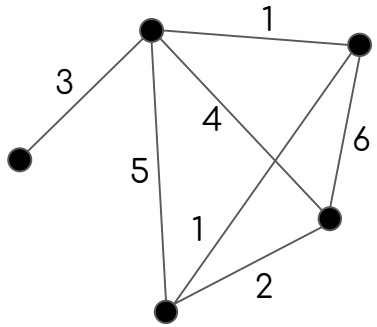
Exam 3 Review

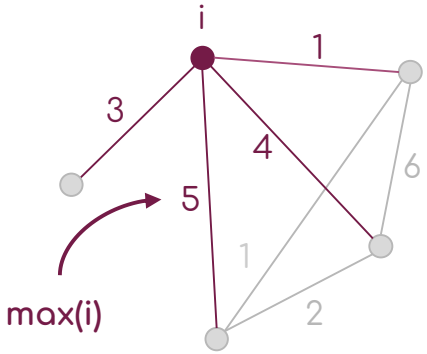
Approximation

Aida Davani

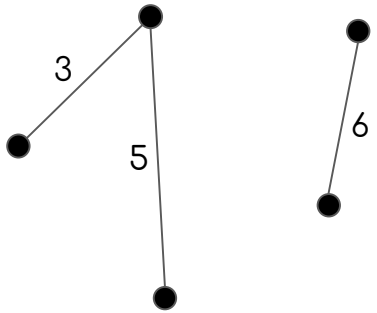
Approximating a Maximum Spanning Tree Problem

Let $G = (V, E)$ be an undirected graph with distinct edge weights $w(u, v)$ on each edge $(u, v) \in E$. For each vertex $v \in V$, let $\max(v) = \max_{(u,v)\in E} \{w(u, v)\}$ be the maximum-weight edge incident on that vertex. Let $S_G = \{\max(v) : v \in V\}$ be the set of maximum-weight edges incident on each vertex, and let T_G be the maximum-weight spanning tree of G, that is, the spanning tree of maximum total weight. For any subset of edges $E' \subseteq E$, define $w(E') = \sum_{(u,v)\in E'} w(u, v)$.

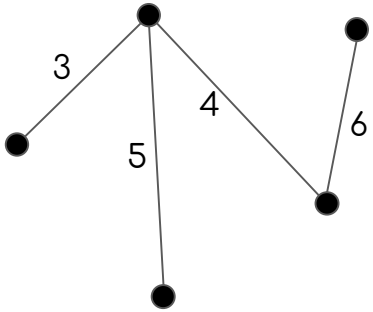




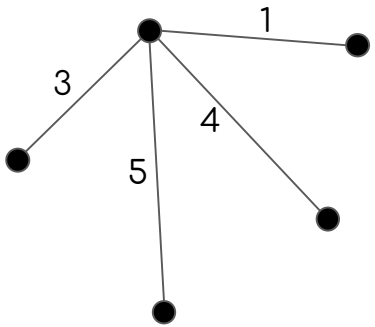
S(G)



T(G)

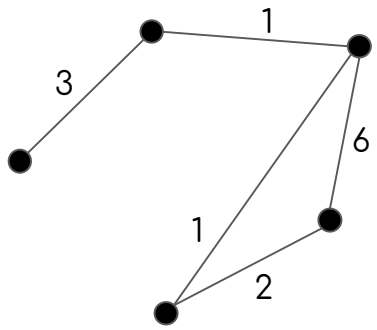


a. Give an example of a graph with at least 4 vertices for which $S_G = T_G$.

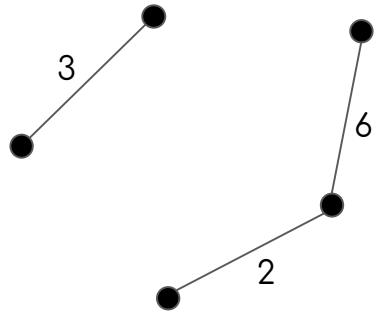


b. Give an example of a graph with at least 4 vertices for which $S_G \neq T_G$.

