CS6014 : Advanced Theory of Computation — Author: Karthik Abinav(CS10B057)

Problem Set #5 — Collaborators:Saikrishna

1. 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?

Solution:

$A \cup B$ belongs to NP. Because given an x we have to check if $x \in B$ or $x \in A$. And to check if $x \in A$ takes NP time.
$A \cup B$ is NP-hard

To show this we have to show that $\forall L \in \mathsf{NP}\ L \leq_m^p A \cup B$. i.e. $x \in L \Leftrightarrow \sigma(x) \in A \cup B$. If we construct such a σ computable in polynomial time we are done.

Since A is NP complete, there exists a σ_1 computable in polynomial time such that, $x \in L \Leftrightarrow \sigma_1(x) \in A$.

Construct the σ such that, $\sigma(x) = \sigma_1(x)$ if $\sigma_1(x) \notin B$.
$\sigma(x) = fixed\ string\ w$ such that $w \notin A$ and $w \notin B$.

Since, $A \cap B = \varnothing$. Hence, $\sigma(x) \in B \Leftrightarrow \sigma(x) \notin A$.
Therefore, $\sigma(x)$ satisfies the reduction for L to $A \cup B$.

Suppose, it is not known about $A \cap B$, then the above reduction wont work. We can just conclude that $A \cup B \in \mathsf{NP}$ and cant comment about the hardness. (Unless $A \cap B = A$ or $A \cap B = B$, in which cases $A \cup B$ will be NP-complete and P respectively).

2. Show that a DNF formula can be converted in polynomial time to a CNF formula, with possibly more number of variables , preserving satisfiability. 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.

Solution:
Part a
The formula ϕ in DNF will be as $P_1 \vee P_2 \vee P_3 \ldots$. Every P_i will be of the form

$x_1 \wedge x_2 \wedge x_3$.. Now , take every clause of the form $P_i \vee P_j$.
If either P_i or P_j is a variable, then directly distribute the expression over the $\vee$ to get an expression in CNF.
Else, introduce a new symbol y_i and make the clauses $(y_i \vee P_i) \wedge (\overline{y_i} \vee P_j)$.
This will preserve the satisfiability of the $(P_i \vee P_j)$ expression as follows:

- If both P_i and P_j is false, putting any value to y_i will preserve satisfiability.
- Similarly if both P_i and P_j is true, then putting any value to y_i will preserve satisfiability.
- Suppose one of them is true and other is false. WLG let P_i be true. Then putting y_i as true will preserve satisfiability.

Now, since y_i is a variable, the subclauses can now be distributed in the same way as above. This will take atmost length of P_i time. And total time taken to convert will be |number of clauses| $*$ |Time taken to expand each clause| . And this will take polynomial time in length of the expression.

Part b
We can show this by proving the contrapositive of the statement. i.e. If there exists a polynomial time conversion from CNF to DNF, then P = NP.

Consider, any formula in the CNF. Now, use the algorithm and convert it into a DNF. The formula will be of the form, $P_1 \vee P_2$.. and having the same satisfiability as the CNF. Now, scan through each of the P_i and see if any clause has both the variables x_i and $\overline{x_i}$. If not, assign to all variables in that clause of form x_j a true and $\overline{x_j}$ a false value and for the variables not in that clause any value.Then report satisfiable. Else, if there is no such clause, report unsatisfiable.
Hence, SAT problem can be solved in polynomial time, since the scanning takes atmost the length of the clause. And since, SAT is NP-Complete, Every problem in NP can be decided in polynomial time. And hence P = NP.

3. Give a polynomial time algorithm for 2-SAT problem that we stated in class. Given a CNF formula ϕ, where each clause has atmost two literals, test satisfiability. Is 2-SAT in NL? Argue.

Solution: Consider any formula ϕ in the $2 - \mathsf{SAT}$ form.
It will be of the form $(x_1 \vee x_2) \wedge (x_3 \vee x_4)$...
Like the previous question, the technique will be to reduce formulas of the form $(x_1 \vee x_2) \wedge (\overline{x_1} \vee x_3)$ to $(x_2 \vee x_3)$. The consistency in satisfiability follows from

distributing the $\vee$ over the $\wedge$ and using the claim in question 2.

