CS5110 Computational Complexity

Assignment 1

C. Akshay ~~S~~antoshi
CS21BTECH11012

1. DOUBLE-SAT = $\{<\phi> \mid \phi$ has atleast two distinct satisfying assignments$\}$

DOUBLE-SAT $\in$ NP

~~Do~~ Any $\phi \in$ DOUBLE-SAT has a polynomial time verifier ~~ge~~ with certificate as two ~~solu~~ assignments to the variables in ϕ. Given two assignments, first we have to check whether ~~ϕ is~~ those are distinct assignments and ~~then~~ if so, ~~we~~ ~~h~~ ~~can~~ for the corresponding assignments, we can check whether ϕ is satisfiable or not.
If YES, then we can ACCEPT, else REJECT.

$\therefore$ DOUBLE-SAT $\in$ NP.

SAT $\leq_p$ DOUBLE-SAT

Let $\phi' \in$ SAT
Let the variables in ϕ' be $x_1, x_2, \ldots, x_n$.
We will construct a new boolean formula by introducing a new variable x_{n+1} and defining the formula as

$$\phi = \phi' \wedge (x_{n+1} \vee \bar{x}_{n+1})$$

We can claim that $\phi \in$ DOUBLE-SAT iff $\phi' \in$ SAT.
If $\phi' \in$ SAT, we will show that ϕ has atleast two distinct satisfying assignments.
Give $x_1, x_2, \ldots, x_n$, the corresponding assignments for which ϕ' is satisfiable. The ϕ' value is TRUE.
Assign x_{n+1} as TRUE in one of the assignment.
We can see that this assignment satisfies ϕ since ϕ' is TRUE and $(x_{n+1} \vee \bar{x}_{n+1})$ is TRUE.

Now, for the second assignment, keep the values of $x_1, \dots, x_n$ same as before, change x_{n+1} to FALSE. This makes $\overline{x}_{n+1}$ as TRUE. Here again ϕ' is TRUE and $(x_{n+1} \vee \overline{x}_{n+1})$ is TRUE. So ϕ is satisfied for this assignment.

Since these are two distinct assignments ~~such that~~ for which ϕ is getting satisfied, $\phi \in$ DOUBLE-SAT.

If $\phi' \notin$ SAT, then no truth value assignments to variables $x_1, x_2, \dots, x_n$ would make ϕ' be TRUE. So irrespective of whether x_{n+1} is TRUE or FALSE, ϕ cannot be satisfied. So it doesn't have atleast two satisfying assignments. Hence, in this case we get $\phi \notin$ DOUBLE-SAT.

$\therefore$ DOUBLE-SAT is NP-COMPLETE.

---

2) Is DNF-SAT in P?

Let ϕ ~~DNF~~ $= C_1 \vee C_2 \vee \dots \vee C_m$ (Clauses have AND of literals)

We will show that DNF-SAT is in P.

Given a ϕ, you can check each clause one-by-one, and see if you can find any clause which doesn't have both x_i and $\overline{x}_i$ in it for all the literals present in that clause.

If you find such a clause, you can assign truth value as TRUE to all the literals inside that clause. Variables not included in that clause can be given truth values of our choice.

For ϕ to be true, at least one of the clause has to be TRUE. So for the previous case, since we have such a clause, ϕ becomes satisfiable and the solution is that particular truth value assignment.

If you find no such clause, then you can be sure that no assignment would satisfy ϕ since each clause has contradicting literals for some i.

If you have m clauses and each clause has atmost k literals, total time to check all clauses ~~in~~ is $O(m.k)$ time.
$\therefore$ Polynomial in the size of the formula.

$\therefore$ DNF-SAT $\in$ P.

---

3) 2-CNF SAT : $C_1 \wedge C_2 \wedge \ldots \wedge C_m$, where each C_j is OR of exactly two literals.

Is 2-CNF in P?

~~Each clause~~ Let $\phi \in$ 2-CNF SAT

Each clause in ϕ would be of the form $(A \vee B)$

For ϕ ~~be~~ to be satisfiable, all clauses must become TRUE for a assignment.

Consider a clause $(A \vee B)$

$A \vee B$ becomes TRUE when atleast one of A or B is TRUE.

(a) If A is false, B has to be TRUE.

(b) If B is false, A has to be TRUE.

Statement (a) can be shown by $\bar{A} \rightarrow B$

Statement (b) can be shown by $\bar{B} \rightarrow A$.

Express every clause using these two implications.