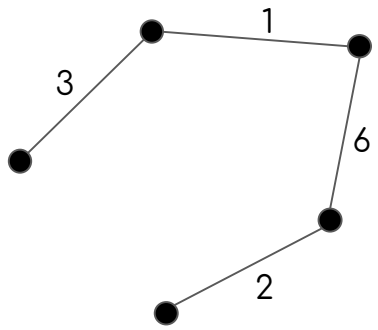
G



S(G)



T(G)



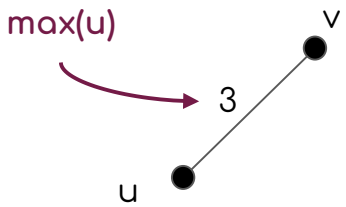
c. Prove that $S_G \subseteq T_G$ for any graph G.

c. Prove that $S_G \subseteq T_G$ for any graph G.

Consider Kruskal's algorithm:
Sort the edges from largest to smallest weight,
Consider them one at a time in order, and if the edge doesn't introduce a cycle, add it to the tree.

Proof by Contradiction

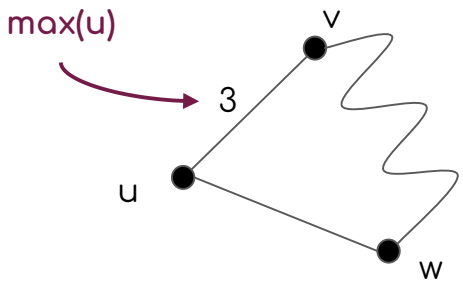
There is an edge (u, v) which is in S(G) and not in T(G)



(u, v) is not added to the maximum spanning tree
So it must introduce a cycle

Proof by Contradiction

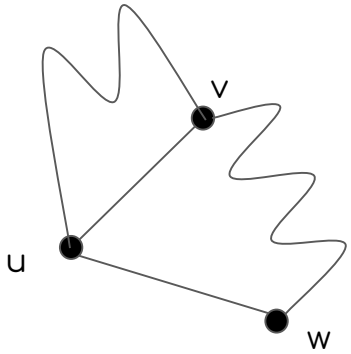
If (u, v) is introducing a cycle, there is some edge (u, w) which is already included in the spanning tree



Removing (u,w) and adding (u, v) should lead to a better maximum spanning tree.

Proof by Contradiction

If by doing that a cycle appears, there were already a path between u and v



That path + (u, w) + the path from w to v, creates a cycle in the original spanning tree

d. Prove that $w(T_G) \geq w(S_G)/2$ for any graph G.

d. Prove that $w(T_G) \geq w(S_G)/2$ for any graph G.

Based on part c. all edges in S_G are already in T_G.

Therefore $w(T_G) \geq w(S_G) \geq w(S_G)/2$

e. Give an $O(V + E)$-time algorithm to compute a 2-approximation to the maximum spanning tree.

Let N(v) denote the neighbors of a vertex v. An algorithm APPROX-MAX-SPANNING that finds S_G produces a subset of edges of a minimum spanning tree by part (c), and is a 2-approximation algorithm by part (d).

Let N(v) denote the neighbors of a vertex v. The following algorithm APPROX-MAX-SPANNING produces a subset of edges of a minimum spanning tree by part (c), and is a 2-approximation algorithm by part (d).

Algorithm 6 APPROX-MAX-SPANNING-TREE(V, E)

```
1: T = ∅
2: for v ∈ V do
3:     max = −∞
4:     best = v
5:     for u ∈ N(v) do
6:         if w(v, u) ≥ max then
7:             v = u
8:             max = w(v, u)
9:         end if
10:    end for
11:    T = T ∪ {v, u}
12: end for
13: return T
```

Dynamic Programming

Kiran Lekkala

DP Example-1

You are about to venture out of your home to go on a trek to Costco. You know the maximum weight W your car can carry and there is a set S of n potentially useful items you can buy from Costco (toilet paper, paper towels, margarita mix, etc). Each item i has a weight wi (a positive integer) and a benefit value bi (positive, not necessarily integer). Due to concerns about shortages, you are only allowed to buy 1 of each item from Costco. The goal is to select a subset of the items such that the total weight is at most W while maximizing the total benefit of the items. Give a dynamic programming algorithm to compute the largest subset of items you can buy from Costco, subject to the total weight constraint. Explain the complexity.

