Computability and Complexity theory
Assignment # 4

April 18, 2023

Roy, Akash | CS22M007
M.Tech CS | Indian Institute of Technology, Madras

1. Which of the following statements are true, which are false? Give a short proof of your answers.
   (a) If NP = P, then every non-trivial language in P is NP-complete.
   (b) Let L, L' be languages that are non-trivial (i.e., neither empty nor Σ^*). If $L \subseteq L'$ and L is NP-hard, then L' is NP-hard.
   (c) There are L and L' such that $L \leq_m^P L'$ but not $L' \leq_m^P L$.

Ans: (a). The first statement is true. Every non-trivial problem in P is then polynomial time reducible to every problem in NP hence to SAT or Vertex-Cover, that is by definition NP complete.

(b). Following is the definition for NP-Hardness,

Definition 0.1: NP-Hardness

A problem A is NP-Hard if and only if all problems $y \in NP \leq_m^p$ to A.

From the definition any problem $y \in$ NP is polynomial time reducible to L. Now consider the language L' which is from question $L \subseteq L'$ has all the strings from L. So L is properly contained within L'. So any NP problem is still poly-time reducible to strings of L contained within L'. So all the problem in NP is now reducible to L'.

From definition 0.1 L' is now NP-Hard. Hence the statement is true.

(c). Seems true. If $L = L'$ then it is always false. Now otherwise if $L \leq_m^P L'$ that means there is a polynomial time computable function f that checks $x \in L \iff f(x) \in L'$. Given there exists inverse of the polynomial time computable function f^{-1} then it is entirely possible to construct $x \in L' \iff f^{-1}(x) \in L$. This gives $L' \leq_m^P L$. This means if f^{-1} is not possible to construct then this statement will hold for some language L and L'.

2. Let HamPath be the language $\{G \mid G$ has a Hamiltonian path$\}$. Using the ideas used in the class, show that HamPath is NP-complete.

To show HAMPATH is NP Complete.

We already know 3SAT is an NP Complete problem.
Enough to show $3SAT \leq_m^P$ HAMPATH

$$HAMPATH = \{ G \mid G \text{ has hamiltonian path} \}$$

Hamiltonian path is a path such that a walk along the path visits every vertex exactly once.

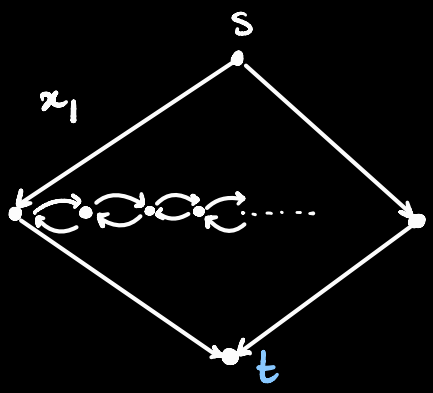
Say a reduction f.

$$\Phi = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee x_4) \wedge \dots \wedge (\dots)$$

Given an instance of Φ, our reduction f should reduce Φ to a graph G in such a way that

$\Phi = 1$ if and only if G has a hamiltonian path

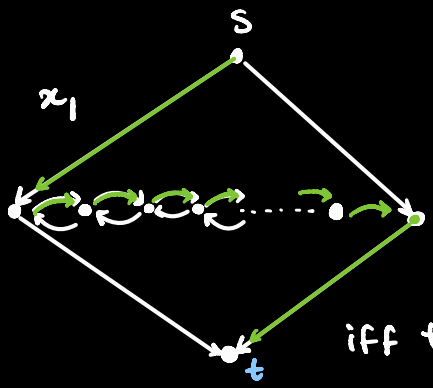
Let's define a substructure in the graph



This is a variable gadget.

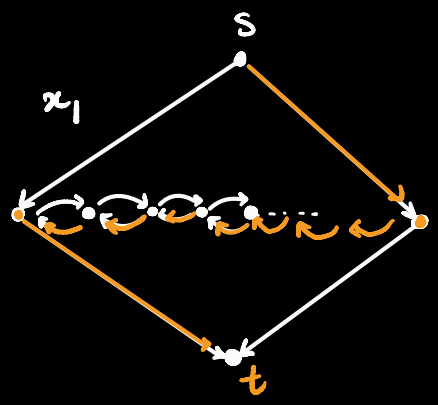
I define two walks walk A & walk B are two different type of walk on the variable gadget.

