CSCI 570 Exam 3 Review Slides

- NP Completeness
- Linear Programming
- Approximation Algorithm
- HW 11 Review

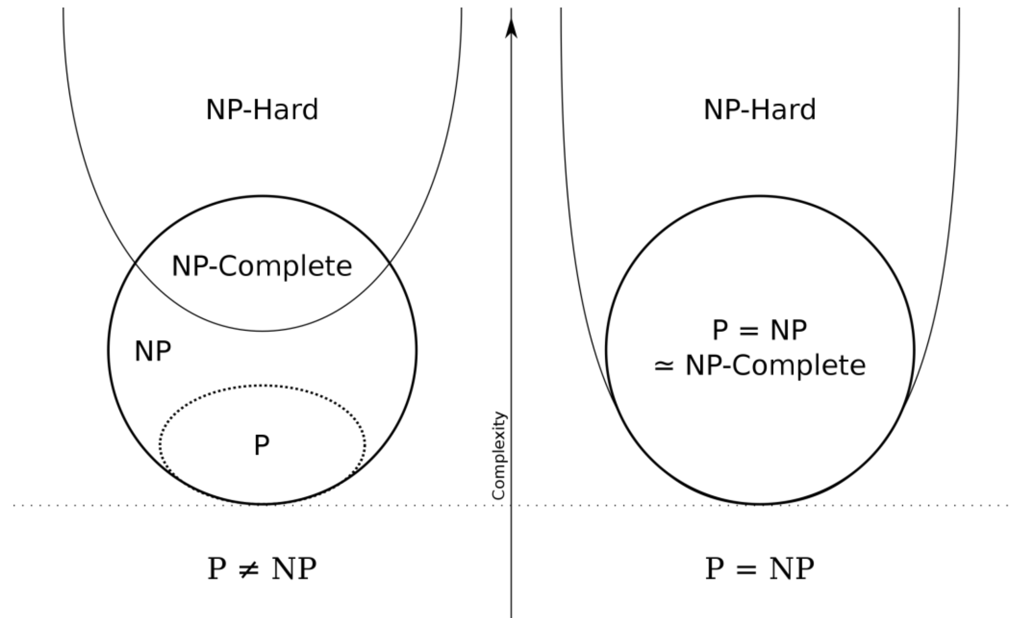
NP Completeness

Jun Yan

2022/11

Key Concepts

- Decision Problems vs Optimization Problems (yes/no vs finding the min/max)
- P: Decision problems deterministically *solvable in polynomial time*
- NP: Decision problems deterministically *verifiable in polynomial time*
- Polynomial-time reducible
  - $Y \leq_P X$: *Arbitrary instances of problem Y can be solved using a polynomial number of standard computational steps, plus a polynomial number of calls to a black box that solves problem X.*
- NP-Hard
- NP-Complete



How to Prove NP-Completeness

Given a problem X, strategies for proving it's NP-complete:

- Prove that $X \in NP$.
  - Define a polynomial-length **certificate** and a polynomial-time **verifier**.
- Choose a problem Y that is known to be NP-complete.
- Prove that $Y \leq_P X$.
  - Define a polynomial algorithm to construct an X-instance x based on a Y-instance y.
  - Prove: x is satisfiable for X $\Rightarrow$ y is satisfiable for Y
  - Prove: y is satisfiable for Y $\Rightarrow$ x is satisfiable for X

Known NPC Problems

- 3-SAT
- Independent Set
- Vertex Cover
- Set Cover
- Set Packing
- Hamiltonian Cycle, Hamiltonian Path
- TSP
- 0-1 Knapsack
- Subset Sum

True/False Question #1

- If INDEPENDENT SET can be reduced to a decision 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.

True/False Question #2

- 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.

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 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

Plan:

- Can reduce from IND-SET

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

- Independent Set: A set of vertices such that no two vertices are adjacent to each other.

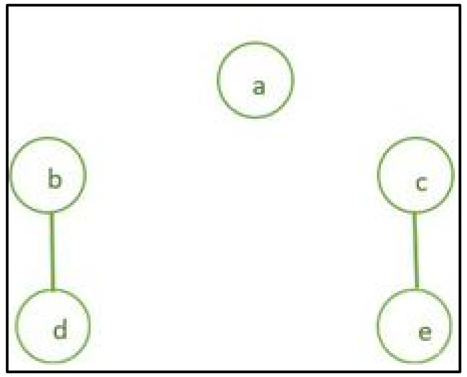
The CLIQUE Problem: Given an undirected graph G = (V, E), and a positive integer k, decide whether the graph G contains a clique of size at least k.

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.

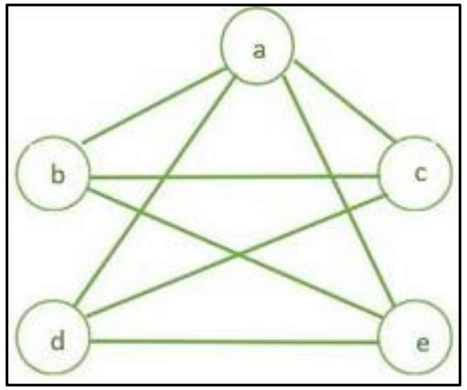
b) IND-SET to CLIQUE

Construction: Create an instance of the CLIQUE problem <G_c, k> based on <G, k> for the IND-SET problem - G_C being the complement of G (edges reversed)



G

{a,b,c} is an independent set in G



G_C

{a,b,c} is a clique in G_c

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) IND-SET to CLIQUE

Construction: Create an instance of the CLIQUE problem <G_c, k> based on <G, k> for the IND-SET problem - G_C 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 G_C → i.e. the same set of vertices form a clique in G_C → proving that G_C has a clique of size k.

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

Long Question #2

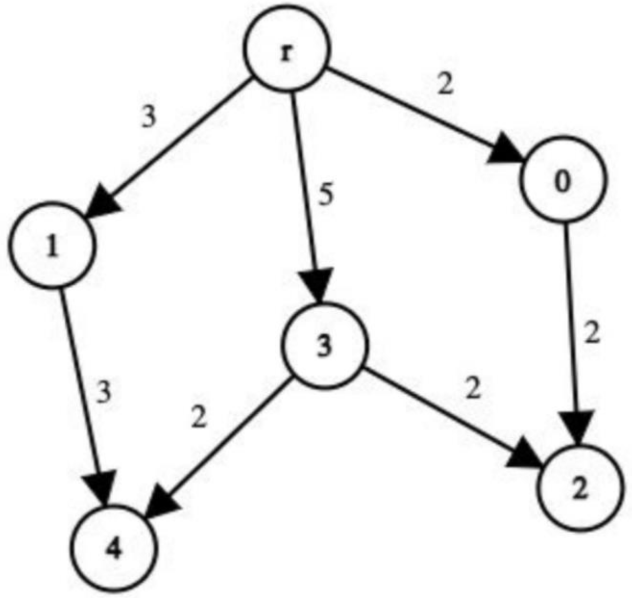
Suppose you move to a new city. The city is defined by a directed graph $G=(V,E)$ and each edge $e \in E$ has a non-negative cost c_e associated to it. Your living place is represented as a node $s \in V$.