DP Example-1: Solution

You may recognize this as the **0-1 Knapsack Problem**

$$OPT(i, c) = \max\{OPT(i-1, c), OPT(i-1, c-w_i) + v_i\} \quad \text{if} \quad c - w_i \geq 0$$

Initialize OPT to be a matrix n+1 rows and W +1 columns. The first row corresponds to no more items, the first column corresponds to 0 weight. Fill the first row and the first column with 0s. This algorithm takes $O(nW)$.

What if the value of each item is not limited (unbounded knapsack)?

$$OPT(c) = \max_{1 \leq i < n} OPT(c - w_i) + v_i \quad \text{if } c - w_i \geq 0$$

DP Example-2

Given a value N, if we want to make change for N cents, and we have infinite supply of each of *coins = { S1, S2, .. , Sm}* valued coins, how many ways can we make the change? The order of coins doesn't matter.

For example, for N = 4 and *coins = {1,2,3}*, there are four solutions: {1,1,1,1},{1,1,2},{2,2},{1,3}. So output should be 4. For N = 10 and S = {2, 5, 3, 6}, there are five solutions: {2,2,2,2,2}, {2,2,3,3}, {2,2,6}, {2,3,5} and {5,5}. So the output should be 5.

DP Example-2

Following is the recurrence relation:

$$OPT(i, j) = OPT(i - 1, j) + OPT(i, j - coins(i - 1))$$

Base cases:

$$OPT(0, j) = 0$$
$$OPT(i, 0) = 1$$
$$OPT(0, 0) = 1$$

DP Example-2

	0	1	2	3	4	5
{}	1	0	0	0	0	0
{1}	1	1	1	1	1	1
{1,2}	1	1	2	2	3	3
{1, 2, 5}	1	1	2	2	3	4

Linear Programming (LP)

Ta-Yang Wang

Problem

A company produces two kinds of products. A product of the first type requires 2 hours of assembly labor, 1 hour of testing, and 2 dollar worth of raw materials. A product of the second type requires 3 hours of assembly, 3 hours of testing and 1 dollar worth of raw materials. Given the current personnel of the company, there can be at most 90 hours of assembly labor and 80 hours of testing, each day. Products of the first and second type have a market value of 9 dollar and 7 dollar respectively. Suppose that up to 50 hours of overtime assembly labor can be scheduled, at a cost of 2 dollar per hour. Formulate a linear programming problem that can be used to maximize the daily profit of the company.

Solution

- Variables
  - x: number of products of first type produced
  - y: number of products of second type produced
  - z: overtime in assembly (hours)
- Objective function
  - max 7x+6y-2z (daily profit: market value - raw material cost - overtime cost)
- Constraints
  - 2x+3y-z <= 90 (hours for assembly work)
  - x+3y <= 80 (hours for testing)
  - z <= 50 (hours for overtime assembly)
  - x,y,z >= 0

Greedy (Approximation)

TA: Alex Wang

Knapsack Approximation

Consider the 0 - 1 knapsack problem. Give an O(nlogn) algorithm that gives an ½ approximation. None of the items have weight greater than the total capacity.

Hint: be greedy

Solution

Let algorithm A be sorting the elements by decreasing value, and selecting as many elements until your bag cannot hold anymore. Let algorithm B be sorting the elements by decreasing value/weight density. Then we select items in this order to add to our bag until we do not have enough capacity to fit the next item. Now we run both algorithms and take the algorithm that returns the larger value. This will be a ½ approximation of the optimal solution.

Why?

Solution (continued)

Let OPT denote the optimal value, and V_A and V_B be values obtained from running the algorithms A and B, respectively. WLOG, assume that items 1, 2, …, k are the items that put into the bag with algorithm B, and that j is the first item that does not fit into the bag. Cleary the sum of values of 1, 2, …, k is equal to V_B, and $V_B + v_j >= OPT$. Now $v_j <= V_A$, since with algorithm A we start by taking the item with greatest value. Then we have that $V_A + V_B >= OPT$, meaning that at least one of V_A or V_B is greater than ½ OPT, so we are done.