If ~~two of~~ any two or more implications of the clauses, have a common literal, for e.g., $\bar{A} \rightarrow B$ (from one clause) and ~~B→C~~ $B \rightarrow \bar{C}$ (from another clause), you can visualize this in a combined way like ~~that~~ the one below:

$$\bar{A} \rightarrow B \rightarrow \bar{C} \rightarrow \ldots$$

If for any variable, you get a chain like the one below:

$$\bar{A} \rightarrow B \rightarrow \bar{C} \rightarrow \ldots \rightarrow A.$$

it forces that $\bar{A}$ has to be FALSE (or A is TRUE), since both $\bar{A}$ and A appear in the same chain. If $\bar{A}$ was TRUE, it would lead to both $\bar{A}$ and A being TRUE at the same time which is a contradiction.

Another possibility, is for any variable, you end up getting a chain like the one below:

$$A \rightarrow B \rightarrow C \rightarrow \ldots \rightarrow \bar{A}.$$

In this case, it forces A to be ~~TRUE~~. FALSE.

(Note that, we ~~can~~ don't include cycles, since we can break the chain once we come across the same literal).

Now, we can clearly see that the formula ϕ is satisfiable iff there is no variable having the two chains simultaneously. (i.e, from A to $\bar{A}$ and from $\bar{A}$ to A).

~~Poly time Algo~~:

Construct a graph G with nodes as variables of ϕ along with their negations. So if the graph has n variables, you would have 2n nodes in total.

For every clause $(A \vee B)$, add two directed edges to the graph, with one edge from $\bar{A}$ to B and another from $\bar{B}$ to A.

Algo:

For each variable x,

Check if there is a path from x to $\bar{x}$

If no, assign x as TRUE.

If yes,

Check if there is a path from $\bar{X}$ to X.

If no, assign X as false.

If yes, return REJECT (indicating unsatisfiability)

RETURN ACCEPT (indicating satisfiability).

You can do a DFS Search for each connected component to check if the path A to $\bar{A}$ and $\bar{A}$ to A exist in the graph G.

DFS Search takes poly. time.

4) If $P=NP$, which are the languages that are NP-complete?

~~If~~ $A \in$ NP-complete if

(1) $A \in NP$ and

(2) $\forall B \in NP \quad B \leq_p A$

Trivial decision problems that are always true or always false independent of the input are not NP-complete.
They are not NP-complete.
Trivial problems have $O(1)$ time T.M but there does not exist a reduction f such that for $A \in NP$ and B being a trivial problem $x \in A \iff f(x) \in B$.
Suppose B is universal, then if $x \notin A$, we don't have a mapping for x in B since $\overline{B} = \phi$, there are no strings outside B. (The T.M for B accepts everything.)
Suppose B is empty, then if $x \in A$, we don't have a mapping for it since $B = \phi$ and it accepts no string.

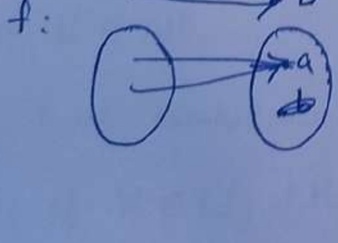
If $P=NP$, then every problem in NP can be solved in poly-time. So even NP-complete problems are solvable in poly-time.
Let $L \in$ NP-complete

$\forall A \in NP \quad A \leq_p L$ and $L \in P$.

Since $P=NP$, all NP-complete languages are in P.

~~If~~ All $P=NP$ ~~problems~~ languages (except for the trivial languages) are in NP-complete if $P=NP$.

We can create reductions from $A \in NP$, $B \in$ NP-complete $\in P$



For strings in A, it gets mapped to a particular ~~B∈B~~ string 'a' $\in B \in P$.
For strings outside B, it gets mapped to a particular string 'b' $\notin B$.

5. Show that if P=NP, there is a polynomial time algo. to find a satisfying assignment to a 3-SAT formula if such an assignment exists.

~~If P=NP~~ Let $\phi \in$ 3-SAT formula having variables $x_1, \dots, x_n$.
Now, we want to find the truth value assignments to these variables so that ϕ becomes satisfiable.

Since P=NP, there exists a T.M M which determines whether ϕ is satisfiable or not in poly-time.

Let us start with unitialized truth values for the variables.

Let $\phi_0 = \phi$

For i=1 to n

$\phi_c = \phi_{i-1}$

Run M with input as $(\phi_{i-1} \wedge x_i)$ {This is used for testing whether x_i is T}

If M outputs ACCEPT, then assign x_i to be TRUE. and ~~add~~ update ϕ_i to $\phi_{i-1} \wedge x_i$.

If M outputs REJECT, then assign x_i to be FALSE and update ϕ_i to $\phi_{i-1} \wedge \overline{x}_i$

Return truth values of $x_1, x_2, \dots, x_n$.

Adding constraints to ϕ incrementally helps in progressively building a satisfying assignment.

Since there is a loop that runs k times and a T.M M that is a poly-time decider, we have an algo that runs in poly-time and returns a satisfying assignment for $\phi \in$ 3-SAT.