They are the following:



This is one type of walk to reach to t.

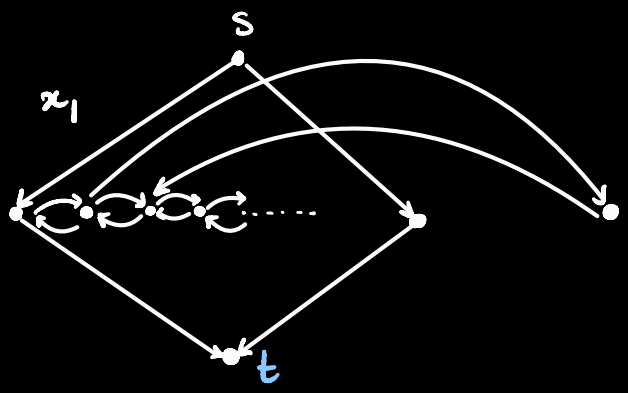
This walk is followed iff the variable is set to TRUE.



This is another type of walk to reach t.

This walk is followed for a variable gadget iff the variable is set to false.

One clause gadget is a single vertex connected with the variable gadget.



clause gadget.

Observation A key observation is that for type II walk it's impossible to cover the clause gadget. Hence hamiltonian path is not possible if x_4 is set to false.

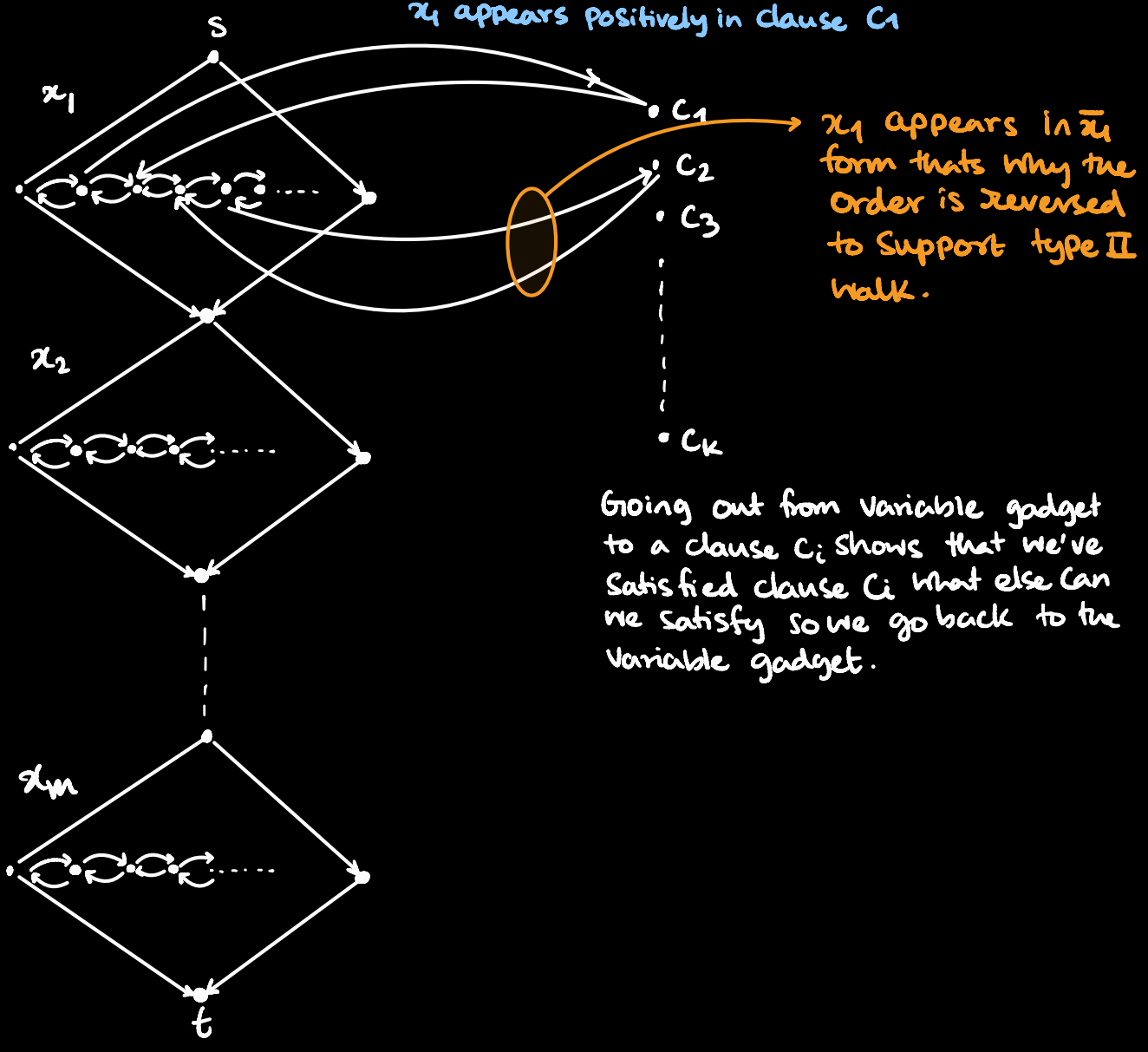
Now with the variable gadget & the clause gadget defined let's construct a G such that

