP

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

$$\text{subject to:}$$

$$\sum_{u_i \in S_j} x_j \geq 1 \quad \forall\, u_i \in U$$

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

Step 2: Relax to LP

Drop the requirement that $x_i \in \{0, 1\}$, and convert it to $0 \leq x_i \leq 1$

Now the constraints are linear and we have converted the integer linear programming problem to a linear programming problem which can be solved in polynomial time

Let $\{x^*\}$ be the LP solution, and $W^{LP} = \sum w_i x_i^*$

Step 3: Rounding to get approximate solution

We include a set S_i in the set cover if $x_i^* \geq 1/k$

So our approximate set cover is $F = \{S_i \mid x_i^* \geq 1/k\}$

Step 4: Show that your approximate solution is valid

$$\sum_{u_i \in S_j} x_j \geq 1 \quad \forall\, u_i \in U$$

We show that F is indeed a correct set cover. Consider above constraint for each element u_i.

Each u_i belongs to at most k subsets. Therefore it is not possible for all $x_j < 1/k$ because then $\sum x_j < \sum 1/k < k \times 1/k = 1$, violating above constraint.

Therefore at least one of x_j should be $\geq 1/k$

Therefore at least one of S_j will be included in our approximate set cover F

Therefore all elements are covered by F

Step 5: Prove that your approximate solution ≤ k optimal solution

Let F^{OPT} be the minimum-weighted set cover, and W^{OPT} = sum of weights of subsets included in F^{OPT}

W^{OPT} is the value of the objective function of the original ILP problem

W^{LP} is the value of the objective function of the relaxed LP problem

Therefore $W^{LP} \leq W^{OPT}$

Step 5: Prove that your approximate solution ≤ k optimal solution

Let W^{approx} = sum of weights of subsets included in our approximate set cover F

$$
\begin{aligned}
W^{\text{approx}} &= \sum_{S_i \in F} w_i \\
&\leq \sum_{S_i \in F} k \; x_i^* \; w_i \qquad (x_i^* \geq 1/k) \\
&\leq \sum_{i=1}^{m} k \; x_i^* w_i \\
&= k \sum_{i=1}^{m} x_i^* w_i = k \; W^{\text{LP}}
\end{aligned}
$$

Therefore $W^{\text{approx}} \leq k \; W^{\text{LP}}$

Combining with $W^{\text{LP}} \leq W^{\text{OPT}}$, we get $W^{\text{approx}} \leq k \; W^{\text{OPT}}$

Manufacturing Problem

A company manufactures four items A, B, C, and D on two machines X and Y. The time (in minutes) to manufacture one unit of each item on the two machines is shown on the right.

The profit per unit for A, B, C, and D is $10, $12, $17, and $8 respectively.

The floor space taken up by each unit of A, B, C, and D is 0.1, 0.15, 0.5, and 0.05 sq. meters respectively. Total floor space available is 50 sq. meters.

Customers require that twice as many units of B should be produced as C.

Machine X is out of action (maintenance/breakdown) for 5% of the time, and machine Y for 7% of the time.

Assuming a working week 35 hours long, formulate the problem of how to manufacture these products so as to maximize profit as a linear program. You don't need to solve it.

Item	Time taken	
	Machine X	Machine Y
A	10	29
B	12	19
C	13	33
D	8	23

Solution

1. Let x_A, x_B, x_C, and x_D be the units of A, B, C, D manufactured by machine X.
   Let y_A, y_B, y_C, and y_D be the units of A, B, C, D manufactured by machine Y.

2. Objective Function:
   Maximize $10(x_A + y_A) + 12(x_B + y_B) + 17(x_C + y_C) + 8(x_D + y_D)$

3. Constraints:
   a. Floor Space
      $0.1(x_A + y_A) + 0.15(x_B + y_B) + 0.5(x_C + y_C) + 0.05(x_D + y_D) \leq 50$

   b. Customer requirement
      $2(x_B + y_B) = x_C + y_C$

Solution

3. Constraints:
   c. Time constraint

   $10\ x_A + 12\ x_B + 13\ x_C + 8\ x_D \leq 0.95 \times 35 \times 60$

   $29\ y_A + 19\ y_B + 33\ y_C + 23\ y_D \leq 0.93 \times 35 \times 60$

   d. Non-negative units

   $x_A, x_B, x_C, x_D, y_A, y_B, y_C, y_D \geq 0$

Spaceship Radar

A set of n space stations need your help in building a radar system to track spaceships traveling between them.