Divide and Conquer

TA: Rutu Desai

Question.

Suppose you have a set of n identical gold bars, one of which is impure. The bars all look the same, however the impure bar is lighter by an unknown amount. You have a rudimentary weighing scale which you can use to compare the weight of two sets of objects. The scale tells you if its two sides weigh equal or tells you which side is lighter. Once you're done using the scale, you're charged an amount of 2^k if you used the scale k times, thus using it is very costly.

Design an algorithm to determine the impure bar while minimizing the worst-case charge for using the weighing scale (it should be **strictly less than Theta(n)**).

Solution

Algorithm:

Make three partitions, two of size (int) n/3, and the third with remaining bars.

Compare the first two partitions. If they are equal, the third partition contains the impure, if not the lighter of the two does.

Proceed with that partition and repeat till the resultant partition contains the sole impure bar.

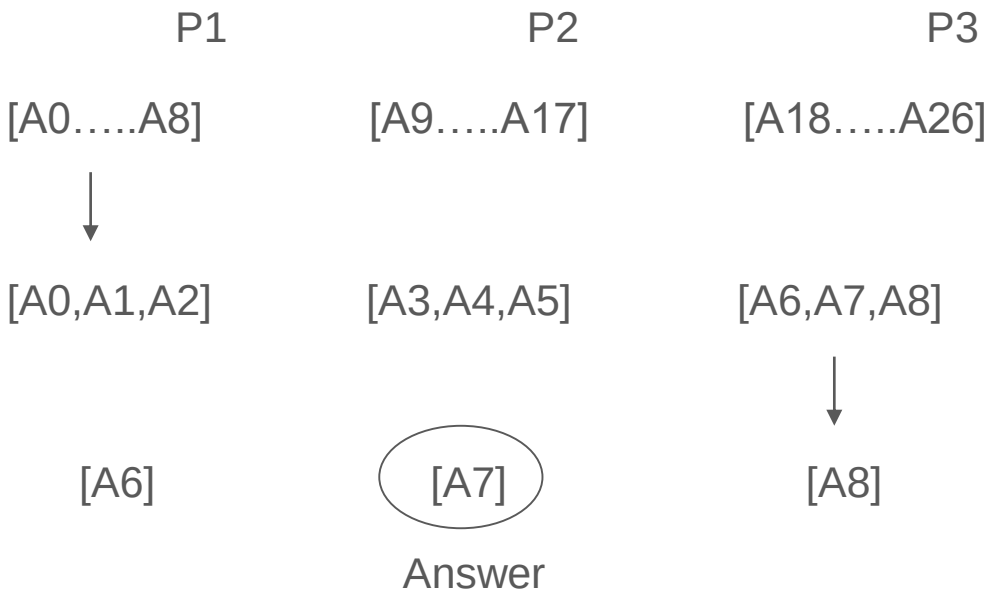
The number of times the scale is used T(n) follows the recurrence T(n) = T(n/3) + 1, solving which gives T(n) = $\log_3$ n, & a charge of Theta(n^ ($\log_3$ 2))

Dividing into two halves at each stage, gets you to pay at most Theta(n)

Let A be an array of 27 gold bars and let 8th be an impure one.

A = [A0, A1, A2,.....A26]

Dividing it into 3 parts



Network Flow

Krishna Kadam

Problem

The computer science department at a major university has a tutoring program. There are m tutors, $\{t_1, \ldots, t_m\}$ and n students who have requested the tutoring service $\{s_1, \ldots, s_n\}$. Each tutor t_i has a set T_i of topics that he/she knows, and each student s_j has a set of topics S_j that he/she wants help with. We say that tutor t_i is suitable to work with student s_j if $S_j \subseteq T_i$. (That is, the tutor t_i knows all the topics of interest to student s_j.) Finally, each tutor t_i has a range $[a_i , b_i]$, indicating that this tutor would like to work with at least a_i students and at most b_i students. Given a list of students, a list of tutors, the ranges $[a_i , b_i]$ for the tutors, and a list of suitable tutors for each student, present an efficient algorithm that determines whether it is possible to generate a pairing of tutors to students such that:

- Each student is paired with exactly one tutor.
- Each tutor t_i is paired with at least a_i and at most b_i students.
- Each student is paired only with a suitable tutor.