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 3-SAT.

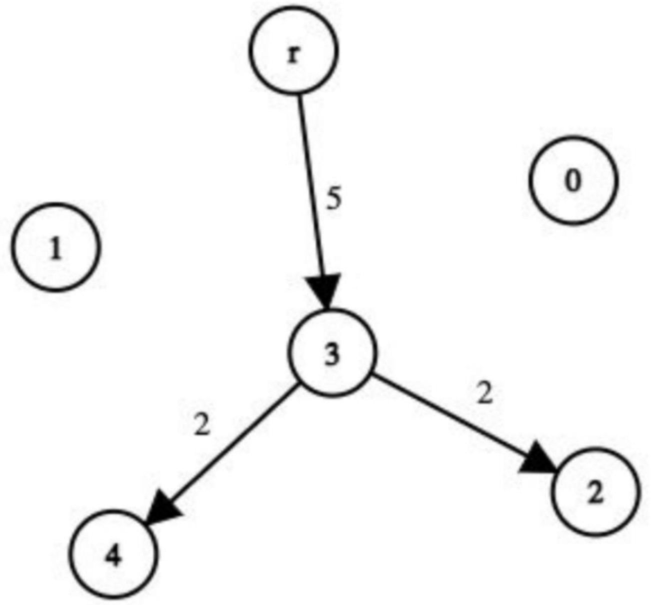
Example

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 ∈ E has a non-negative cost c_e associated to it. Your living place is represented as a node s ∈ 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 ∈ 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 ∈ V, and a number k, <u>does there exist a subgraph G'=(V',E') that contains a path from s to each x ∈ X, having a total edge cost of E' at most k</u>? Show that ROAM is NP-complete with a reduction from 3-SAT.

a) Show that ROAM is in NP

Solution: We show that a subgraph G' as a certificate. Verifier checks:

- all the nodes in X are reachable - can be done using DFS, thus in poly-time;
- 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 ∈ E has a non-negative cost c_e associated to it. Your living place is represented as a node s ∈ 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 ∈ 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 ∈ V, and a number k, does there exist a subgraph G'=(V',E') that contains a path from s to each x ∈ X, having a total edge cost of E' at most k? Show that ROAM is NP-complete with a reduction from 3-SAT.

3-SAT: determining the satisfiability of a formula in conjunctive normal form where each clause has 3 literals.

Hints for construction

1) "Satisfy ALL clauses" (3-SAT) VS "Reach ALL nodes in X" (ROAM)

2) "Assigning values to variables" (3-SAT) VS "Picking edges to construct the path from r to a node in X"

(b) Reduction from 3-SAT: Construct

A new node r for living place.

Nodes x_i and ~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 & ~x_i.

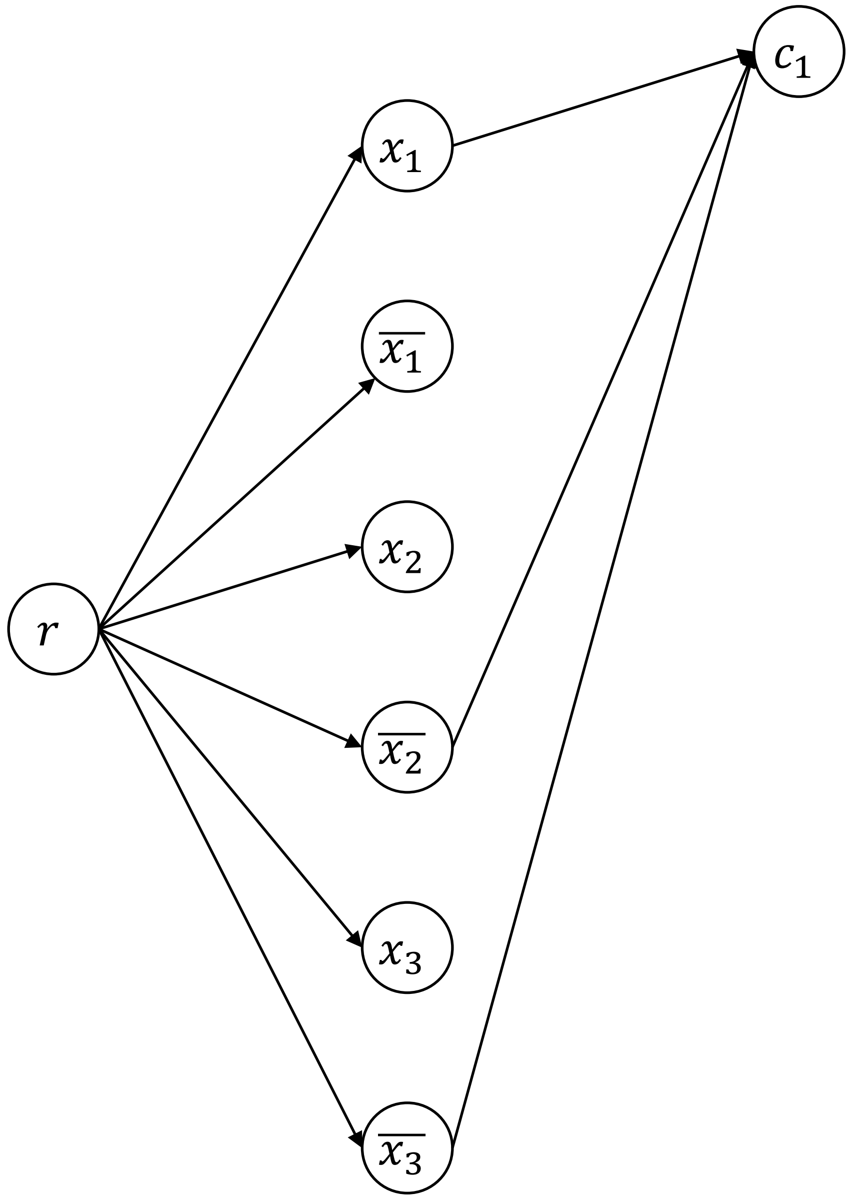
All edges have weight 1.

Subset X contains all u_i and c_i.

(c) Claim: The 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.

(n = #vars, m = #clauses)

Clause: $x_1 \vee \overline{x_2} \vee \overline{x_3}$



(b) Reduction from 3-SAT: Construct

A new node r for living place.

Nodes x_i and ~x_i for each literal, all connected to r.

Nodes c_i for each clause, connected to its literals.