Now, the algorithm will scan through the formula for pairs of clauses having x_i in one and $\overline{x_i}$ in the other and reduce them. There can be atmost $O(n^2)$ such pairs for each variable. Next replace all clauses having $(x_i \vee \overline{x_i})$ with a 0. Now , initial formula is satisfiable, if the new obtained formula is not a fallacy(i.e. 0).

The running time of the algorithm takes polynomial time in the number of variables times the number of clauses.

4. Consider the complexity class DP (D stands for *difference*) as the set of problems L such that $L = A \cap B$ where A and B are two languages in NP and coNP respectively. Argue that $\mathsf{DP} \subseteq \mathsf{P}^{\mathsf{NP}}$. Argue that Exact-Clique is DP-complete.

Solution: Part a
Let $L \in \mathsf{DP}$. Implies there exists A and B such that $A \cap B = L$. Now construct an OTM with a SAT as an oracle. The description of the machine is as follows:

- Given an input x, checks if $x \in \mathsf{A}$ by doing the poly time reduction to SAT via a function σ_1 and queries the oracle for $\sigma_1(x) \in \mathsf{SAT}$.
- checks if $x \notin \overline{B}$ by computing the poly time reduction to SAT via the function σ_2 and queries the oracle. If $x \notin \overline{B} \implies x \in B$ because B is coNP.
- The machine accepts if both queries give accept. Else, rejects.

Since, the reductions are poly time, this OT machine also runs in polynomial time. Hence, $L \in \mathsf{DP} \implies L \in \mathsf{P}^{\mathsf{NP}}$. Therefore, $\mathsf{DP} \subseteq \mathsf{P}^{\mathsf{NP}}$.
Part b
To check $L \in$ Exact-Clique we have to check two parts - $(G, k) \in$ Clique and $(G, k+1) \notin$ Clique Define $clique_k$ as set of all graphs with a clique of size k. and similarly, $clique_{k+1}$ as set of all graphs with a clique of size k+1. exact-clique $\in \mathsf{DP}$. The set $A = clique_k$ and the set $B = clique_{k+1}$. And A is NP complete and B is coNP complete.
$\forall L \in \mathsf{DP}$ There exists a σ_1 computable in poly time(via machine M_1) such that, $x \in L \Leftrightarrow \sigma_1(x) \in A$ and a poly time computable(via machine M_2) σ_2 such that , $x \in L \Leftrightarrow \sigma_2(x) \in B$. (Because A and B are NP complete and coNP complete respectively).

Construct a machine N which simulates M_1 and M_2. Given an input y, it computes σ_1 by simulating M_1 and checks if $\sigma_1(y) \in A$.

Then computes σ_2 by simulating M_2 ad checks if $\sigma_2(y) \in B$.
If both are true accepts, else rejects.
Since both these machines run in poly time the machine N also runs in poly time.Also, the language accepted by N is EXACT-CLIQUE . This shows that EXACT-CLIQUE is DP complete.

5. Cook-Levin Theorem is proved by reducing any $L \in \mathsf{NP}$ to SAT. Is there any relation between the number of accepting paths of the non-deterministic machine (deciding L) and the number of satisfying assignments of the formula produced by the reduction? Check the same for the NP-completeness proof for INDEPENDENT SET PROBLEM that we did in class.

Solution:
In the proof we did in class, we constructed the clause in SAT by considering every bit in the configuration from start to end as a variable, and constructed a clause whose truth value will be 1 exactly when the sequence of configurations will be accepting one. Hence, the number of accepting configurations is precisely the number of satisfying assignments to the constructed SAT clause.

In case of the INDEPENDENT SET PROBLEM, we choose a subset(S) of vertices such that,
$x_i \in S$ if $x_i = 0$ in the assignment to $3 - \mathsf{SAT}$.And all the clauses which evaluate to 1 were also added into S.
Hence, the number of sets S is precisely the number of sets i's for which $x_i = 0$. This is the number of satisfiable assignments for $3 - \mathsf{SAT}$. This is twice the number of satisfiable assignments for SAT. Because, suppose the clause in SAT was of the form, $x_1 \vee x_2 \vee x_3$, then it is converted to $3 - \mathsf{SAT}$ as $(x_1 \vee x_2 \vee y_1) \wedge (\overline{y_1} \vee x_3 \vee y_2)$. And for a fixed values of x_i s, there is a choice on the last variable y_n to be either 0 or 1(in the above example y_2). Hence, the number of independent sets is twice the number of accepting paths in the non deterministic machine.