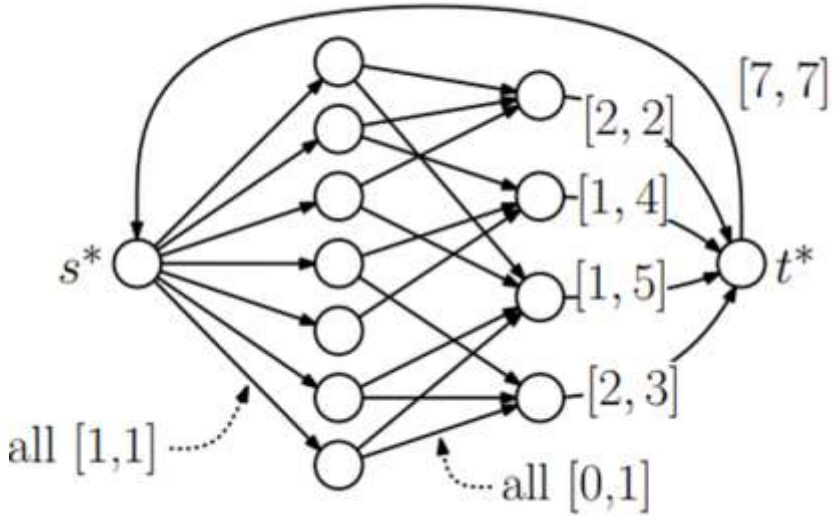
Solution

Construction:

First, we create student vertices $\{s_1, \ldots, s_n\}$ and tutor vertices $\{t_1, \ldots, t_m\}$, a special source vertex s^* , and a special sink vertex t^* . All the node demands are set to 0.

We create the edges as follows:

- For each student s_j , create edge (s^*, s_j) of capacity [1, 1].
- For each suitable student-tutor pair, create edge (s_j, t_i) of capacity [0, 1].
- From each tutor t_i , create edge (t_i, t^*) of capacity $[a_j, b_j]$.
- Create an edge (t^*, s^*) of capacity [n, n].



Solution

Claim: There is a feasible solution to the student-tutor pairing instance iff G has a

Proof:

(⇒) If there is a valid student-tutor pairing, then G has a feasible circulation.

We create a circulation for G as follows,

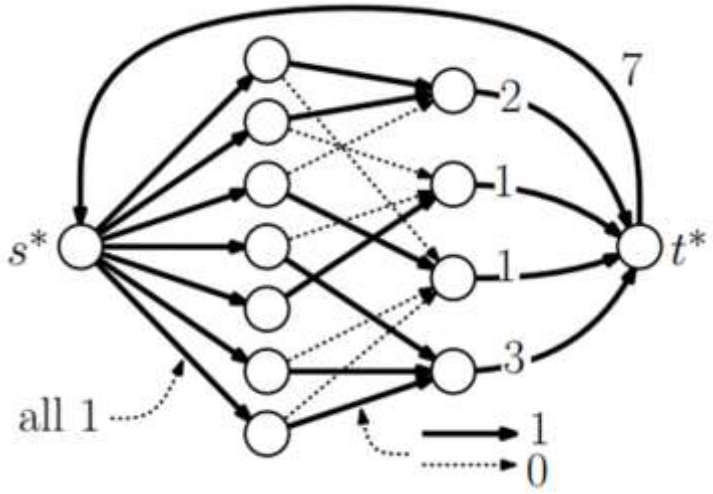
By construction, there is a one unit of flow along each edge (s^* , s_j).

For edges (s_j , t_i), the flow is 1 if student s_j is paired with tutor t_i.

For each tutor t_i , let c_j denote the number of students paired to this tutor.

We set the flow along edge (t_i , t^*) to c_j .

Finally, we set the flow along edge (t^* , s^*) to n.



Thus, there is a feasible circulation in G. Because the pairing is valid, each student is paired with exactly one tutor, and thus its demand of zero is satisfied. Also, each tutor is paired with between a_i to b_i students, and so its flow demand is also satisfied.