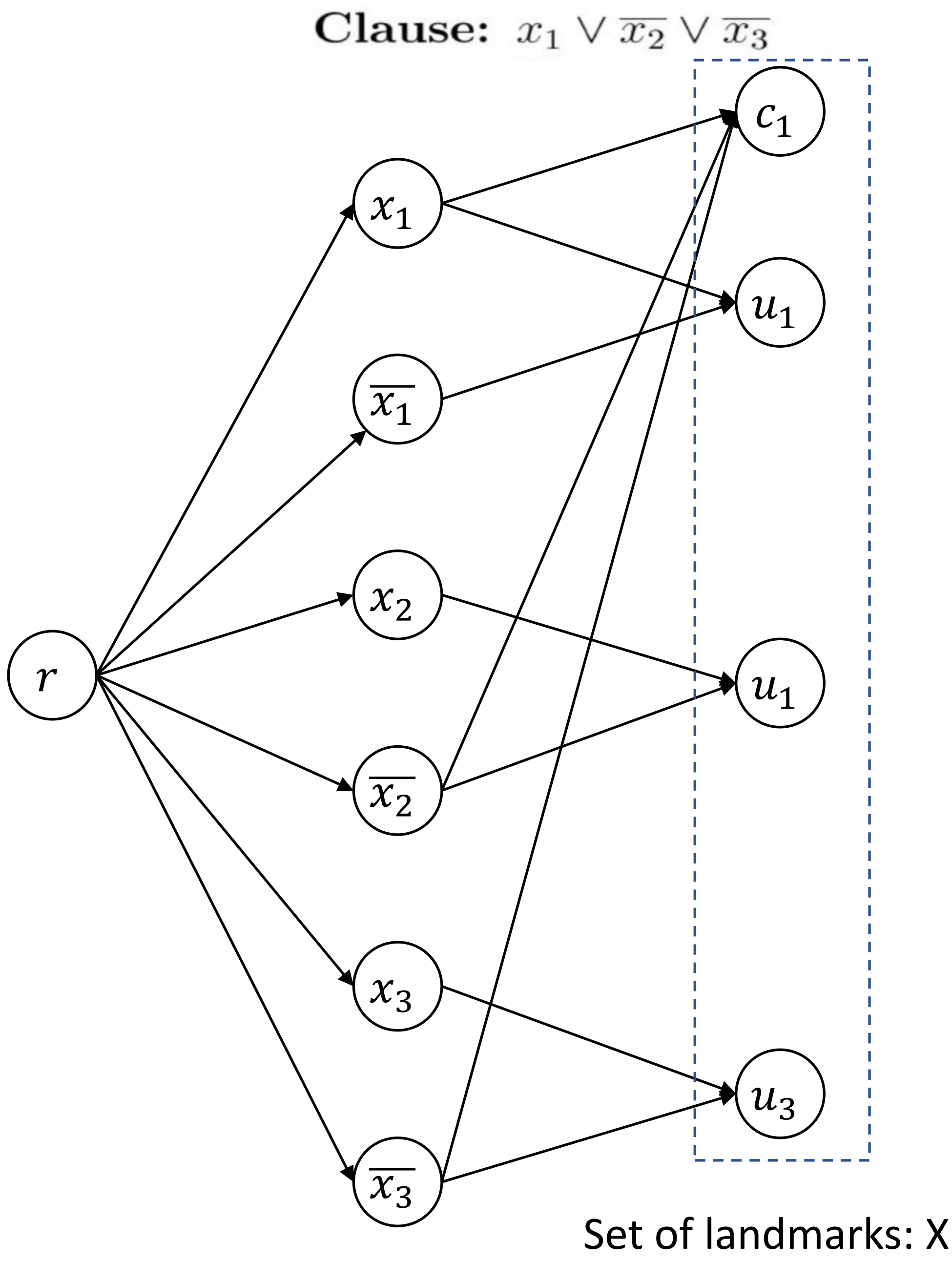
Nodes u_i for each variable, connected to x_i & ~x_i.

All edges have weight 1.

Subset X contains all u_i and c_i.

(c) Claim: The 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.

(n = #vars, m = #clauses)



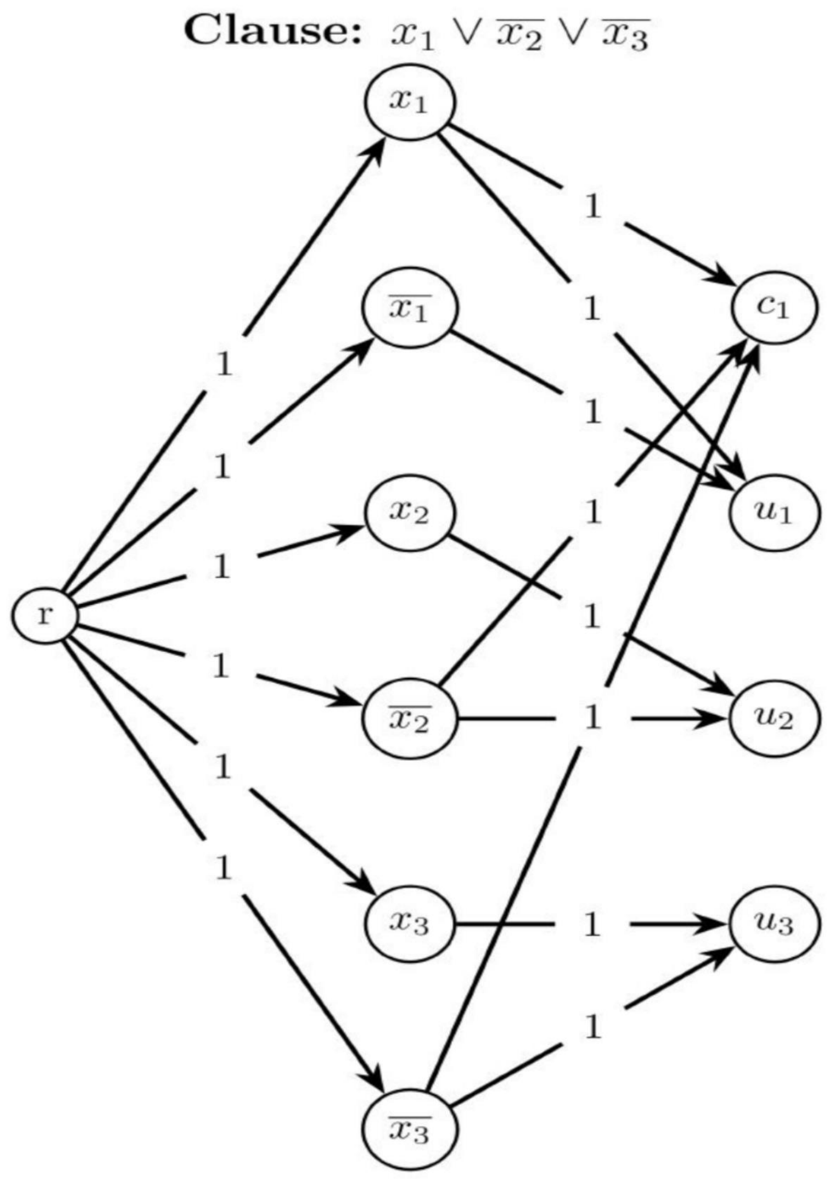
(d) 3-SAT has a satisfying assignment => Constructed graph has a subgraph with routes from r to each x in X with weight at most 2n + m

Proof: Consider the satisfying assignment. For any x_i, let L_i be the literal that is true (i.e. x_i or ~x_i).

