MCS 312: NP Completeness and Approximation algorithms

Instructor

Neelima Gupta

ngupta@cs.du.ac.in

Table of Contents

NP - Hardness

- The aim to study this class is not to solve a problem but to see how hard is a problem?

Reduction

- The crux of NP-Hardness is *reducibility*
  - We say that a problem P is reduced to another problem Q if *an* instance of P can be easily transformed into an instance of Q, the solution to which provides a solution to the instance of P.
  - Intuitively it means that if one can solve Q then one can solve P also, i.e. P is "no harder to solve" than Q or Q is at least as hard as P.

$$\begin{tikzcd}
I_P \arrow[r, "\textit{Poly. Time}"] & I_Q \arrow[d, "\textit{Poly. Time}"] \\
S_P & S_Q \arrow[l] \\
\textit{(Yes/No)} & \textit{(Yes/No)}
\end{tikzcd}$$

I_P : Instance of problem P

I_Q : Instance of problem Q

S_P : Solution of problem P

S_Q : Solution of problem Q

Transformation Characterstics :

- If A(Q) is yes then A(P) is yes
- Vice versa
- It should be done in polynomial time

Thanks to: Shivam Sharma
(24, MCS '09)

Reducibility

- An example:
  - P: Given a set of Booleans, is at least one TRUE?
  - Q: Given a set of integers, is their sum positive?
  - Transformation: $(x_1, x_2, \ldots, x_n) = (y_1, y_2, \ldots, y_n)$ where $y_i = 1$ if $x_i =$ TRUE, $y_i = 0$ if $x_i =$ FALSE
- Another example:
  - Solving linear equations is reducible to solving quadratic equations
    - *How can we easily use a quadratic-equation solver to solve linear equations?*

NP hard

Q is s.t.b. NP-hard if $\forall\ P \in NP,\ P \leq_p Q$

$$\begin{tikzcd}
I_P \arrow[r, "\text{Poly. Time}"] & I_Q \arrow[d, "\text{Poly. Time}"] \\
\underset{\text{(Yes/No)}}{S_P} & \underset{\text{(Yes/No)}}{S_Q} \arrow[l]
\end{tikzcd}$$

I_P : Instance of problem P

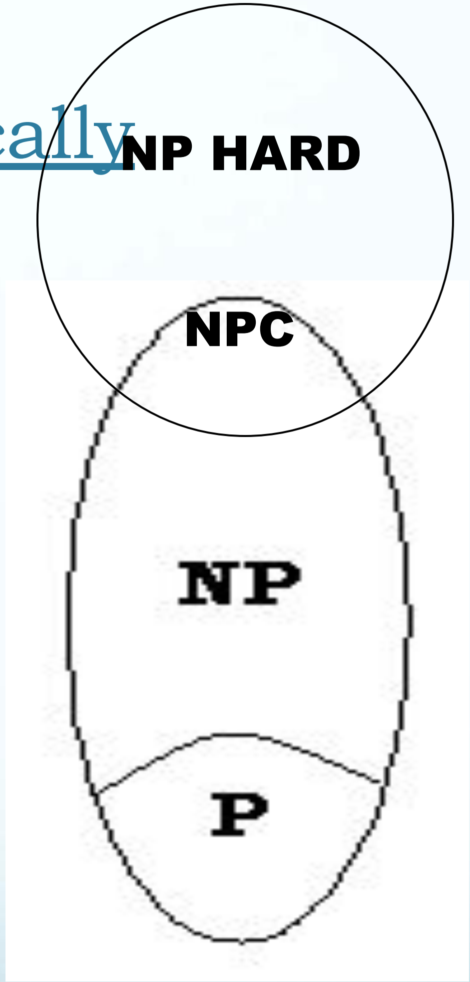
I_Q : Instance of problem Q

S_P : Solution of problem P

S_Q : Solution of problem Q

Thanks to: Shivam Sharma
(24, MCS '09)