Solution

(⇐) If there is a feasible circulation for G, then there is a feasible student-tutor pairing.

Because the edge capacities are integers, we may assume that this is an integer flow. We define a pairing as follows. This is a valid student-tutor pairing because

1. Demands are all zero, and each student has an incoming capacity of [1, 1] it follows that each student is paired with exactly one tutor.

2. Each tutor has an outgoing capacity of $[a_i , b_i]$, it follows that each tutor is paired with the appropriate number of students.

3. We only create edges between suitable student-tutor pairs, thus the pairings selected are all suitable.

P/NP and reductions

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 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.

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 subset of the NP-hard problems.

NP - Multiple choice Questions

The problems 3-SAT and 2-SAT are

a) both in P
b) both NP-complete
c) NP-complete and in P respectively
d) undecidable and NP-complete respectively

NP - Multiple choice Questions

The problem 3-SAT and 2-SAT are

a) both in P
b) both NP-complete
c) NP-complete and in P respectively
d) undecidable and NP-complete respectively

c) is correct

NP - Multiple choice Questions

Which of the following statements are TRUE?

a) The problem of determining whether there exists a cycle in an undirected graph is in P.

b) The problem of determining whether there exists a cycle in an undirected graph is in NP.

c) If a problem A is NP-Complete, there exists a non-deterministic polynomial time algorithm to solve A.

NP - Multiple choice Questions

Which of the following statements are TRUE?

1) The problem of determining whether there exists a cycle in an undirected graph is in P.

2) The problem of determining whether there exists a cycle in an undirected graph is in NP.

3) If a problem A is NP-Complete, there exists a non-deterministic polynomial time algorithm to solve A.

All are correct. Detecting a cycle in an undirected graph can be done with a DFS, thus in P. Since it is in P, it is in NP as well. Since the problem A is NP-Complete, it is in NP and thus, there exists a non-deterministic polynomial time algorithm to solve A.

NP - Long Question 1

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.

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.

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 SAT.

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 variable, all connected to r.

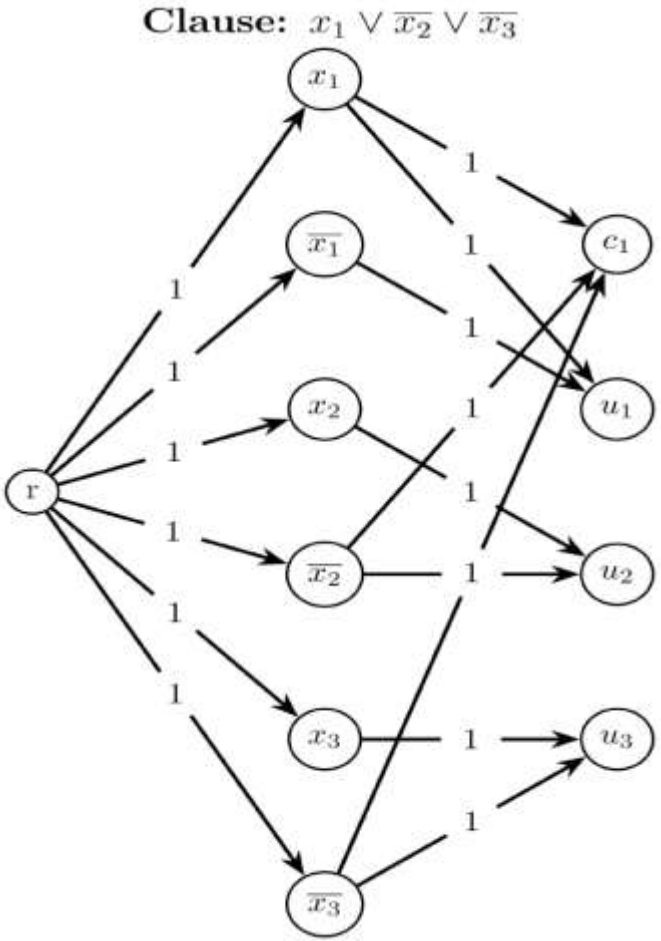
Nodes c_i for each clause, connected to its literals.