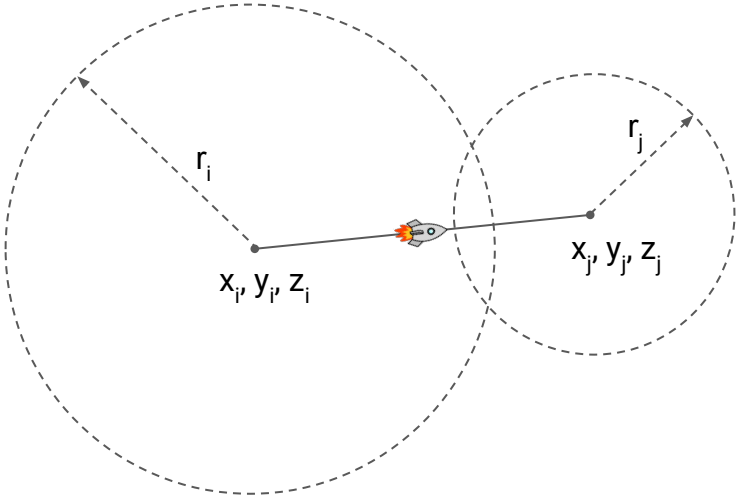
The i^{th} space station is located in 3D space at coordinates (x_i, y_i, z_i). The space stations never move. Each space station i will have a radar with positive power r_i, where r_i is to be determined.

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.

Spaceship Radar



Given x_i, y_i, z_i for all space station i, find r_i such that spheres intersect and total cost is minimized

Solution

Let d_{ij} be the distance between space stations i and j

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}$$

$$\text{minimize} \sum_{i=1}^{n} r_i$$

$$\text{subject to:}$$

$$r_i + r_j \geq d_{ij} \quad \forall\, i, j = 1 \ldots n$$

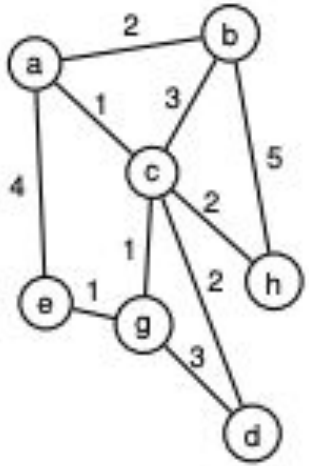
$$r_i > 0 \quad \forall\, i = 1 \ldots n$$

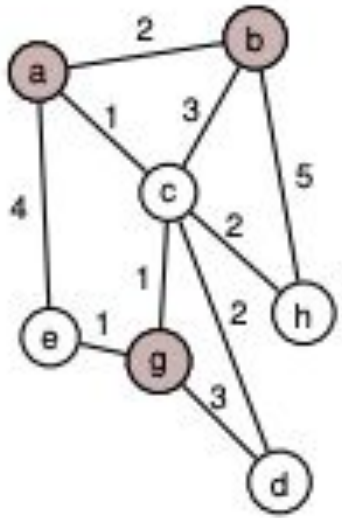
Approximation Algorithm

Review Session CSCI 570
Exam 3

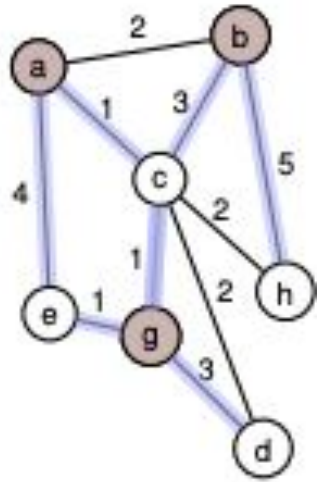
Weighted Maximum Cut Problem

For a graph G = (V, E) with edge weights w_e, the maximum cut problem is to find a cut $S \subseteq V$ such that the weight of edges across $(S, \overline{S})$ is maximized.





$S = \{a, b, g\}$



$S = \{a, b, g\}$

$w(S) = 18$

Naive Randomized Algorithm

- Randomly assign each vertex in either S or $\overline{S}$=V\S with uniformly random selection:

$P(v \in S)=P(v \in \overline{S})=1/2$

Value_Cut= $\sum w_{(u,v)}$, (u,v) is a crossing edge

Naive Randomized Algorithm

$P(v \in S)=P(v \in \overline{S})=1/2$

- The probability that an edge(u,v) is a cut edge:

Decompose probability space into atomic events

$P((u,v) \in C)=0.25+0.25=1/2$

(u,v) crosses and probs	u is in S 0.5	u is not in S 0.5
v is in S 0.5	p=0.25	p=0.25
v is not in S 0.5	p=0.25	p=0.25