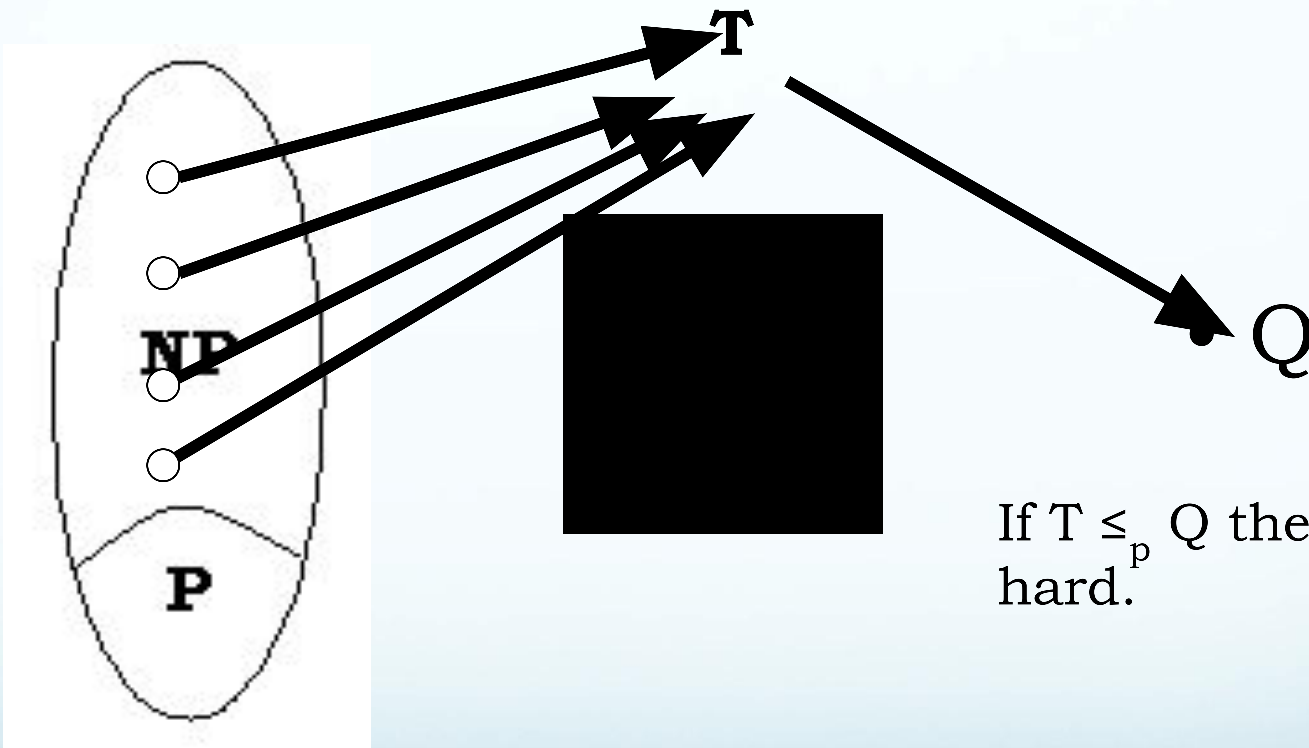
Diagrammatically



Thanks to: Shivam Sharma
(24, MCS '09)

If all problems $R \in NP$ are reducible to T, then T is NP-Hard

And $T \in NP$ then T is NPC.



If $T \leq_p Q$ then Q is NP hard.

Thanks to: Shivam Sharma (24, MCS '09)

The SAT Problem

- One of the first problems proved to be NP-Hard was *satisfiability* (SAT):
  - Given a Boolean expression on n variables, can we assign values such that the expression is TRUE?
  - Ex: $((x_1 \rightarrow x_2) \vee \leftarrow((\leftarrow x_1 \leftrightarrow x_3) \vee x_4)) \wedge \leftarrow x_2$
  - **Cook's Theorem:** The satisfiability problem is NP-Hard (actually NP Complete…will do this later)
    - **Note: Argue from first principles, not reduction**
    - **Proof: not here**

Conjunctive Normal Form

- Even if the form of the Boolean expression is simplified, the problem is NP-Hard (NP Complete)
  - *Literal*: an occurrence of a Boolean or its negation
  - A Boolean formula is in *conjunctive normal form*, or *CNF*, if it is an AND of clauses, each of which is an OR of literals
    - Ex: $(x_1 \vee \leftarrow x_2) \wedge (\leftarrow x_1 \vee x_3 \vee x_4) \wedge (\leftarrow x_5)$
  - *3-CNF*: each clause has exactly 3 distinct literals
    - Ex: $(x_1 \vee \leftarrow x_2 \vee \leftarrow x_3) \wedge (\leftarrow x_1 \vee x_3 \vee x_4) \wedge (\leftarrow x_5 \vee x_3 \vee x_4)$
    - Notice: true if at least one literal in each clause is true

The 3-CNF Problem

- Thm 36.10: Satisfiability of Boolean formulas in 3-CNF form (the *3-CNF Problem*) is NP-Hard (NP-Complete)
  - Proof: Nope
- The reason we care about the 3-CNF problem is that it is relatively easy to reduce to others
  - Thus by proving 3-CNF NP-Hard we can prove many seemingly unrelated problems
    NP-Hard

CLIQUE is NP Hard

- Pick up a problem known to be NPHard and
  - Transform (reduce) the known problem to CLIQUE
  - 0 Give the transformation
    1. Show that under the transformation : solution of known problem is yes => solution to CLIQUE is yes.
    2. Show that under the transformation : solution of CLIQUE is yes => solution of the known problem is yes.
    3. Show that the transformation can be done in time polynomial in the length of an instance of the known problem.

SO, THREE STEPS TO REDUCE A KNOWN PROBLEM TO CLIQUE.

