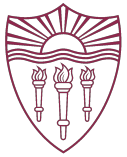
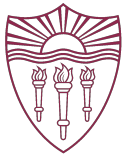


# CSCI 570

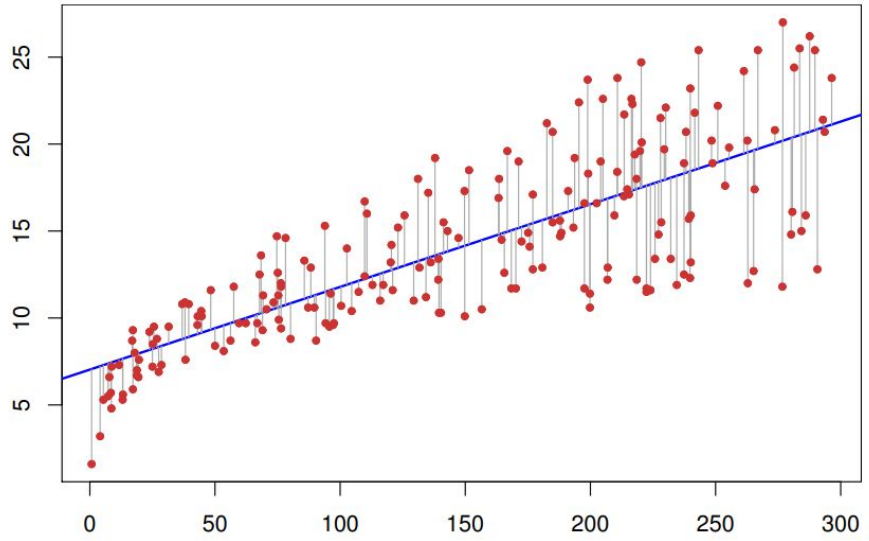
## Exam 3 Review Slides

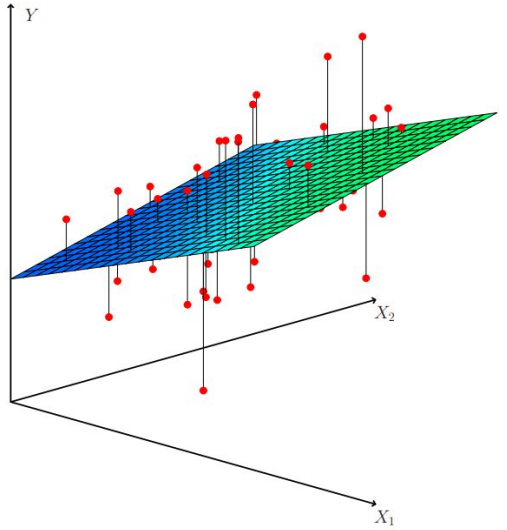


# Linear Programming

# Question 1 – Linear Regression

Given points $(x_1, y_1), \ldots, (x_n, y_n)$, 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 $i$th point

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

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

# 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)

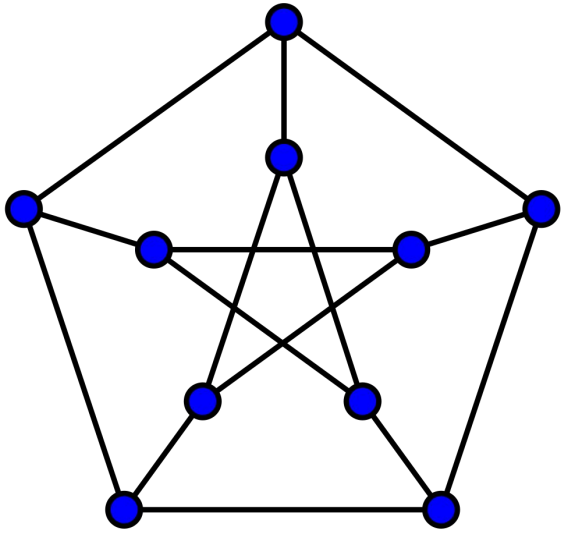
$$
\begin{aligned}
\text{Maximize} \quad & z_1 + \cdots + z_m \\
\text{subject to} \quad & z_k \leq y_i, \text{for } c_k = x_i \wedge \overline{x_j} \\
& z_k \leq 1 - y_j, \text{for } c_k = x_i \wedge \overline{x_j} \\
& y_i \in \{0, 1\}, \text{for } i = 1, \ldots, n.
\end{aligned}
$$

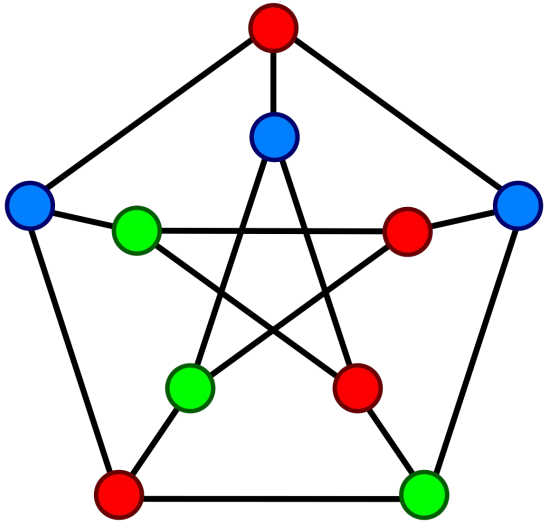
**Remarks (Just FYI, not strictly in the scope of class)**