Construct the subgraph by including

1) The nodes corresponding to each L_i (along with 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 => 3-SAT has a satisfying assignment

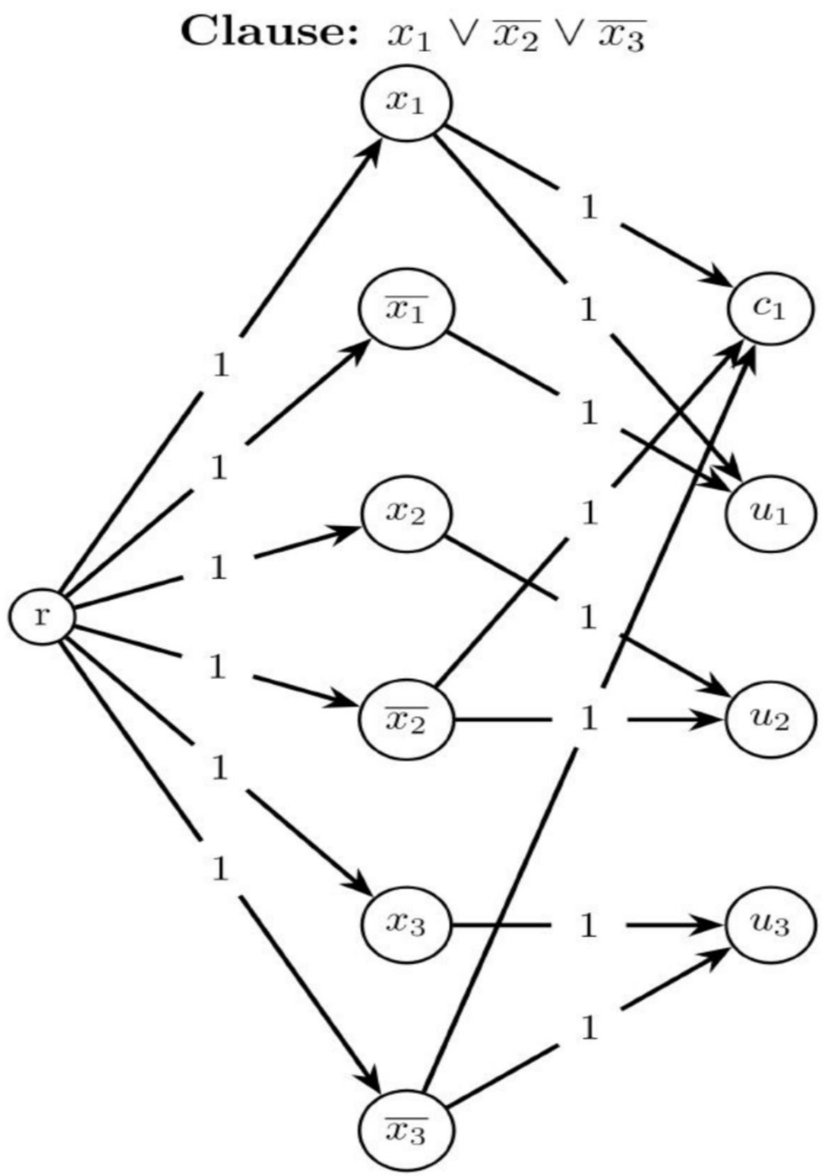
Proof: Consider the subgraph H that spans X.

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.



Long Question #3

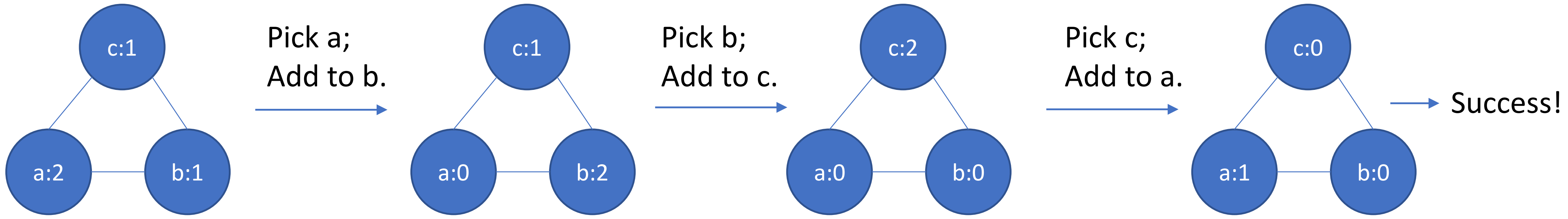
Given an undirected connected graph G = (V, E) in which a certain number of tokens t(v)≥1 placed on each vertex v. You will now play the following game.

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 picks such that you are left with exactly one token in the whole graph.

Prove that the problem of deciding the existence of such a sequence of moves is NP-hard.

(The size of the certificate --- a sequence of picks --- is polynomial to the input (token) value rather than the input size. Therefore, the problem is not in NP.)*



Given an undirected connected graph G = (V, E) in which a certain number of tokens t(v)≥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 picks 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 deciding the existence of such a sequence of moves is NP-hard.

Hints for construction:

1) "Sequence of picks" VS "path in a graph: a sequence of nodes".

2) Can leverage token assignment to ensure that each node is visited exactly once.

Hamiltonian path: a path that visits each vertex in the graph exactly once.

Given an undirected connected graph G = (V, E) in which a certain number of tokens t(v)≥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 picks 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 deciding the existence of such a sequence of moves is NP-hard.

Reduction from Hamiltonian path:

For an instance Hamiltonian path with G, we construct |V| instances of the above problem with the same graph G. In each instance, we set one node as the starting node. We place 2 tokens on the starting node and 1 token on all the remaining nodes. The |V| instances have |V| distinct starting nodes, covering all nodes in G.

Claim: Any of the |V| instances of the above problem has a solution if and only if here is a Hamiltonian path in G.

Given an undirected connected graph G = (V, E) in which a certain number of tokens t(v)≥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 picks 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 deciding the existence of such a sequence of moves is NP-hard.