3-CNF → Clique

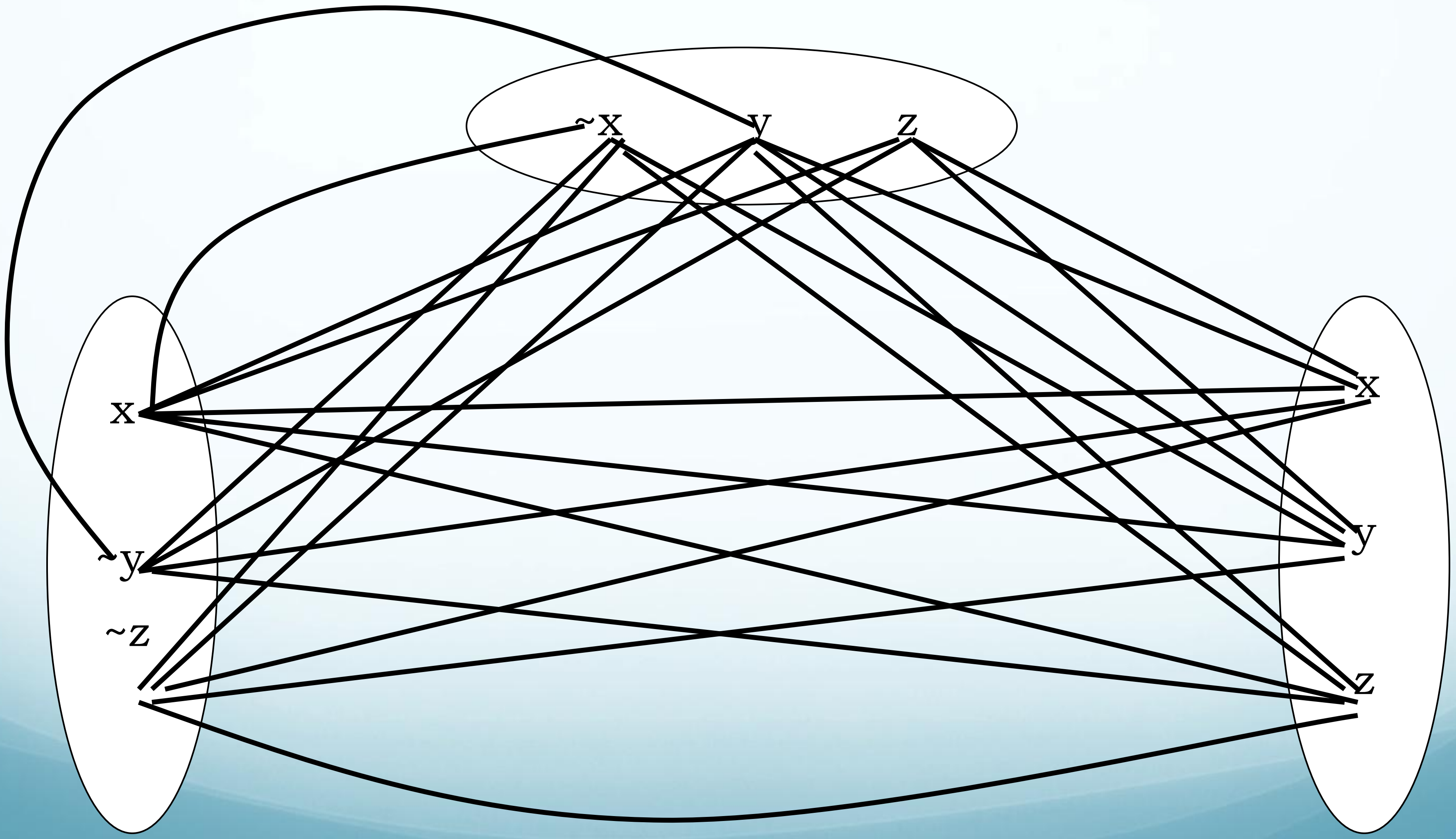
- *What should the reduction do?*
- A: Transform a 3-CNF formula to a graph, for which a *k*-clique will exist (for some *k*) iff the 3-CNF formula is satisfiable

3-CNF → Clique

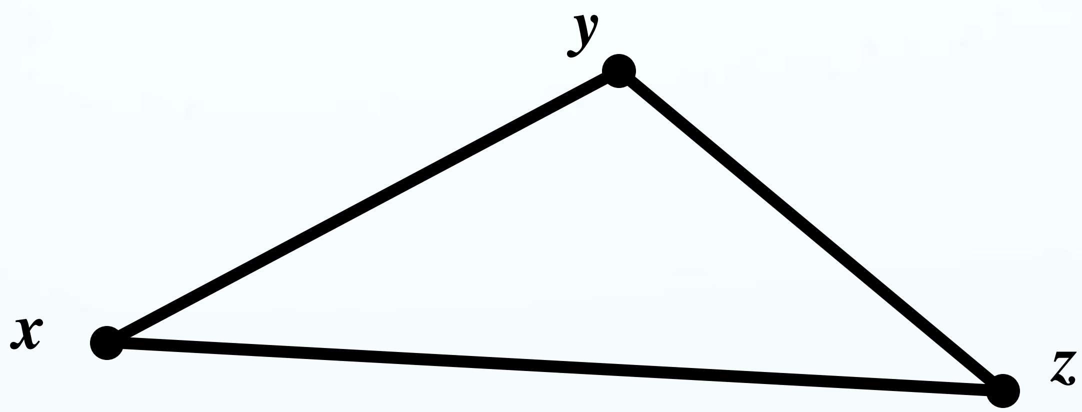
- The reduction:
  - Let $B = C_1 \wedge C_2 \wedge \ldots \wedge C_k$ be a 3-CNF formula with k clauses, each of which has 3 distinct literals
  - For each clause put a triple of vertices in the graph, one for each literal
  - Put an edge between two vertices if they are in different triples and their literals are *consistent*, meaning not each other's negation

Let the expression in 3CNF be: (~x v y v z) $\wedge$ (x v ~y v ~z) $\wedge$ (x v y v z)

Expression → Graph



Clique thus formed:



Note:- There are many other possible cliques in previous mapping. This is one of the possible cliques.