- 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 $G = (V, E)$ with $|V| = n$. 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

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

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

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

# Solution 3 – Graph Coloring (5/5)

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

**Remark**

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

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

# Approximation Algorithms

# Independent Set

- An independent set (IS) in a graph is a collection of vertices that are mutually non-adjacent.
- We know, for general independent set problem, no approximation algorithm can guarantee to stay within a constant factor of the optimal solution.
- IS problem for a bounded degree graph:
  - Let Δ be the maximum degree on any node in G
  - Approximation algorithm, say *Greedy-IS*;
  - OPT : the optimal solution;
  - For Greedy-IS, we should be able to define a performance ratio ρ, which is a bound on the maximum ratio b/w -

$$\frac{|\text{OPT}|}{|\text{solution found by the heuristic } \textit{Greedy-IS}|}$$

# Greedy Independent Set

Greedy-IS(G):

S ← Null

While G is not empty:

Choose v such that d(v) = min [d(w): w ∈ V(G)]

S ← S ∪ {v}

K ← {v} ∪ N(v)

G ← G - K

Output S

# Intuition to Proof

For each vertex added to the solution, at most $\Delta$ others are removed

Where $\Delta$ = maximum degree in the bounded degree graph,

N(v) is the set of neighbors of v.

# Proving for value of ρ

- If $K_t$ be the set in the t-th iteration in Greedy-IS, then size of any $K_t$ is:
- $|K_t| \leq \Delta + 1$ ,
  - as $K_t$ only includes v and N(v) which can be at most Δ
- Additionally, $|K_t \cap OPT| \leq \Delta + 1 \sim \Delta$
  - 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 $|OPT| \leq \sum_{i=1}^{t} \Delta = \Delta t$
- t is the minimum possible size of S, i.e., $|S| \geq t$
- $|OPT| \leq \Delta t \leq \Delta |S|$
- $|S| \geq |OPT|/\Delta$

# Pick the correct one

When Π is a minimization problem, an efficient approximation algorithm A has a ratio α iff A is a polynomial time algorithm AND

1. For all instance I of Π, A produces a feasible solution A(I) such that val(A(I), I) ≤ α val (OPT(I), I) and α ≥ 1
2. For all instance I of Π, A produces a feasible solution A(I) such that val(A(I), I) ≤ α val (OPT(I), I) and α ≤ 1
3. For all instance I of Π, A produces a feasible solution A(I) such that val(A(I), I) ≥ α val (OPT(I), I) and α ≥ 1

# Pick the correct one

When Π is a minimization problem, an efficient approximation algorithm A has a ratio α iff A is a polynomial time algorithm AND

1. For all instance I of Π, A produces a feasible solution A(I) such that val(A(I), I) ≤ α val (OPT(I), I) and α ≥ 1
2. For all instance I of Π, A produces a feasible solution A(I) such that val(A(I), I) ≤ α val (OPT(I), I) and α ≤ 1
3. For all instance I of Π, A produces a feasible solution A(I) such that val(A(I), I) ≥ α val (OPT(I), I) and α ≥ 1

# Pick the correct one

When Π is a maximization problem, an efficient approximation algorithm A has a ratio α iff A is a polynomial time algorithm AND

1. For all instance I of Π, A produces a feasible solution A(I) such that val(A(I), I) ≤ α val (OPT(I), I) and α ≥ 1
2. For all instance I of Π, A produces a feasible solution A(I) such that val(A(I), I) ≥ α val (OPT(I), I) and α ≤ 1
3. For all instance I of Π, A produces a feasible solution A(I) such that val(A(I), I) ≥ α val (OPT(I), I) and α ≥ 1

# Pick the correct one

When Π is a maximization problem, an efficient approximation algorithm A has a ratio α iff A is a polynomial time algorithm AND

1. For all instance I of Π, A produces a feasible solution A(I) such that val(A(I), I) ≤ α val (OPT(I), I) and α ≥ 1
2. For all instance I of Π, A produces a feasible solution A(I) such that val(A(I), I) ≥ α val (OPT(I), I) and α ≤ 1
3. For all instance I of Π, A produces a feasible solution A(I) such that val(A(I), I) ≥ α val (OPT(I), I) and α ≥ 1

# P/NP and reductions

# NP- True / False / Unknown

If INDEPENDENT SET can be reduced to a decision problem X, then X can be reduced to INDEPENDENT SET. (Ind Set <p X)

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 - 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 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

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.

# Long Question

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, ${}^{k}C_{3}$).