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 x_1, x_2, ... x_n. Then, when we use the above problem considering x_1 as a starting point, we can go through the list of nodes in the Hamiltonian path. We know that x_1 has two tokens and rest of the nodes have 1 token. When we consider x_1 as a starting point, we delete its two tokens and add a token to x_2, so now x_2 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 neighbor 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 picks 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-hard.

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 x_1, x_2, ... x_n, when we consider the problem with only x_1 having two tokens. We can see that the only possible starting point is x_1 as there is no other node with two tokens. Also, when we push a token to node x_2, our next move needs to be removing two tokens from x_2, 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, x_1, x_2, ... x_n 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 x_1, x_2, ... x_n has visited each node exactly once. Since we can push token only to a neighbor node, the sequence x_1, x_2, ..., x_n will be a valid path and hence, will be a valid Hamiltonian path.

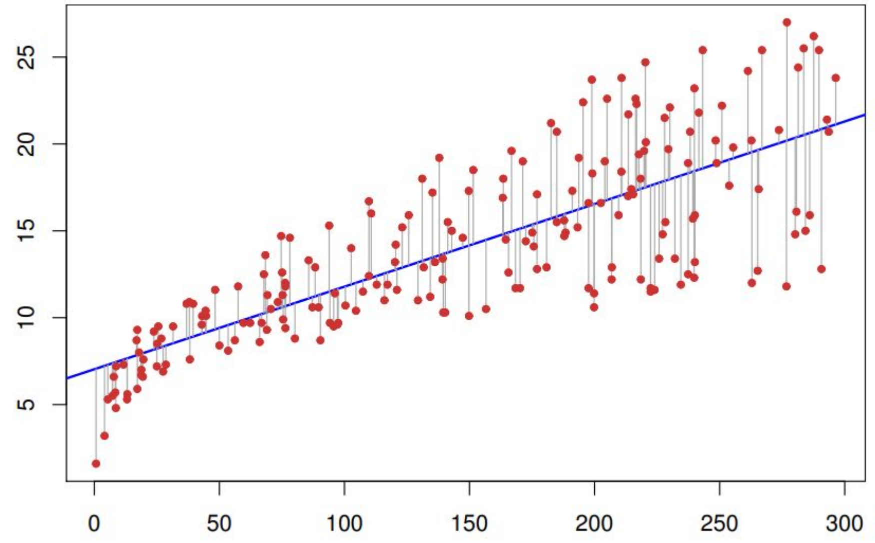
THANK YOU

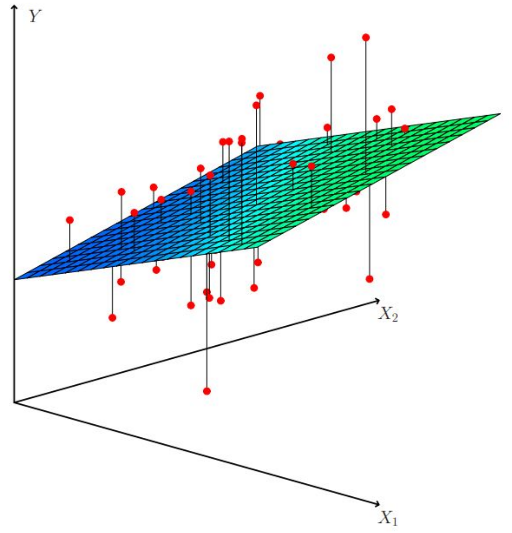
Linear Programming

Ta-Yang Wang

Question 1 – Linear Regression

Given points $(x_1, y_1), \dots, (x_n, y_n)$, find a line with equation $y = ax + b$ minimizing the expression $\sum_{i=1}^{n} |ax_i + b - y_i|$. Formulate this nonlinear optimization problem as a linear program.





Solution 1 – Linear Regression

- Introduce auxiliary variable c_i for the error at the ith point

$$|ax_i + b - y_i| \le c_i \Rightarrow ax_i + b - y_i \le c_i \text{ and} -(ax_i + b - y_i) \le c_i$$

$$\begin{aligned} \text{Minimize} \quad & c_1 + \cdots + c_n \\ \text{subject to} \quad & ax_i + b - y_i \le c_i \text{ for } i = 1, \dots, n \\ & -(ax_i + b - y_i) \le c_i \text{ for } i = 1, \dots, n \end{aligned}$$

Question 2 – MAX TFSAT

A variation of the satisfiability problem is the MAX True-False SAT, or MAX TFSAT for short. Given n Boolean variables $x_1, \ldots, x_n$, and m clauses $c_1, \ldots, c_m$, each of which involves two of the variables. We are guaranteed that each clause is of the form $x_i \wedge \overline{x_j}$. Formulate an integer linear program to maximize the number of satisfied clauses.

Solution 2 – MAX TFSAT (1/2)

Objective

- Maximize the number of satisfied clauses

Constraints

- Each clause is of the form $x_i \wedge \overline{x_j}$
  - $x_i \wedge \overline{x_j}$ is satisfied if and only if x_i is set to TRUE and x_j is set to FALSE

Variables

- y_i: Binary variable, 1 if x_i is set to TRUE
- z_k: Binary variable, 1 if c_k is satisfied

Solution 2 – MAX TFSAT (2/2)

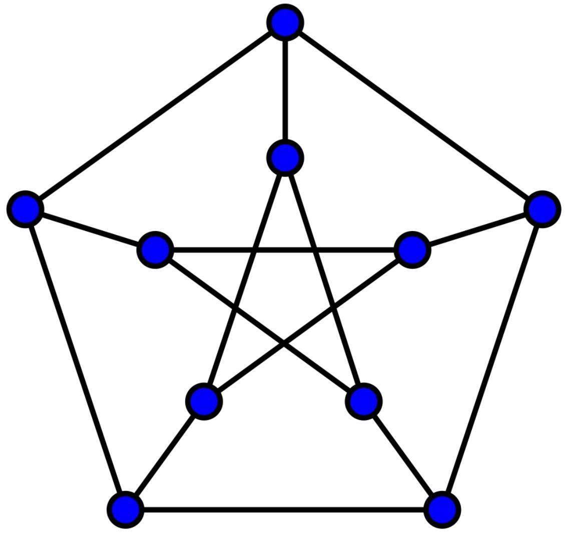
$$
\begin{aligned}
\text{Maximize} \quad & z_1 + \cdots + z_m \\
\text{subject to} \quad & z_k \le y_i, \text{for } c_k = x_i \wedge \overline{x_j} \\
& z_k \le 1 - y_j, \text{for } c_k = x_i \wedge \overline{x_j} \\
& y_i \in \{0, 1\}, \text{for } i = 1, \dots, n.
\end{aligned}
$$