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.

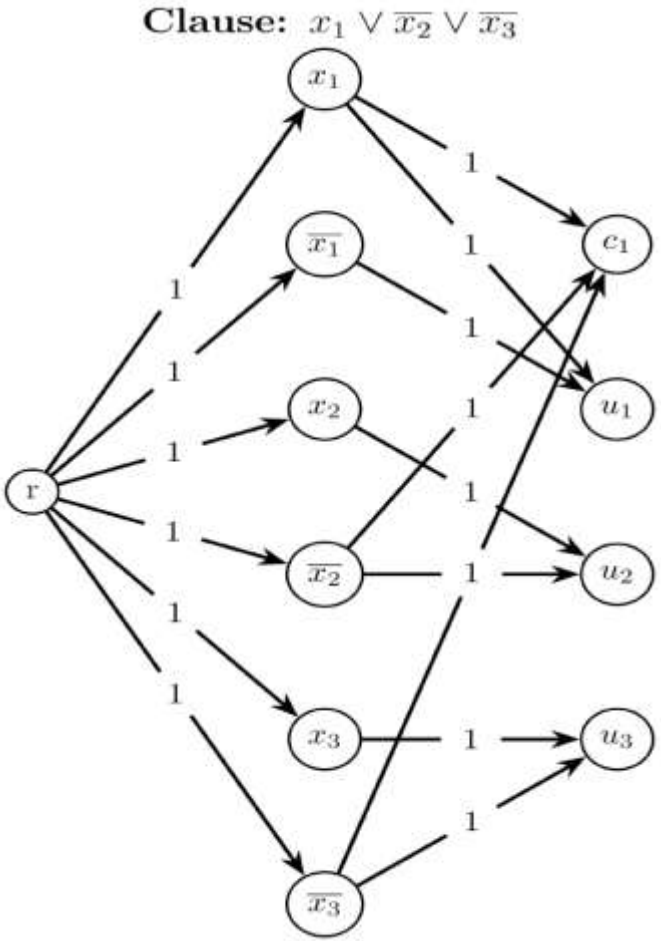
Nodes c_i for each clause, connected to corresponding literals.

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

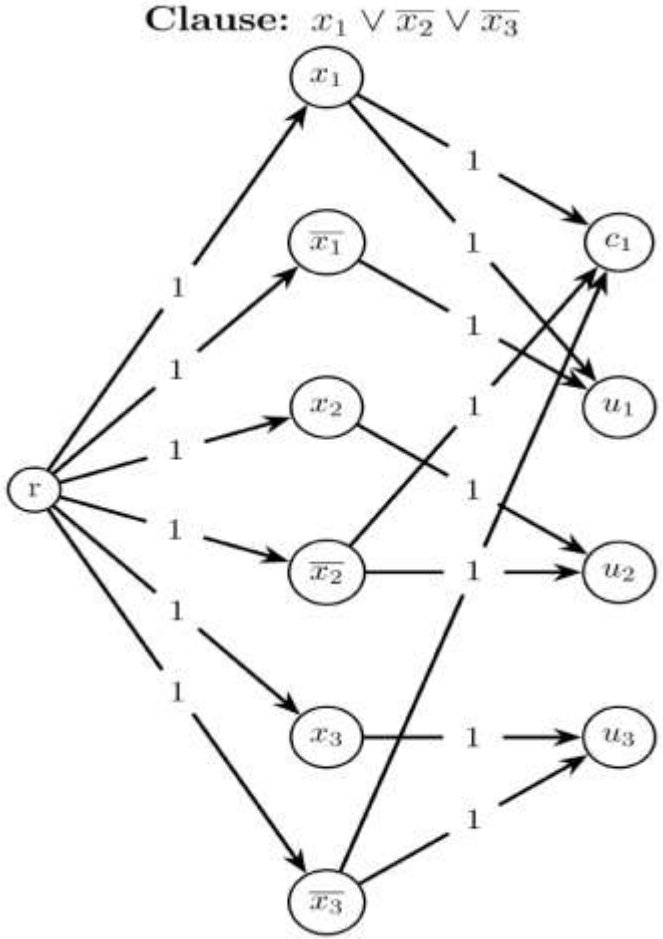


(d) SAT has a satisfying assignment => Constructed graph has a subgraph with routes from r to each x in X with weight at most 2n + m

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.