$\Phi \in 3SAT \Leftrightarrow G$ has a hamiltonian path

Say Φ is a boolean formula having m variable and K clauses

$$\Phi = C_1 \wedge C_2 \wedge C_3 \wedge \ldots \wedge C_m$$
$$= (x_4 \vee \ldots) \wedge (\bar{x_4} \vee \ldots) \wedge \ldots$$

Graph G_1:



This is the total Construction of graph G_1 from Φ.

Now to Show

Φ is satisfiable iff there is a Hamiltonian path $\mathcal{H}$ in G_1.

Theorem: Φ is Satisfiable iff G has a Hampath.

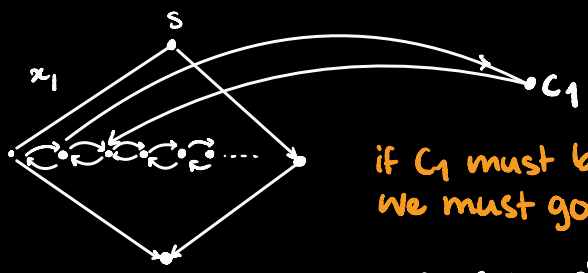
Proof: ($\Rightarrow$) forward direction is Straight forward.

Φ is satisfiable means we can now walk on the graph G, type 1 or type 2 depending upon the literal type. Then there is a clear hamiltonian path from s to t. Because for all clause to be satisfied some of the literals must [variable gadget] set to true/false & based on that we'll be able to cover the clause gadgets without duplication.

($\Leftarrow$) There is a hamiltonian path p in graph G from s to t.

Following this path p starting from s whatever way is the path assign every literal that value either true or false.

Because of the Construction of graph



if C_1 must be satisfied by x_1 then we must go left [from Observation]

means we should set the variable x_1 to true.

This is how every clause will be satisfied.

∴ above theorem is true.

This graph G from Φ can also be done using polynomial time.

Hence

$$3SAT \leq_m HAMPATH$$

∴ HAMPATH IS NP-COMPLETE.

3. Show that SAT $\in$ NP $\cap$ co-NP if and only if NP = co-NP.

Ans: SAT is a NP complete problem. This means for all problems in NP polynomial time reducible to SAT. Say arbitrary problem $x \in$ NP.

We can say that $x \leq_m^P$ SAT. For forward direction: given SAT $\in$ NP $\cap$ co-NP, to show NP = co-NP.

Theorem 0.1: co-NP completeness

For all $x \in$ co-NP, x is poly-time reducible to any co-NP complete problem, for example co-SAT.

From question SAT $\in$ NP and SAT $\in$ co-NP. As SAT $\in$ co-NP then we can say that SAT $\leq_m^P$ co-SAT (from theorem 0.1 up above).

So let's take any problem $g \in NP$, $g \leq_m^P$ SAT $\leq_m^P$ co-SAT. This implies $g \leq_m^P$ co-SAT. This implies $g \in$ co-NP. So for any $g \in$ NP g is also in co-NP. Hence NP = co-NP.

(reverse direction) Given NP = co-NP, to show SAT $\in$ NP $\cap$ co-NP. This is trivial to show. We know SAT $\in$ NP and NP = co-NP, so NP = NP $\cap$ co-NP. So SAT $\in$ NP $\cap$ co-NP.

So we can say from the statements up above SAT $\in$ NP $\cap$ co-NP $\iff$ NP = co-NP.

4. Let A be an NP-complete language and B be in P. Prove that if $A \cap B = \phi$, then $A \cup B$ is NP-complete. What can you say about the complexity of $A \cup B$ if A and B are not known to be disjoint? Justify your answers.