3-CNF → Clique

- Prove the reduction works:
  - If B has a satisfying assignment, then each clause has at least one literal (vertex) that evaluates to 1
  - Picking one such "true" literal from each clause gives a set V' of *k* vertices. V' is a clique (*Why?*)
  - If G has a clique V' of size k, it must contain one vertex in each triple (clause) (*Why?*)
  - We can assign 1 to each literal corresponding with a vertex in V', without fear of contradiction

Reduction takes polynomial time

- Let there be n variables in the 3-CNF with k clauses
- Then, the input size is theta(k + n).
- Size of the graph = 3k*3(k-1)

Vertex Cover is NP-Hard

- Pick up a problem known in NP-hard
  - CLIQUE

Clique $\leq_p$ Vertex Cover

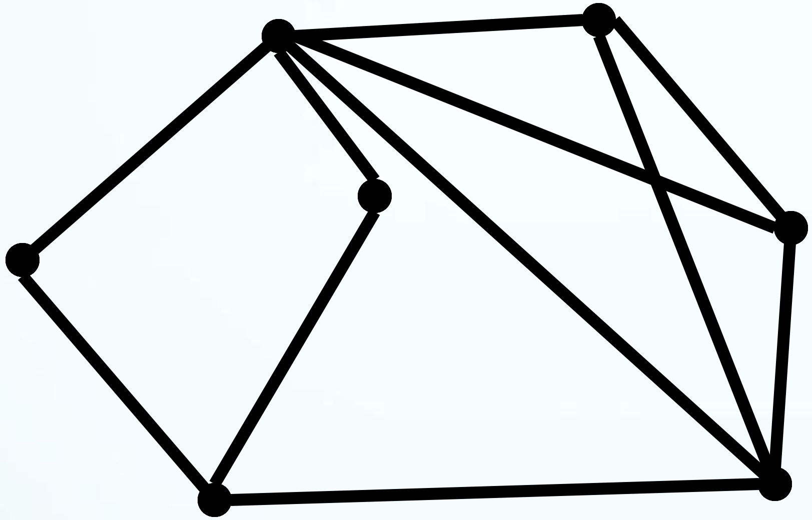
- Let the instance of Clique (I_c) be <G, k>.

- Reducing it to instance of VC (I_{vc}) be <G', |V|-k>

  where G' : E(G')=Edges b/w vertex pair not present in G and |V|-k is the vertex cover.

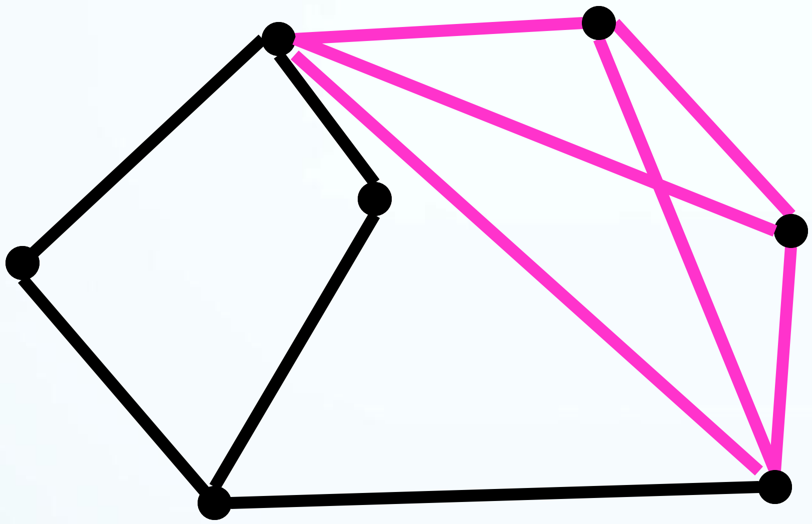
- Catch behind this choice : Because it works…!!!

