

P/NP and reductions

NP - MCQ

Which of the following statements are TRUE? (G , k are problem inputs wherever mentioned)

- a) The problem of deciding if there exists a cycle in an undirected graph G is in P .
- b) The problem of deciding if there exists a cycle in an undirected graph G is in NP .
- c) The problem of deciding if there exists a cycle in an undirected graph G of size at least k is in NP .
- d) The problem of deciding if there exists a cycle in an undirected graph G of size at least k is NP -complete.

NP - MCQ

- a) The problem of deciding if there exists a cycle in an undirected graph G is in P .
- b) The problem of deciding if there exists a cycle in an undirected graph G is in NP .
- c) The problem of deciding if there exists a cycle in an undirected graph G of size at least k is in NP .
- d) The problem of deciding if there exists a cycle in an undirected graph G of size at least k is NP -complete.

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.

Deciding if there is a cycle of size at least k , has efficient certification & reduces FROM Hamiltonian cycle (G has HC iff it has a cycle of size $k=n$), making it NP -hard.

Long Question

In an undirected graph $G = (V, E)$, we say that 3 vertices form a triangle if each pair is connected by an edge. Given G , 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.

(Hint: To show NP-hardness, use the Independent Set (IS) problem)

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 . This can be done in polynomial time.

Intuition for reduction

- IS: Subset should have no edges
- TRIRICH: Subset has lots of triangles (thus, lots of edges)

Graph complement G^C : If (u,v) is an edge in G , it is not in G^C , and vice-versa.

Sparsity in G

\leftrightarrow Density in G^C

k vertices with no edges

$\leftrightarrow k$ vertices with all pairs having edges

$\leftrightarrow k$ vertices with all triplets making a triangle

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 IS

Given the input (G, k) , simply create the TRIRICH instance $(G^C, k, {}^kC_3)$.

If $k = 1$, return YES if G has a vertex, otherwise no

If $k = 2$, return YES if G has at least 1 pair of vertices with no edge, otherwise no

If $k \geq 3$, return TRIRICH($G^C, k, {}^kC_3$).

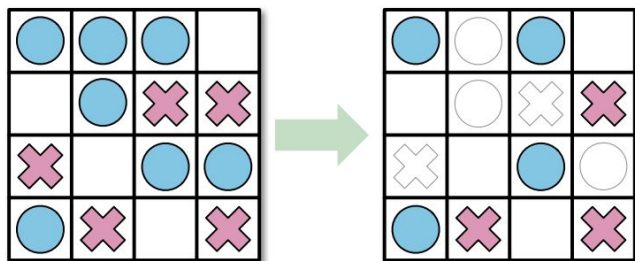
G has an IS of size $\geq k$ iff G^C has a subgraph of size at most k having at least kC_3 triangles.

Long Problem

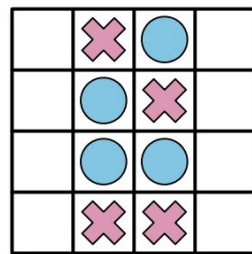
Consider the following solitaire game. The puzzle consists of an $n \times m$ grid of squares, where each square may be empty, occupied by a red stone, or occupied by a blue stone. The goal of the puzzle is to remove some of the given stones so that the remaining stones satisfy two conditions:

- every row contains at least one stone, and
- no column contains stones of both colors.

For some initial configurations of stones, reaching this goal is impossible.



A solvable puzzle and one of its many solutions.



An unsolvable puzzle.

Prove that it is NP-hard to determine, given an initial configuration of red and blue stones, whether this puzzle can be solved. (Hint: reduce from SAT/3SAT)

We show that this puzzle is NP-hard by describing a reduction from 3SAT (can choose SAT as well, works the same way).

Let Φ be a 3CNF boolean formula with n variables and m clauses. We transform this formula into a puzzle configuration in polynomial time as follows. The size of the board is $m \times n$ (rows \times cols). The stones are placed as follows, for all cells (i, j) :

- If the literal x_j appears in the i th clause of Φ , we place a blue stone at (i, j)
- If the negated variable $\sim x_j$ appears in the i th clause of Φ , we place a red stone at (i, j) .
- Otherwise, we leave cell (i, j) blank

This reduction clearly requires only $O(mn)$ time, thus, polynomial time.