Ans: There is turing machine running deterministic polynomial time accepting strings in B and there is a turing machine running in non-deterministic polynomial time accepting strings in A. Say we construct a new non-deterministic polynomial time bounded turing machine for $A \cup B$ that'll accept strings from A as well as B in the following way

Algorithm 1: TURING MACHINE $M_{A \cup B}$

Input to the turing machine $M_{A \cup B}$: x

1 Run in parallel x on M_A non-deterministically // turing machine for accepting A
2 Run in parallel x on M_B // poly-time turing machine accepting B

Output from turing machine : Accept if either one accepts.

This new turing machine has capability to accept any strings of A. A is a NP complete problem, as well as B which is in class P. So any new problem $f \in NP$ I'll reduce this to A and then run on the machine $M_{A \cup B}$. This makes $A \cup B$ a NP complete problem.

No where in my solution I assumed anything about the disjoint-ness of A and B. So even if $A \cap B \neq \phi$ then also this statement holds $A \cup B$ is NP complete.

5. Show that if $\mathsf{P} \neq \mathsf{NP}$, there cannot be a polynomial time algorithm that switches a CNF formula to an DNF formula preserving satisfiability.

Ans: We should not be able to come up with a polynomial time conversion of CNF to DNF conditioned on the fact that P $\neq$ NP.

Say there is an algorithm $\mathbb{A}$ that converts a CNF formula to DNF formula in polynomial time. We know that 3-CNFSAT is NP complete. Now let's run the following algorithm POLY-SAT-Solver,

Algorithm 2: POLY-SAT-Solver

Input to the algorithm : A ϕ formula in 3-CNF form

1 $\phi_{\text{DNF}} = \mathbb{A}(\phi_{\text{CNF}})$ // Subroutine call to convert a CNF to DNF in polytime

2 Solve DNF-SAT(ϕ_{DNF}) in polynomial time as discussed in class.

Output from algorithm: True or False if ϕ_{DNF} is satisfiable

Now our algorithm for 3-CNF-SAT runs in polynomial time. But as P $\neq$ NP and 3-CNF-SAT is NP complete this means 3-CNF-SAT must not run in polynomial time. So our algorithm POLY-SAT-Solver is not possible to construct which depends upon algorithm $\mathbb{A}$, so algorithm $\mathbb{A}$ which is a poly-time CNF to DNF converter must not exist.

6. Let M be a deterministic Turing machine that only queries oracle strings that are shorter than the input string. Show that if $A = L(M^A)$ and $B = L(M^B)$ then $A = B$. (Hint: show that for all n, $A^{\leq n} = B^{\leq n}$ using induction starting with $n = 1$. Here, $A^{\leq n}$ represents set of all strings of length at most n in A.)

Ans:

Hypothesis 0.1: Induction hypothesis

$\forall x \in \{0,1\}^n$ the statement $L(M^A) = L(M^B)$ holds good.

To show this holds good for $\forall x \in \{0,1\}^{n+1}$. Enough to show for all $x \in \{0,1\}^{n+1}$ $x \in A \iff x \in B$.

For base case $x = \epsilon$ this statement holds true. Because for string ϵ the turing machine M do not call any oracle (there is nothing less than the size of ϵ string). So for $x = \epsilon$, $M^A = M = M^B$. So if $x \in L(M^A) = L(M)$ then $x \in L(M) = L(M^B)$.

Say $x \in A$ then from definition M^A accepts x because given in the question A = L(M^A). Similarly if $x \in B$ then from definition M^B accepts x because given in the question B = L(M^B).

For input $x \in \{0,1\}^{n+1}$ turing machine quires oracle in $\{0,1\}^n$. From hypothesis M^A can be replaced by M^B. For input length n A = B. So changing oracle from A to B thus replacing M^A with M^B does not change the output of M, meaning $M^A = M^B$. Thus if $x \in A$ then $x \in B$ and vice versa.

Similarly if $x \notin A$ then $x \notin B$ and vice-versa. With this we can say $x \in A \iff x \in B$. So we can say the following theorem

Theorem 0.2

$\forall x \in \{0,1\}^n$ the statement $L(M^A) = L(M^B)$ holds good if $A = L(M^A)$ and $B = L(M^B)$ where M^l is a deterministic turing machine, with access to l oracle and M only queries shorter string than the input string.