G

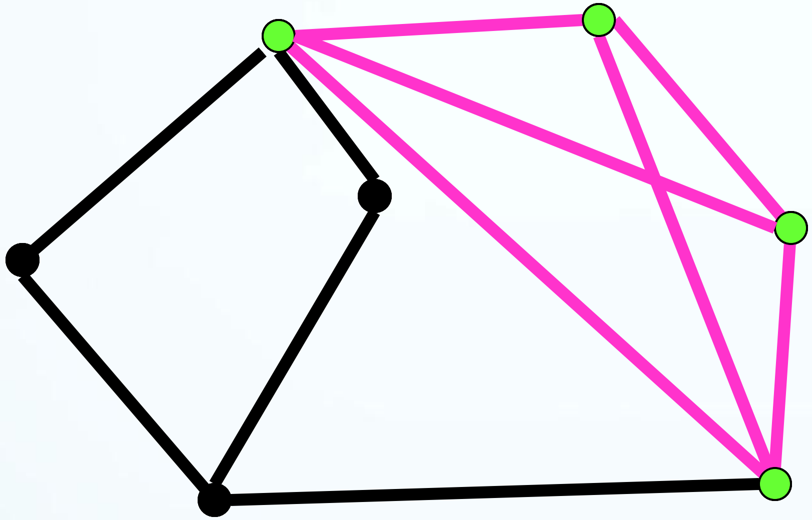


Green ovals represent CLIQUE for this graph

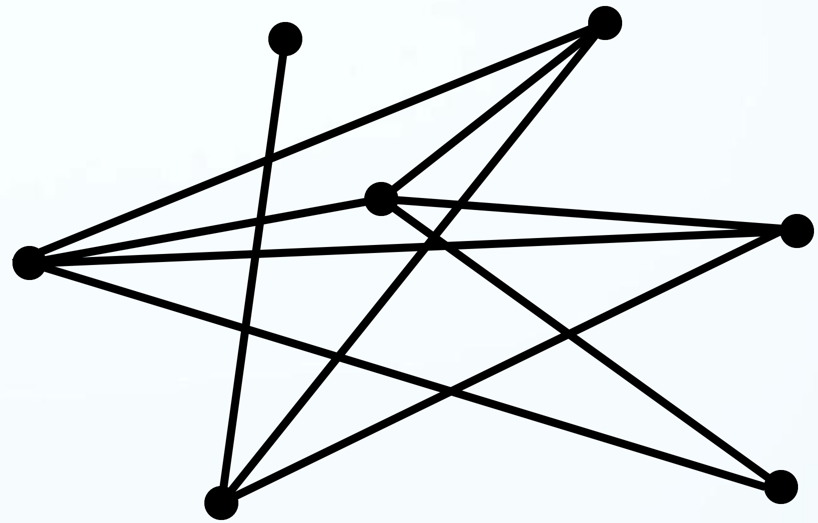
G'



G



G’



Big ovals represent the VC for graph G’

Clique → Vertex Cover

- Reduce *k*-clique to vertex cover
  - The *complement* G_C of a graph G contains exactly those edges not in G
  - Compute G_C in polynomial time
  - G has a clique of size *k* iff G_C has a vertex cover of size $|V| - k$

Clique → Vertex Cover

- Claim: If G has a clique of size *k*, G_C has a vertex cover of size |V| - *k*
  - Let V' be the *k*-clique
  - Then V - V' is a vertex cover in G_C
    - Let (*u*,*v*) be any edge in G_C
    - Then *u* and *v* cannot both be in V' (*Why?*)
    - Thus at least one of *u* or *v* is in V-V' (*why?*), so edge (*u*, *v*) is covered by V-V'
    - Since true for *any* edge in G_C, V-V' is a vertex cover

Clique $\rightarrow$ Vertex Cover

- Claim: If G_C has a vertex cover $V' \subseteq V$, with $|V'| = |V| - k$, then G has a clique of size k
  - For all $u,v \in V$, if $(u,v) \in G_C$ then $u \in V'$ or $v \in V'$ or both (*Why?*)
  - Contrapositive: if $u \notin V'$ and $v \notin V'$, then $(u,v) \in E$
  - In other words, all vertices in V-V' are connected by an edge, thus V-V' is a clique
  - Since $|V| - |V'| = k$, the size of the clique is k

Independent Set Problem

Independent Set : A subset S of V is said to be independent if no 2 nodes in S are joined by an edge.

Problem Statement: Given a graph G=(V,E), find an independent set that is as large as possible.

Exercise

- Show that Independent Set is NP Hard by reducing it from
  - 3 CNF
  - Clique
  - Vertex Cover
- Show that Vertex Cover is NP Hard by reducing it from
  - 3 CNF

Subset Sum Problem

Problem Statement

Given: A finite set S of natural numbers.

A target $t \in N$.

To Find: If there exists a subset S' of S whose elements sum up to t.

Subset Sum Problem

We now prove that Subset Sum Problem is NP-Complete.

Subset Sum is in NP. For an instance <S,t>, let S' be the certificate. Checking whether elements of S' sum to t can be done in polynomial time.

Subset Sum Problem

Subset Sum is NP Hard

We show this by proving that 3-SAT is reducible to Subset Sum in polynomial time.

Given: 3-SAT formula Φ over variables x_1, x_2, ……x_n with clauses C_1, C_2……C_k

Subset Sum Problem

Without loss of generality, we make the following 2 assumptions:

- No clause contains both a variable and its negation. WHY?

  (Because such a clause would be trivially satisfied.)
- Each variable appears in at least 1 clause. WHY?

  (Because otherwise, it does not matter what value is assigned to it.)

Subset Sum Problem

Reduction Process - through example

Consider the 3-SAT formula : $\Phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$

where $C_1 = (x_1 \vee x_2' \vee x_3')$
$C_2 = (x_1' \vee x_2' \vee x_3')$
$C_3 = (x_1' \vee x_2' \vee x_3)$
$C_4 = (x_1 \vee x_2 \vee x_3)$

A satisfying assignment is $\langle x_1=0, x_2=0, x_3=1 \rangle$

Subset Sum Problem

Steps:

- Create 2 numbers in set S for each variable x_i and 2 numbers for each clause C_j.
- These numbers are in base 10 and each has n+k digits.
- Each digit corresponds to a variable or a clause. Label least significant k digits by clauses and most significant n digits by variables.