For example:

We have Φ as follows:

$(a \vee b \vee c) \wedge (a \vee \sim b \vee d) \wedge (c \vee b \vee \sim d) \wedge (\sim a \vee c \vee d)$

a appears at clause 1 (= row 1) and it's the first var so it goes to column 1 as Blue.

$\sim a$ appears at clause 4 (= row 4) and it's the first var so it goes to column 1 as Red.

d appears at clause 2 (= row 2) and it's the fourth var so it goes to column 4 as Blue.

B	B	B	
B	R		B
	B	B	R
R		B	B

We claim that this puzzle has a solution if and only if Φ is satisfiable. This claim immediately implies that solving the puzzle is NP-hard. We prove our claim as follows:

First, suppose Φ is satisfiable; consider an arbitrary satisfying assignment. For each index j , remove stones from column j according to the value assigned to x_j :

- If $x_j = \text{True}$, remove all red stones from column j (keep blue if any).
- If $x_j = \text{False}$, remove all blue stones from column j (keep red if any).

In other words, remove precisely the stones that correspond to False literals. Because every variable is either True/False, each column now contains stones of only one color (if any). On the other hand, each clause of Φ must contain at least one True literal, and thus each row still contains at least one stone. We conclude that the puzzle is solvable.

On the other hand, suppose the puzzle is solvable; consider an arbitrary solution. For each index j , assign a value to x_j depending on the color of stones left in column j (Each column has at most one color since we have a valid puzzle solution):

- If column j contains blue stones, set $x_j = \text{True}$.
- If column j contains red stones, set $x_j = \text{False}$.
- If column j is empty, set x_j arbitrarily.

In other words, assign values to the variables so that the literals corresponding to the stones in puzzle solution are all True. Each row still has at least one stone, so each clause of Φ contains at least one True literal, so this assignment makes $\Phi = \text{True}$. We conclude that Φ is satisfiable.



Linear Programming

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:

$$\text{Maximize } 10(x_A + y_A) + 12(x_B + y_B) + 17(x_C + y_C) + 8(x_D + y_D)$$

Solution

3. Constraints:

Floor Space

$$0.1(x_A + y_A) + 0.15(x_B + y_B) + 0.5(x_C + y_C) + 0.05(x_D + y_D) \leq 50$$

Customer requirement

$$2(x_B + y_B) = x_C + y_C$$

Time constraint

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

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

Non-negative units

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

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, \dots, x_n , and m clauses c_1, \dots, c_m , each of which involves two of the variables. We are guaranteed that each clause is of the form $x_i \wedge \bar{x}_j$. 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 $x_i \wedge \bar{x}_j$
 - $x_i \wedge \bar{x}_j$ 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)

$$\begin{array}{ll}\text{Maximize} & z_1 + \cdots + z_m \\ \text{subject to} & z_k \leq y_i, \quad \forall c_k = x_i \wedge \bar{x}_j \\ & z_k \leq 1 - y_j, \text{ for } c_k = x_i \wedge \bar{x}_j \\ & y_i \in \{0, 1\}, \text{ for } i = 1, \dots, n.\end{array}$$

K-lane Highway :)

You are back driving on your favorite k-lane highway! The route can be viewed as consisting of points p_0, \dots, p_n , where p_0 is the starting point, p_n the ending point, and you can switch lanes at any p_i but must stick to a lane between any p_{i-1} to p_i . It takes time T_{ij} to reach p_i from p_{i-1} in any lane j . You are now so skilled that **switching lanes does not cause any delay at all**. But, there are other important considerations:

- Switching to a lane far away is not safe, in particular, you can switch from lane h to lane j only if $|h-j| \leq 2$ (equivalently, cannot switch for $|h-j| > 2$).
- You are a little sleepy, and staying in the same lane for long might make you fall asleep. So, you do NOT want to stay in the same lane for three consecutive sections. E.g. if you stay in the same lane for $p_4-p_5-p_6$ and switch, that is okay (2 sections at a time), but cannot do so for $p_4-p_5-p_6-p_7$ (3 sections at once))
- The highway has a nice oceanview on one side and cool mountain hills on the other side. So you decide that you want to drive at least once in lane 1, as well as at least once in lane k , so as to take a good look at both views.