6. Show that $A \le_p B$ and $B \le_p C \Rightarrow A \le_p C$.

If $A \le_p B$, then there exists a poly-time computable function $f : \Sigma^* \longrightarrow \Sigma^*$ s.t for every $x \in \Sigma^*$

$$x \in A \iff f(x) \in B \quad \text{——— (1)} \qquad O(n^{k_1})$$

If $B \le_p C$, then there exists a poly-time computable function $g : \Sigma^* \to \Sigma^*$ s.t for every $y \in \Sigma^*$

$$y \in B \iff g(y) \in C \quad \text{——— (2)} \qquad O(n^{k_2})$$

Define a new function $h : \Sigma^*$ to Σ^* as $h(x) = g(f(x))$.

From (1), if $x \in A \iff f(x) \in B$

From (2), if $f(x) \in B \iff g(f(x)) \in C$.

Combining we get, $x \in A \iff \underbrace{g(f(x))}_{h(x)} \in C$ $\quad O((n^{k_1})^{k_2}) = O(n^{k_1 k_2})$

$h(x) \in C$.

The composition of two functions takes poly time and both f and g are computable in poly time, so we have h also computable in poly-time.

$$h(x) = g(f(x))$$

$x \in A \iff h(x) \in C$ in poly time.

$\therefore A \le_p C$.

---

7. Show that a language L is CO-NP complete if and only if $\bar{L}$ is NP-complete.

If ~~$\bar{L}$ is~~ L is ~~co~~NP complete, then $\bar{L}$ is co-NP-complete:

Let $\bar{L'} \in$ CO-NP.

$\Rightarrow L' \in$ NP.

Since L is ~~co~~NP complete, we have $L' \le_p L$

So, if $x \in L'$, then $f(x) \in L$

Let f be reduction from L' to L.

$x \in L'$ iff $f(x) \in L$

$\Rightarrow x \in \bar{L'}$ iff $f(x) \in \bar{L}$.

$\Rightarrow f$ is a reduction from $\bar{L'}$ to $\bar{L}$.

We started with any $\bar{L'}$ belong to co-NP and showed that

$\bar{L'} \leq_P \bar{L}$ (L is NP-complete, so $L \in NP$, $\Rightarrow \bar{L} \in$ co-NP)

$\therefore \bar{L}$ is co-NP complete.

If L is co-NP complete, then $\bar{L}$ is NP-complete:

Let $\bar{L'} \in NP$

$\Rightarrow L' \in$ co-NP.

Since L is co-NP complete we have $L' \leq_P L$

Let f be reduction from L' to L.

$x \in L'$ iff $f(x) \in L$.

$\Rightarrow x \in \bar{L'}$ iff $f(x) \in \bar{L}$

$\Rightarrow f$ is a reduction from $\bar{L'}$ to $\bar{L}$

We started with any $\bar{L'}$ belong to NP and showed that

$\bar{L'} \leq_P \bar{L}$ (L is co-NP complete, so $L \in$ co-NP, $\Rightarrow \bar{L} \in NP$)

$\therefore \bar{L}$ is NP complete.

---

8. Show that NP $\neq$ co-NP $\Rightarrow$ P $\neq$ NP.

If P = NP $\Rightarrow$ NP = co-NP.

Consider $L \in$ co-NP

$\Rightarrow \bar{L} \in NP$.

Since P = NP, $\bar{L} \in P$.

$\Rightarrow L \in P$ (since complement of language in P also lies in P).

We got co-NP $\subseteq$ P. We already know that P $\subseteq$ co-NP.

$\therefore$ co-NP = P. We already have P = NP. So, co-NP = P = NP.

9. EXACT-CLIQUE = $\{<G,k> \mid$ G is an undirected graph and k is a natural no. such that the largest clique in G is of size exactly k $\}$.

EXACT-CLIQUE $\in \Sigma_2 \wedge \Pi_2$.

Σ_2 : $L \in \Sigma_2$, $x \in L \iff \exists y, \forall z$ s.t $|y|, |z| \in poly(|x|)$, $V(x,y,z) = 1$.

$(G,k) \in$ EXACT-CLIQUE

$\iff \exists$ a clique C of size k $\forall$ cliques $C' \in G$, s.t size of $C' \leq k$.

Π_2 : $L \in \Pi_2$: $x \in L \iff \forall y, \exists z$ s.t $|y|, |z| \in poly(|x|)$, $V(x,y,z) = 1$.

$(G,k) \in$ EXACT-CLIQUE

$\iff \forall S \subseteq V(G), |S| > k, \exists S' \subseteq V(G)$ s.t ((S' is a clique of size $\leq k$) $\vee$ (S' is not a clique))

$\therefore$ EXACT-CLIQUE $\in \Sigma_2 \wedge \Pi_2$.