Subset Sum Problem

- *Do the following for i = 1……n*
- *If xi = 1 in the assignment, include v_i in S', otherwise include v_i'. In the example,*
- *x1=0 => x1'=1 , v_1' is selected*
- *x2=0 => x2'=1 , v_2' is selected*
- *x3=1 => x3=1 , v_3 is selected*

(x1 v x2' v x3') Λ (x1' v x2' v x3') Λ (x1' v x2' v x3) Λ (x1 v x2 v x3)

Satisfying assignment x1=0, x2=0, x3=1

x_1', x_2' and x_3 are selected.

	X1	X2	X3	C1	C2	C3	C4
V1	1	0	0	1	0	0	1
V1'	**1**	0	0	0	1	1	0
V2	0	1	0	0	0	0	1
V2'	0	**1**	0	1	1	1	0
V3	0	0	**1**	0	0	1	1
V3'	0	0	1	1	1	0	0
S1	0	0	0	1	0	0	0
S1'	0	0	0	2	0	0	0
S2	0	0	0	0	1	0	0
S2'	0	0	0	0	2	0	0
S3	0	0	0	0	0	1	0
S3'	0	0	0	0	0	2	0
S4	0	0	0	0	0	0	1
S4'	0	0	0	0	0	0	2
t	1	1	1	4	4	4	4

(x1 v x2’ v x3’) Λ (x1’ v x2’ v x3’) Λ (x1’ v x2’ v x3) Λ (x1 v x2 v x3)

Satisfying assignment x1=0, x2=0, x3=1

	X1	**X2**	**X3**	**C1**	**C2**	**C3**	**C4**
V1	1	0	0	1	0	0	1
V1'	**1**	0	0	0	1	1	0
V2	0	1	0	0	0	0	1
V2'	0	**1**	0	**1**	1	1	0
V3	0	0	**1**	0	0	1	1
V3'	0	0	1	1	1	0	0
S1	0	0	0	**1**	0	0	0
S1'	0	0	0	**2**	0	0	0
S2	0	0	0	0	1	0	0
S2'	0	0	0	0	2	0	0
S3	0	0	0	0	0	1	0
S3'	0	0	0	0	0	2	0
S4	0	0	0	0	0	0	1
S4'	0	0	0	0	0	0	2
t	1	1	1	4	4	4	4

(x1 v x2' v x3') Λ (x1' v x2' v x3') Λ (x1' v x2' v x3) Λ (x1 v x2 v x3)

Satisfying assignment x1=0, x2=0, x3=1

	X1	**X2**	**X3**	**C1**	**C2**	**C3**	**C4**
V1	1	0	0	1	0	0	1
V1'	**1**	0	0	0	**1**	1	0
V2	0	1	0	0	0	0	1
V2'	0	**1**	0	**1**	**1**	1	0
V3	0	0	**1**	0	0	1	1
V3'	0	0	1	1	1	0	0
S1	0	0	0	**1**	0	0	0
S1'	0	0	0	**2**	0	0	0
S2	0	0	0	0	1	0	0
S2'	0	0	0	0	**2**	0	0
S3	0	0	0	0	0	1	0
S3'	0	0	0	0	0	2	0
S4	0	0	0	0	0	0	1
S4'	0	0	0	0	0	0	2
t	1	1	1	4	4	4	4

(x1 v x2' v x3') Λ (x1' v x2' v x3') Λ (x1' v x2' v x3) Λ (x1 v x2 v x3)

Satisfying assignment x1=0, x2=0, x3=1

	X1	**X2**	**X3**	**C1**	**C2**	**C3**	**C4**
V1	1	0	0	1	0	0	1
V1'	**1**	0	0	0	**1**	**1**	0
V2	0	1	0	0	0	0	1
V2'	0	**1**	0	**1**	**1**	**1**	0
V3	0	0	**1**	0	0	**1**	1
V3'	0	0	1	1	1	0	0
S1	0	0	0	**1**	0	0	0
S1'	0	0	0	**2**	0	0	0
S2	0	0	0	0	1	0	0
S2'	0	0	0	0	**2**	0	0
S3	0	0	0	0	0	**1**	0
S3'	0	0	0	0	0	2	0
S4	0	0	0	0	0	0	1
S4'	0	0	0	0	0	0	2
t	1	1	1	4	4	4	4

(x1 v x2' v x3') Λ (x1' v x2' v x3') Λ (x1' v x2' v x3) Λ (x1 v x2 v x3)

Satisfying assignment x1=0, x2=0, x3=1

	X1	X2	X3	C1	C2	C3	C4
V1	1	0	0	1	0	0	1
V1'	**1**	0	0	0	**1**	**1**	0
V2	0	1	0	0	0	0	1
V2'	0	**1**	0	**1**	**1**	**1**	0
V3	0	0	**1**	0	0	**1**	**1**
V3'	0	0	1	1	1	0	0
S1	0	0	0	**1**	0	0	0
S1'	0	0	0	**2**	0	0	0
S2	0	0	0	0	1	0	0
S2'	0	0	0	0	**2**	0	0
S3	0	0	0	0	0	**1**	0
S3'	0	0	0	0	0	2	0
S4	0	0	0	0	0	0	**1**
S4'	0	0	0	0	0	0	**2**
t	1	1	1	4	4	4	4

Subset Sum Problem

Construct S and t as follows:

- t has a 1 in each digit labeled by a variable and 4 in each clause-digit.
- For each x_i, there exist 2 integers v_i, v_i' in S. Both v_i and v_i' have 1 corresponding to digit x_i.
- If x_i appears in C_j, the C_j-digit in v_i = 1

  If v_i' appears in C_j, the C_j-digit in v_i' = 1
- All other digits are zero.