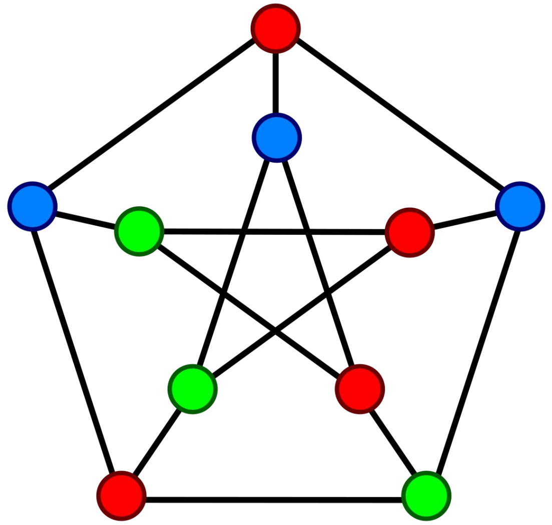
Remarks

- A random assignment achieves a 0.25-approximation
- Randomized rounding for LP relaxation yields a 0.5-approximation
- Semi-definite programming can do even better: 0.79 – CSCI 672

Question 3 – Graph Coloring

Given an undirected graph $G = (V, E)$. The goal is to color the nodes of the graph with minimum number of colors, such that adjacent nodes have different colors. Formulate this problem as an integer linear program.





Solution 3 – Graph Coloring (1/5)

Objective

- Color nodes using as few different colors as possible

Constraints

- Every node is assigned exactly one color
- No two adjacent nodes can share the same color

Variables

- c_i: Binary variable, 1 if at least one node is assigned color i
- $x_{v,i}$: Binary variable, 1 if node v is assigned to color i

Solution 3 – Graph Coloring (2/5)

Objective

- Use as few different colors as possible
  - n colors are enough

$$\text{Minimize } c_1 + \cdots + c_n$$

Solution 3 – Graph Coloring (3/5)

Constraints

- Every node is assigned exactly one color
    - Sum over all colors for a single node is equal to one

$$x_{v,1} + \cdots + x_{v,n} = 1, \text{for } v \in V$$

Solution 3 – Graph Coloring (4/5)

Constraints

- No two adjacent nodes can share the same color
  - Given a color and a pair of adjacent nodes, at most one of them may have that color assigned

$$x_{u,i} + x_{v,i} \leq 1, \text{for } (u, v) \in E, i = 1, \ldots, n$$

  - Make sure that if any node is colored with color i then c_i is counted

$$x_{v,i} \leq c_i, \text{for } v \in V, i = 1, \ldots, n$$

Solution 3 – Graph Coloring (5/5)

$$\begin{aligned}
\text{Minimize} \quad & c_1 + \cdots + c_m \\
\text{subject to} \quad & x_{v,1} + \cdots + x_{v,n} = 1, \text{for } v \in V \\
& x_{u,i} + x_{v,i} \leq 1, \text{for } (u,v) \in E, i = 1, \ldots, n \\
& x_{v,i} \leq c_i, \text{for } v \in V, i = 1, \ldots, n \\
& x_{v,i}, c_i \in \{0,1\}, \text{for } v \in V, i = 1, \ldots, n.
\end{aligned}$$

Remark

- If G has no isolated nodes, the color conflict can be formulated as

$$x_{u,i} + x_{v,i} \leq c_i, \text{for } (u,v) \in E, i = 1, \ldots, n.$$

THANK YOU

Approximation Algorithm

Jessica Dsouza

Why Approximation Algorithms?

- ➢ Problems that we cannot find an optimal solution in a polynomial time

  Eg: Set Cover, Bin Packing

- ➢ Need to find a near-optimal solution:
  - ❖ Heuristic
  - ❖ Approximation algorithms:

    This gives us a guaranteed approximation ratio

k-Center Problem

➢ Input: Set of n sites $s_1, \ldots, s_n$ and an integer k

➢ Output: Return a set C of k centers s.t. the maximum distance of any site from its nearest center is minimized

Minimize $r(C) = \max_{i \in \{1,\ldots,n\}} d(s_i, C)$,

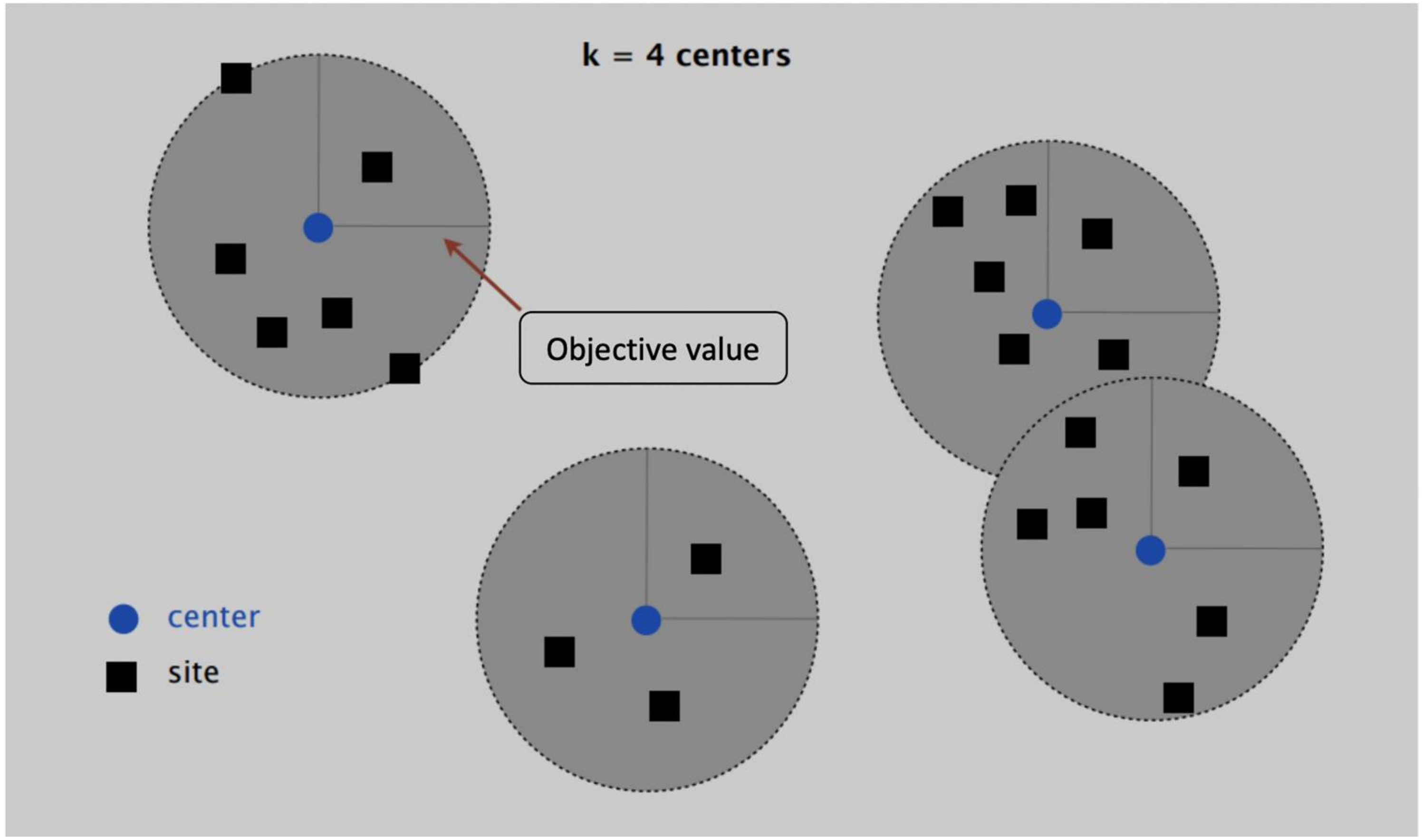
where $d(s_i, C) = \min_{c \in C} d(s_i, c)$