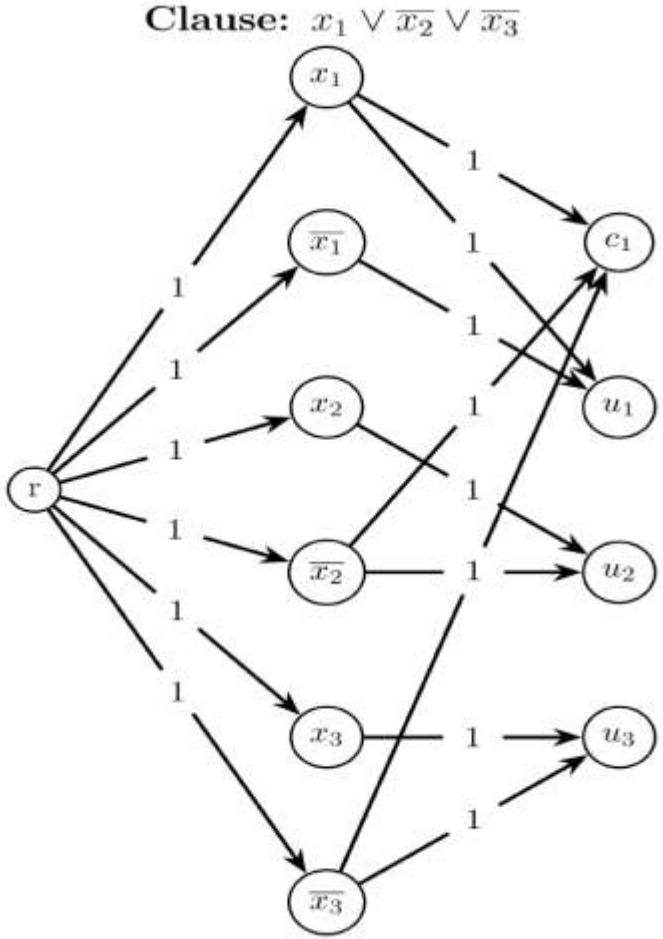


(e) Constructed graph has a subgraph spanning X with weight at most 2n + m => SAT has a satisfying assignment

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 2

In a graph G, we say that 3 nodes form a triangle if each pair is connected by an edge. Given an undirected graph G = (V, E), and positive integers p, q, the triangle-rich subgraph problem (TRIRICH) asks if there exists a subgraph G’ with at most p vertices, having at least q triangles in it. Show that TRIRICH is NP-complete.

In a graph G, we say that 3 nodes form a triangle if each pair is connected by an edge. Given an undirected graph G = (V, E), and positive integers p, q, the triangle-rich subgraph problem (TRIRICH) asks if there exists a subgraph G' with at most p vertices, having at least q triangles in it. Show that TRIRICH is NP-complete.

a) TRIRICH is in NP: A subgraph G' is a certificate. The certifier checks if it has at most p nodes. For each triplet of nodes, we can check if they form a triangle, and thus, count the total in (cubic) poly-time and check if it is at least q.

In a graph G, we say that 3 nodes form a triangle if each pair is connected by an edge. Given an undirected graph G = (V, E), and positive integers p, q, the triangle-rich subgraph problem (TRIRICH) asks if there exists a subgraph G' with at most p vertices, having at least q triangles in it. Show that TRIRICH is NP-complete.

b) TRIRICH is NP-hard: Reduction from CLIQUE (~ If a graph G has a clique of size at least k)

Given the input G, k, simply create the TRIRICH instance (G, k, ${}^{k}C_{3}$).
If k = 1, return YES if G has a vertex, otherwise no
If k = 2, return YES if G has at least 1 edge, otherwise no
If k ≥ 3, return TRIRICH(G, k, ${}^{k}C_{3}$).

G has a clique of size ≥ k iff G has a subgraph of size at most k having at least ${}^{k}C_{3}$ triangles.