Subset Sum Problem

Claim: All v_i and v_i' in S are unique

- v_i (v_i') and v_j (v_j') will be different in most significant positions.
- v_i and v_i' will be different in least significant positions. WHY?

(both cannot belong to the same clause)

Subset Sum Problem

- For all C_j, there exist s_j and s_j' integers in S. Both have 0's in all digits other than the one labeled by C_j.
- s_j has a 1 corresponding to C_j, and s_j' has a 2 corresponding to C_j.
- These integers are slack variables, used to get clause labeled digit position to add to the target value of 4.

Subset Sum Problem

Claim: All s_j and s_j’ in S are unique
(for reasons similar to v_i and v_i’)

Observation: The greatest sum of digits in any digit position is 6. This occurs in clause-digits (v_i and v_i’ make a contribution of 3, s_j and s_j’ make a contribution of 1 and 2 respectively).
Conclusion: Interpretation is in base 10, so no carries would be generated.

REDUCTION DONE !!

Subset Sum Problem

Claim: This reduction can be done in polynomial time.

- S contains 2n+2k values.
- Each has n+k digits.
- Each digit takes time polynomial in (n+k) to be produced.
- t has n+k digits each being produced in constant time.

Hence Proved !

Subset Sum Problem

To Prove: 3-SAT Φ is satisfiable if and only if there exists a subset S' of S whose elements sum to 't'.

Proof

Part 1

Given: Φ has a satisfying assignment.

Subset Sum Problem

- Do the following for i = 1……n

  If x_i = 1 in the assignment, include v_i in S', otherwise include v_i'. In the example, v_1', v_2', v_3 belong to S'.

Note: For each variable digit, the sum of values of S' must be 1 (= those of target t)

Subset Sum Problem

- Each clause is satisfied, therefore has at least one positive literal. Thus, each clause digit has at least one ‘1’ through a vi or vi’ value in S’. (the sum of clause digit may be 1 or 2 or 3).

- Include appropriate non empty subset of slack variables $\{s_j, s_j'\}$ in S’ to achieve the target of 4 in each digit labeled by C_j.

Subset Sum Problem

Since we have matched all target digits of the sum, and there does not exist any carry, therefore the values of S' sum to t.

Subset Sum Problem

Part 2 of the proof

Given: Subset S' of S sums to t.

Observe the following:

- S' must include exactly one of v_i and v_i' for all i. WHY?

  Because otherwise variable digits would not sum to 1

Subset Sum Problem

- If v_i belongs to S', set $x_i = 1$.
- If v_i' belongs to S', set $x_i' = 0$.

Claim: Every clause C_j is satisfied by this assignment.

Proof: Note that in order to achieve a sum of in digits corresponding to Cj, the subset S' must include at least one vi or vi' value that has a value 1 in the digit labeled by Cj.

Subset Sum Problem

- Since we have xi =1 if vi belongs to S', clause Cj is satisfied.
- And since xi =0 if vi' belongs to S', again clause Cj is satisfied.
- Therefore, all clauses are satisfied.

Hence Proved !

Set Cover Problem

Problem Statement

Given

1. A set U of n elements
2. A collection $S_1, S_2, \ldots\ldots, S_m$ of subsets of U
3. A number k

To Find If there exists a collection of at most k of these sets whose union equals all of U.