Given C, note that $r(C)$ is the minimum radius r such that if we draw a ball of radius r around every center in C, then the balls collectively cover all the sites

Constraints

- Sites are points in some metric space with distance d satisfying:
  - Identity: $d(x, x) = 0$ for all x
  - Symmetry: $d(x, y) = d(y, x)$ for all x, y
  - Triangle inequality: $d(x, z) \leq d(x, y) + d(y, z)$ for all x, y, z

Example



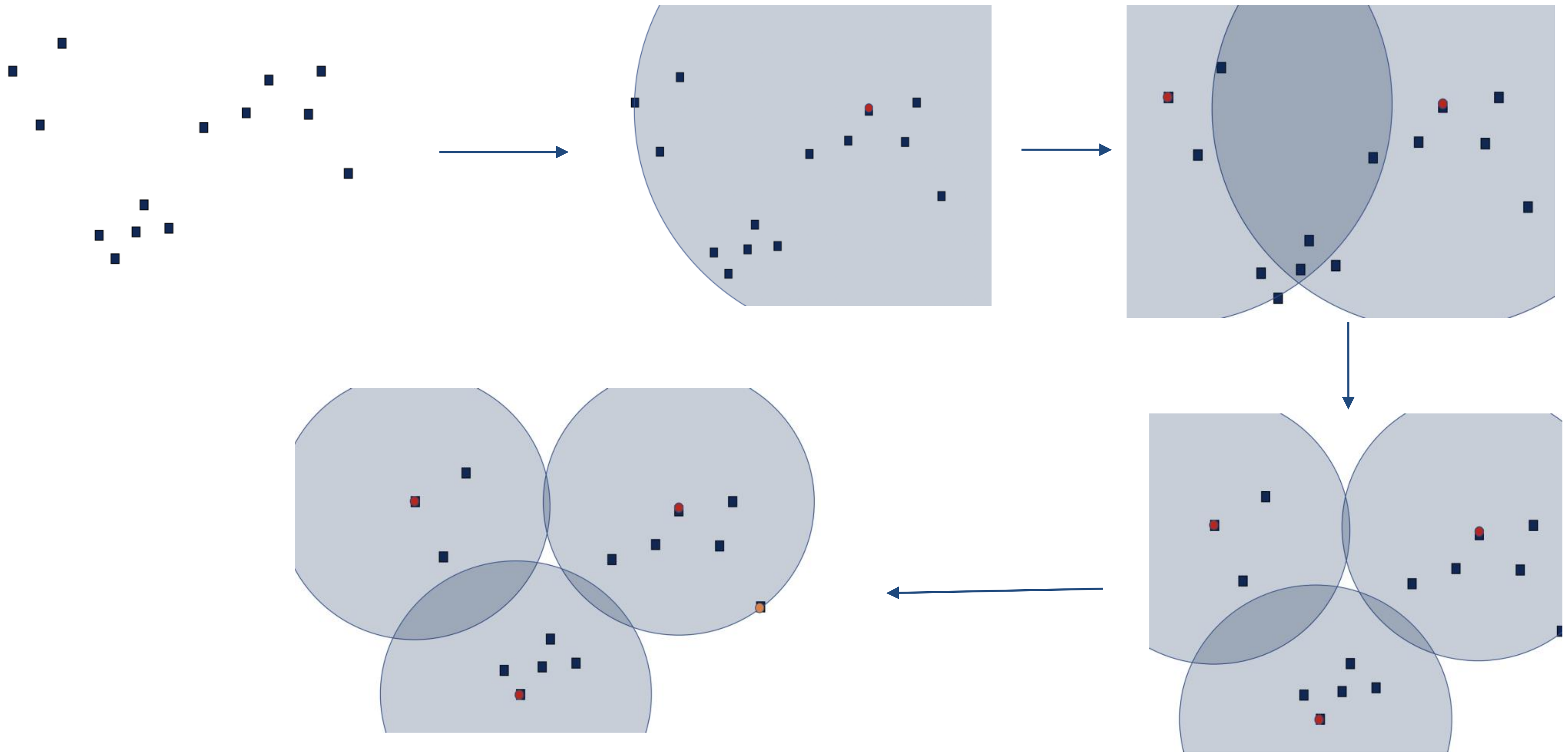
What would be a good greedy algorithm for this problem?

➢ Put the first center at an arbitrary site

➢ Put every next center at a site whose distance to its nearest center is maximum among all sites

```
➢ $C_1 \leftarrow s_1$          (arbitrary site works)
➢ For $j = 2, \dots, k$:
    ➢ $s_i \leftarrow \underset{s}{\operatorname{argmax}}\, d(s, C_{j-1})$; $\Delta_j = d\left(s_i, C_{j-1}\right)$
    ➢ $C_j \leftarrow C_{j-1} \cup \{s_i\}$
➢ Return $C_k$
```

How does the Algorithm work?



Why is this algorithm a 2-approximation?