For k ≥ 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.
Prove => and <= with similar argument as the Dense Subgraph problem.

# True / False / Unknown

For the following statements, answer if they are true/false/unknown, given the current state of knowledge?

- Some NP-complete problems are polynomial-time solvable, and some NP-complete problems are not polynomial-time solvable.
- There is an NP-complete problem that is polynomial-time solvable.
- There is an NP-complete problem that can be solved in $O(n\log n)$ time, where $n$ is the size of the input.
- There is no NP-complete problem that can be solved in $O(n\log n)$ time, where $n$ is the size of the input.

# True / False / Unknown

For the following statements, answer if they are true/false/unknown, given the current state of knowledge?

- Some NP-complete problems are polynomial-time solvable, and some NP-complete problems are not polynomial-time solvable. (False)
- There is an NP-complete problem that is polynomial-time solvable.(Unknown)
- There is an NP-complete problem that can be solved in $O(n\log n)$ time, where $n$ is the size of the input. (Unknown)
- There is no NP-complete problem that can be solved in $O(n\log n)$ time, where $n$ is the size of the input.(Unknown)

# True / False / Unknown

If we could find one NP-hard problem Y, we could prove that another problem X is NP-hard by reducing X to Y.

# True / False / Unknown

If we could find one NP-hard problem Y, we could prove that another problem X is NP-hard by reducing X to Y.

False

# Short Answer

Describe the error in the following fallacious “proof” that P ≠ NP. Consider an algorithm for SAT: *“On input φ, try all possible assignments to the variables. Accept if any satisfy φ.”* This algorithm clearly requires exponential time. Thus, SAT has exponential time complexity. Therefore SAT is not in P. Because SAT is in NP, it must be the case that P ≠ NP

# Short Answer

Describe the error in the following fallacious “proof” that P ≠ NP. Consider an algorithm for SAT: *“On input φ, try all possible assignments to the variables. Accept if any satisfy φ.”* This algorithm clearly requires exponential time. Thus, SAT has exponential time complexity. Therefore SAT is not in P. Because SAT is in NP, it must be the case that P ≠ NP

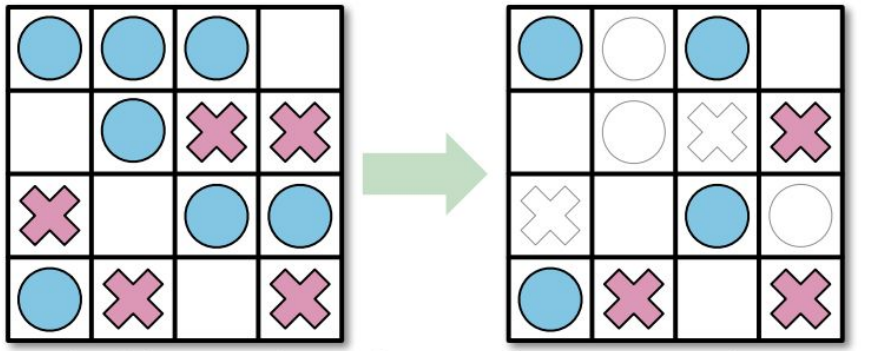
The error lies in the definition of the “time complexity” of SAT. The time complexity of SAT is the running time of the best decider for SAT. The given algorithm is just one, and might not be the best one. Therefore, all it tells us is that the time complexity of SAT is at most exponential.

# Long Problem

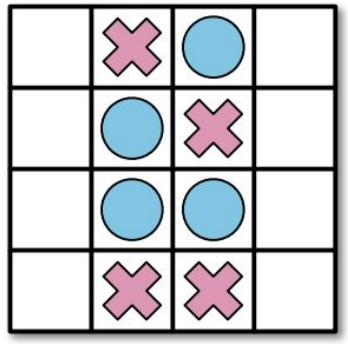
Consider the following solitaire game. The puzzle consists of an n × m grid of squares, where each square may be empty, occupied by a red stone, or occupied by a blue stone. The goal of the puzzle is to remove some of the given stones so that the remaining stones satisfy two conditions:

- 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.

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 × n (rows × cols). The stones are placed as follows, for all cells (i,j):

- If the literal $x_j$ appears in the ith clause of Φ, we place a blue stone at (i, j)
- If the negated variable $\sim x_j$ appears in the ith 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 ∨ b ∨ c) ∧(a ∨ ~b ∨ d) ∧(c ∨ b ∨ ~d) ∧(~a ∨ c ∨ d )

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

~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 xj :

– If xj = True, remove all red stones from column j.

– If xj = False, remove all blue stones from column j.

In other words, remove precisely the stones that correspond to False literals. Because every variable appears in at least one clause, 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 xj 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 xj = True.

– If column j contains red stones, set xj = False.

– If column j is empty, set xj arbitrarily.

In other words, assign values to the variables so that the literals corresponding to the remaining stones 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 Φ = True. We conclude that Φ is satisfiable.