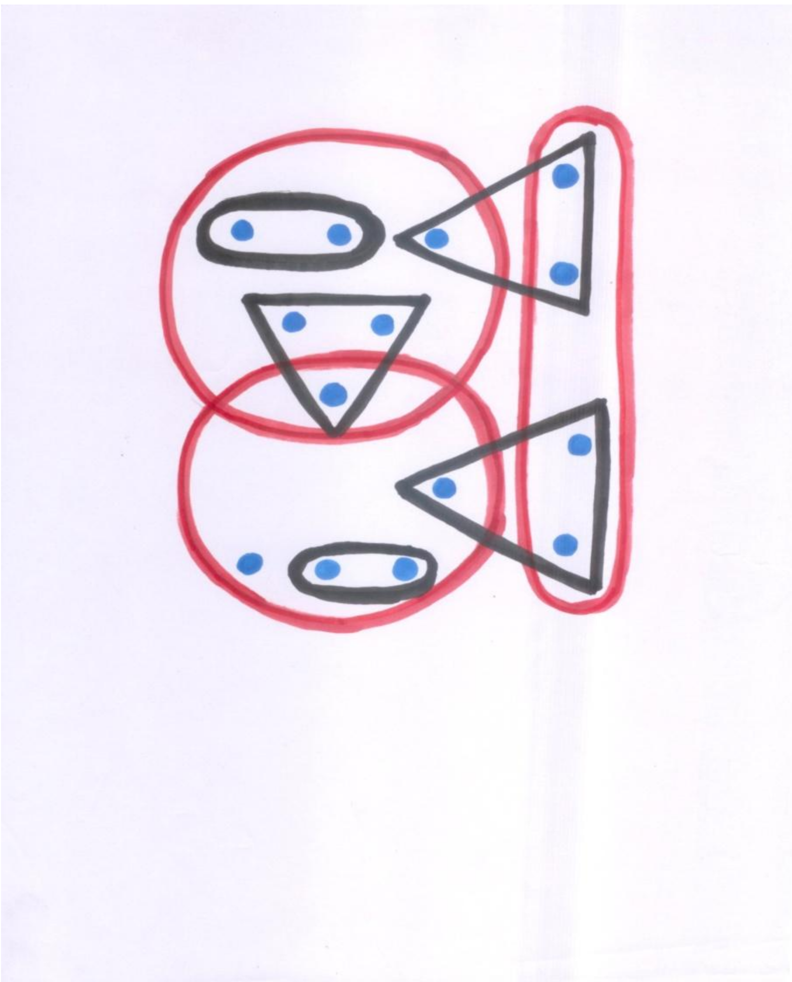
Set Cover Problem

An Application

- Suppose we want to build a system with n functionalities using m available pieces of software.
- Each piece of software possesses some subset of functionalities. Let the set of functionalities possessed by the i^{th} piece of software be denoted by S_i.
- Our goal, then, is to build a system that possesses all the n functionalities using a small number of pieces of software.

Set Cover Problem

An Instance



- The little blue dots are the elements of U
- Black and Red figures represent sets. The dots that lie within a figure are the elements contained by that set.
- The red figure form the set cover.

Set Cover Problem is NP Complete

- Prove that it is in NP
- NP – hardness follows from reduction from vertex cover. HOW?......Assignment

(Metric) Traveling Salesman Problem

Problem Statement

Given A complete graph G with non-negative edge costs (that satisfy triangle inequality)

To Find A minimum cost cycle visiting every vertex exactly once.

Decision Version: Does there exist a TS tour of cost <=k

TSP is NP Complete

- Prove that it is in NP

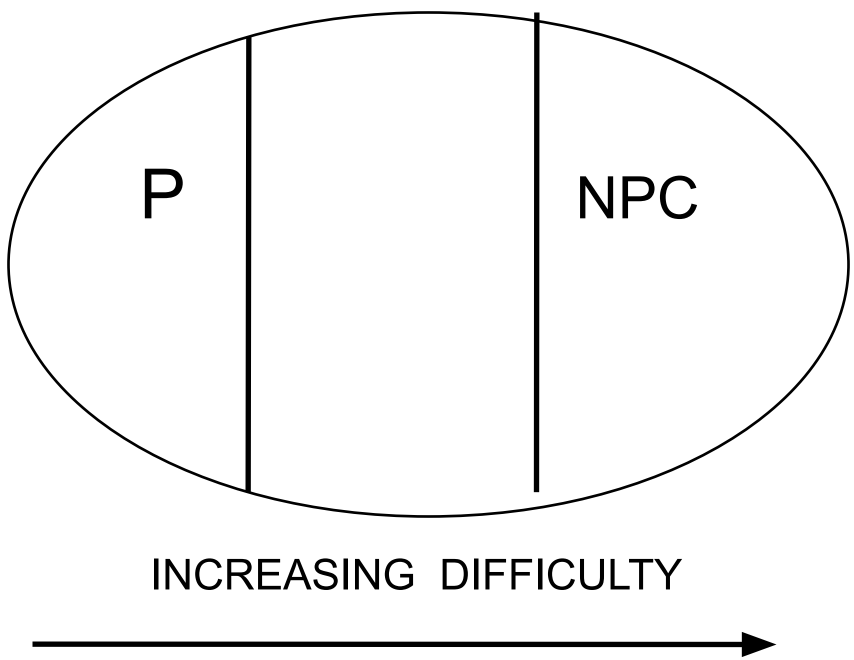
- NP – hardness follows from reduction from Hamiltonian Cycle. HOW? Assignment.

NP-Complete Problems

- The *NP-Complete* problems are an interesting class of problems whose status is unknown
  - No polynomial-time algorithm has been discovered for an NP-Complete problem
  - No supra-polynomial lower bound has been proved for any NP-Complete problem, either
- We call this the *P = NP question*
  - The biggest open problem in CS

NP-Completeness

The space NP of all search problems, assuming
$P \neq NP$



Significance of NP-Completeness

The interest surrounding the class of NP-complete problems can be attributed to the following reasons.

- No polynomial-time algorithm has yet been discovered for any NP-complete problem; at the same time no NP-complete problem has been shown to have a super polynomial-time (for example exponential time) lower bound.
- If a polynomial-time algorithm is discovered for even one NP-complete problem, then all NP-complete problems will be solvable in polynomial-time.
- It is believed (but so far no proof is available) that NP-complete problems do not have polynomial-time algorithms and therefore are intractable. The basis for this belief is the second fact above, namely that if any single NP-complete problem can be solved in polynomial time, then every NP-complete problem has a polynomial-time algorithm. Given the wide range of NP-complete problems that have been discovered to date, it will be sensational if all of them could be solved in polynomial time.

Why Prove NP-Completeness?

- Though nobody has proved that **P** != **NP**, if you prove a problem NP-Complete, most people accept that it is probably intractable
- Therefore it can be important to prove that a problem is NP-Complete
  - Don't need to come up with an efficient algorithm
  - Can instead work on *approximation algorithms*

Acknowledgment

- Shivam Sharma
- Sufyan Haroon

UP Next

- Approximation Algorithms