You want to plan the route to compute the minimum possible travel time given the constraints above. Formulate this problem as an integer linear program (ILP).

a) Define the decision variables of your ILP. (Describe what they represent in English)

$X_{ij} = 1$ if you drive in lane j to reach p_i (from p_{i-1})

b) What is the objective function in your ILP?

$\text{Min}_{i,j} \quad X_{ij} T_{ij}$

c) What are the constraints in your ILP?

$$\sum_j X_{ij} = 1 \quad \forall i = 1, \dots, n \quad // \text{Must choose exactly 1 lane each section}$$
$$X_{ih} + X_{(i+1)j} \leq 1 \quad \forall h, j \text{ s.t. } |h-j| > 2 \quad \forall i = 1, \dots, n-1$$

//cannot choose lanes h and j for adjacent sections when $|h-j| > 2$

$$X_{ij} + X_{(i+1)j} + X_{(i+2)j} \leq 2 \quad \forall j \quad \forall i = 1, \dots, n-2 \quad // \text{Not 3 consecutive sections in any lane}$$

$$\sum_i X_{i1} \geq 1 \quad \& \quad \sum_i X_{ik} \geq 1 \quad // \text{At least once in lane 1, and lane k}$$
$$X_{ij} \in \{0,1\} \quad \forall i, j$$

Approximation Algorithms

- An independent set (IS) in a graph is a collection of vertices that are mutually non-adjacent. For general independent set problem, no constant factor approximation algorithm exists unless $P=NP$.
- IS problem for a bounded degree graph:
 - For graphs with Δ as the maximum degree of any vertex

Greedy-IS(G):

$S \leftarrow \text{Null}$

While G is not empty:

 Choose v s.t. $d(v) = \min [d(w): w \in V(G)]$

$S \leftarrow S \cup \{v\}$, $K \leftarrow \{v\} \cup N(v)$, $G \leftarrow G - K$

Output S

Show that Greedy-IS yields $(1/\Delta)$ -approximation.

Intuition to Proof

For each vertex added to the solution, at most Δ others are removed

Where Δ = maximum degree in the bounded degree graph,

Suppose t iterations in total for Greedy-IS.

Notation: $N(v)$ is the set of neighbors of v .

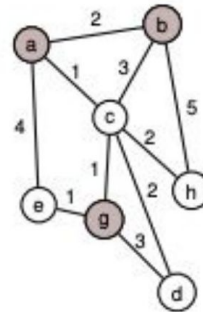
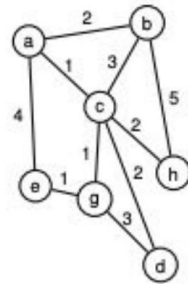
Proving for value of ρ

- If K_i be the set in the t -th iteration in Greedy-IS, then size of any K_i is:
- $|K_i| \leq \Delta + 1$ as K_i only includes some v and $N(v)$ which can be at most Δ
- Additionally, $|K_i \cap \text{OPT}| \leq \Delta$
 - This is because OPT either includes v and none of the members in $N(v)$, OR OPT includes some members from $N(v)$ (at most Δ) but definitely not v .
- So $|\text{OPT}| \leq \sum_{i=1}^t \Delta = \Delta t$
- t is the minimum possible size of S , i.e., $|S| \geq t$
- $|S| \geq |\text{OPT}|/\Delta$

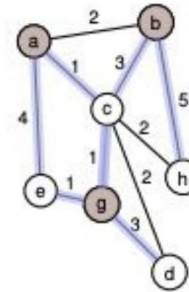


Maximum Cut Problem

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 $(S, \sim S)$ 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. $S = \emptyset$).

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:

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

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:

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

If we sum over all vertices:

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

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.

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

Since OPT uses a subset of edges:

$$2w(C) \geq \sum w(e) \geq \text{OPT}$$