Naive Randomized Algorithm

Since output is random: lower bound on expected cost of output

$$\mathbf{Output} = \sum_{\{i,j\} \in E} w_{ij} \cdot 1(\{i,j\} \text{ crosses cut})$$

$$\mathbf{E(Output)} = \sum_{\{i,j\} \in E} w_{ij} \cdot \Pr(\{i,j\} \text{ crosses cut})$$

$$\mathbf{OPT} \leq \sum_{\{i,j\} \in E} w_{ij}$$

$$\mathbf{E(Output)} = (1/2) \sum_{\{i,j\} \in E} w_{ij}$$

$$\mathbf{E(Output)} \geq (1/2)\mathbf{OPT}$$

Proves factor of 2

In maximization: SOL>=αOPT it is a factor of 2 → 2-approximation

Greedy algorithm

➢ We can switch between different cuts by moving vertices across the cut

1) We start with an arbitrary cut
2) While there are greedy moves improving the cost, perform them

Greedy moves: for a node v the total weight of edges from v to nodes in its own side of the partition exceeds the total weight of edges from v to nodes on the other side of the partition, then we move v to the other side of the partition.

$$\sum_{e \in \delta(v) \cap (S \times S)} w(e) > \sum_{e \in \delta(v) \cap (S \times (V \setminus S))} w(e)$$

Greedy algorithm

- According to the given algorithm at optimality:

$$\sum_{e \in \delta(v) \cap (S \times S)} w(e) \leq \sum_{e \in \delta(v) \cap (S \times (V \setminus S))} w(e) \qquad \forall v \in S$$

$$\sum_{e \in \delta(v) \cap (S \times S)} w(e) + \sum_{e \in \delta(v) \cap (S \times (V \setminus S))} w(e) \leq \sum_{e \in \delta(v) \cap (S \times (V \setminus S))} w(e) + \sum_{e \in \delta(v) \cap (S \times (V \setminus S))} w(e)$$

$$\sum_{e \in \delta(v)} w(e) \leq 2 \sum_{e \in \delta(v) \cap C} w(e)$$

$$\sum_{v} \sum_{e \in \delta(v) \cap C} w(e) \geq \frac{1}{2} \sum_{v} \sum_{e \in \delta(v)} w(e)$$

Greedy algorithm

$$\sum_{v} \sum_{e \in \delta(v) \cap C} w(e) \geq \frac{1}{2} \sum_{v} \sum_{e \in \delta(v)} w(e)$$

Each edge is considered two times by its two ends

$$2w(C) \geq \frac{1}{2} \left(2 \cdot \sum_{e} w(e) \right)$$

OPT≤∑ w(e)

$$2w(C) \geq \sum_{e} w(e) \geq \mathrm{OPT}$$

2-approximation

k-center Problem

- A large amount of data, some of them are similar/dissimilar
- Group similar data together in a certain number of groups
- **Goal:** select some data points among all our data to be the cluster centers so that we can match each data to the nearest cluster center

Distance function $d : V \times V \to \mathbb{R}_{\geq 0}$

(1) Positive semidefiniteness: $d(x, y) \geq 0$ for all $x, y \in V$ and $d(x, y) = 0$ if and only if $x = y$.

(2) Symmetric: $d(x, y) = d(y, x)$.

(3) Triangle inequality: $d(x, y) \leq d(x, z) + d(z, y)$.

Goal: find a set S of k vertices, such that the maximum distance of any vertex to its cluster center is minimized

k-center Problem

Goal: find a set S of k vertices, such that the maximum distance of any vertex to its cluster center is minimized

$$d(i, S) = \min_{s \in S} d(i, s)$$

Radius of S

$$r = \max_{i \in V} d(i, S)$$

$$\min_{S \subseteq V : |S| = k} \max_{i \in V} d(i, S)$$

Greedy approach

first pick a vertex s ∈ V arbitrarily and put it in our set S of cluster centers

next cluster center to be as far away as possible from all the other cluster centers

Pick arbitrary $s \in V$ and initialize $S = \{s\}$. Do while $|S| < k$:

1. $s \leftarrow \text{argmax}_{s \in V} d(s, S)$
2. Update $S \leftarrow S \cup \{s\}$

Greedy algorithm produces a **2-approximation** algorithm for the k-clustering problem

2-approximation

$S^* = \{s_1, \ldots, s_k\}$ optimal solution

r^* Radius of the optimal solution

$V_1^*, \ldots, V_k^*$, Optimal solution divides V vertices into clusters

$i \in V$ is placed in V_ℓ^* if it is closest to s_ℓ among all of the points in S^*

2-approximation

points i and j in the same cluster $d(i,j) \leq d(i,s_\ell) + d(j,s_\ell) \leq r^* + r^* = 2r^*$

consider S ⊆ V set of points selected by the greedy algorithm

the first iteration where the algorithm selects a point $i \in V_\ell^*$ to S

algorithm had already selected a point $i' \in V_\ell^*$ and added it to S

The greedy algorithm always selects points furthest away from the current set of points in S, every other point j ∈ V where j has not been added to S must have distance bounded by: $d(j,S) \leq d(i',i)$

$$d(i',i) \leq 2r^* \quad d(i,S) \leq 2r^* \quad r \leq 2 \cdot r^*$$