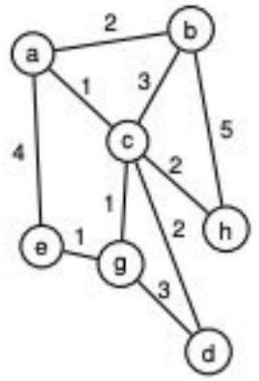
Theorem: If C^* is the optimal set of k centers, then $r(C_k) \leq 2\, r\,(C^*)$

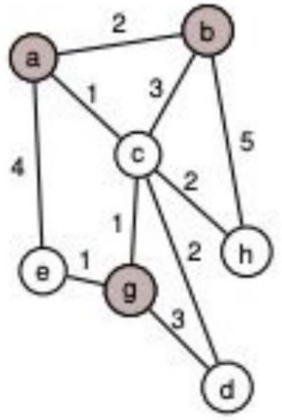
Proof:

- ➢ Draw a ball of radius $r\,(C^*)$ from each center in C^*
- ➢ By pigeonhole principle, at least two $s_i\,,\, s_j \in C_{k+1}$ must belong to the same ball (say centered at $c^* \in C^*$)
  - Hence, $d(s_i\,,\, c^*)\,,\, d(s_j\,,\, c^*) \leq r\,(C^*)$
- ➢ But by our claim: $r(C_k) \leq d(s_i\,,\, s_j) \leq d(s_i\,,\, c^*) * + d(s_j\,,\, c^*) \leq 2 * r\,(C^*)$
- ➢ Done!

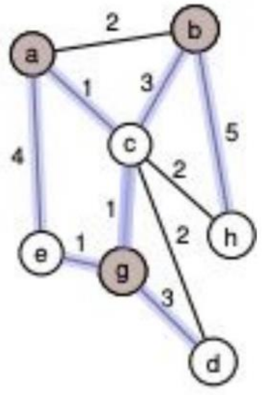
Maximum Cut Problem

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





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



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

$w(S) = 18$

Greedy Algorithm

These greedy moves give us a simple optimization procedure:

1. Begin with an arbitrary cut (e.g. S = Ø).

2. If for a node v the total weight of edges from v to nodes in its current 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.

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

Why is this a 2-approximation?

To analyze the approximation ratio, observe that at optimality:

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

and similarly for $v \in V \setminus S$. We can find an equivalent form by adding the weight of cut edges to both sides:

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

Why is this a 2-approximation?

Simplifying:

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

If we sum over all vertices:

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

The left hand side is exactly twice the value of the cut, while the right hand side (sum of degree cuts) counts every edge twice.

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

Since OPT uses a subset of edges:

$$2w(C) \ge \sum w(e) \ge \mathrm{OPT}$$

HW 11 review

Parth Goel

Zero-Weight-Cycle Problem

You are given a directed graph $G=(V,E)$ with weights on its edges $e \in E$. 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 (Step 1)

Certificate - A list of edges in E that form a simple cycle in G.

Certifier -

- Check if the edges are valid
- Check if the cycle is a valid simple cycle
- Check that the sum of weights on those edges is 0

Solution

2 - Choosing Subset sum

3 - Proving that the given problem is NP-Hard by reducing Subset Sum to Zero-Weight-Cycle Problem

We are given the number $w_1, ..., w_n$, and we want to know if there is a subset that adds up to exactly W. We construct an instance of the Zero-weight-cycle in which the graph has nodes 0, 1, 2, ..., n, n+1 and an edge (i, j) for all pairs $i < j$. The weight of the edge (i, j) is equal to w_j. Finally, there is an edge (n+1, 0) of weight $-W$. All edges to n+1 have capacity 0.

Claim and proof

We claim that there is a subset that adds up to exactly W if and only if G has a zero-weight-cycle.

Forward proof: If there is such a subset S, then we define a cycle that starts at 0, goes through the nodes whose indices are in S, and then returns to 0 on the edge $(n+1, 0)$. The weight of $-W$ on the edge $(n+1, 0)$ precisely cancels the sum of the other edge weights.

Backward proof: Conversely, all cycles in G must use the edge $(n+1, 0)$, and so if there is a zero-weight-cycle, then the other edges must exactly cancel $-W$, in other words, their indices must form a set that adds up to exactly W which gives us the subset by mapping the nodes in the cycle to the indices.

Zero-Weight-Cycle Problem

Given a graph $G = (V, E)$ with an even number of vertices as the input, the HALF-IS problem is to decide if G has an independent set of size $|V|/2$. Prove that HALF-IS is in NP-Complete.

Solution (Step 1)

Certificate - A list of nodes in V.

Certifier -

- Check if the nodes are valid such that there are no edges between them
- Check if the no. of nodes selected is equal to |V| / 2

Solution

2 - Choosing Independent Set

3 - Proving that the given problem is NP-Hard by reducing IS to HALF-IS

Consider an instance of IS, which asks for an independent set $A \subset V$, $|A| = k$, for a graph $G(V, E)$, such that vertices in A disconnected from each other:

- If $k = |V|/2$, IS reduces to HALF-IS.
- If $k < |V|/2$, then add m new disconnected nodes such that $k + m = (|V| + m)/2$, i.e., $m = |V| - 2k$. Note that the modified set of nodes V' has an even number of nodes. Since the additional nodes are all disconnected from each other, they form a subset of the independent set. Therefore, the new graph $G'(V', E')$ where $E' = E$ has an independent-set of size $|V'|/2$ if and only if $G(V, E)$ has an independent set of size k.
- If $k > |V|/2$, then again add $m = 2k - |V|$ new nodes to form the modified set of nodes V'. Connect these new nodes to all the other $|V| + m-1$ nodes. Since these m new nodes are connected to every other, none of them should belong to an independent set.Therefore, the new graph $G'(V', E')$ has an independent-set of size $|V'|/2$ if and only if $G(V, E)$ has an independent set of size k.

Claim and proof

We claim that there is an IS of size k iff there is a independent set of size |V’|/2 in the corresponding HALF-IS graph G’(V’,E’)

We can get the forward and backward proof for each of the 3 cases from the previous explanation.

Min-3-SAT

Suppose we have a variation on the 3-SAT problem called Min-3-SAT, where the literals are never negated. Of course, in this case it is possible to satisfy all clauses by simply setting all literals to true. But, we are additionally given a number k, and are asked to determine whether we can satisfy all clauses while setting at most k literals to be true. Prove that Min-3-SAT is NP-Complete.

Solution (Step 1)

Certificate - A list of literal that are set to be true

Certifier -

- Size of the list <= k
- All the clauses are satisfied

Solution

2 - Choosing Vertex Cover

3 - Proving that the given problem is NP-Hard by reducing Vertex Cover to Min-3SAT

For any given instance of the vertex cover problem, we can construct an equivalent Min-3-SAT problem with variables for each vertex of a graph. Each edge (u, v) of the graph can be represented by a clause $(u \lor u \lor v)$ or $(u \lor v \lor v)$ which can be satisfied only by including either u or v among the true variables of the solution.

Claim and proof

We claim that For the constructed Min-3-SAT problem, there is a satisfying assignment within k true variables if and only if there is a vertex cover within k vertices to the corresponding vertex cover problem.

Forward proof: If there is a satisfying assignment of k variables to true, we have a corresponding solution to the vertex cover problem by selecting those vertices in the vertex cover set.

Backward proof: Conversely, if we have k vertices corresponding to the vertex cover problem then we can assign true values to exactly k variables and solve the Min-3-SAT problem.