# Homework 2

**3.1.** Prove that one can check the satisfiability of a 2-CNF (a conjunction of disjunctions, each containing 2 literals) in polynomial time.

**1.** SAT, also known as Boolean satisfiability problem, is a problem of assigning boolean values to variables to satisfy a given boolean formula. Formula usually would be in CNF form (conjunctive normal form). This is a formula having conjunction of multiple clauses, where each clause is a disjunction of literals. Each literal is a boolean variable. 2-SAT (2-satisfiability) is a restriction of the SAT problem, in 2-SAT every clause has atmost up to two literals.

$$(a \vee \neg b) \wedge (\neg a \vee b) \wedge (\neg a \vee \neg b) \wedge (a \vee \neg c)$$

**2.** Idea is to use the resolution method to convert given propositional logic (CNF), $F$, to equivalent set of disjunction clauses $\overline{F}$. Once we get the disjunctions($\overline{F}$), we look for a clause that is false to know if the given CNF is unsatisfiable. By the rule of contradiction, we check the satisfiablity by looking for false disjunctions in the given formula. For a 2-CNF, resolution method will give only clauses with 2-literals or 1-literal or empty set and this conversion is of complexity order Linear $O(n+m)$, where $n$ is number of literals, $m$ is number of clauses.

**3.** $F$ is not satisfiable if and only if $\overline{F}$ contains the empty clause. We call a clause, in a CNF, empty clause if it is false. To figure out the empty clause, first we need to convert the problem to a different form, the so-called implicative normal form. We create a implication graph for our formula. For each literal $x$, there will be two vertices ($x$, $\neg x$) as its edge. For the example in the point 1, below are their implications.

| | | | |
|---|---|---|---|
| $\neg a \Rightarrow \neg b$ | $a \Rightarrow b$ | $a \Rightarrow \neg b$ | $\neg a \Rightarrow \neg c$ |
| $b \Rightarrow a$ | $\neg b \Rightarrow \neg a$ | $b \Rightarrow \neg a$ | $c \Rightarrow a$ |

**4.** Using the above implications we can draw the implication graph. 2-CNF formula $F$ is unsatisfiable if and only if there exists a literal $x$ such that $x$ is reachable from $\neg x$ and $\neg x$ is reachble from $x$ in the implication graph $G_F$. From the implications from point 3, we can say there is not direct path from $x_i$ to $\neg x_i$ or vice versa in the set of variables $(a, b, c)$. That is no direct path between $(a \to \neg a), (b \to \neg b), (c \to \neg c)$. That said, the example 2-CNF is satisfiable.

**5.** To determine the paths between vertices of a Graph G, we compute *adjacencymatrices* $A(G)$ for Graph $G$. Then we compute matrix $B^k = A \vee I$ where $I$ is the identity matrix. We can define the operations $\vee$ and $\wedge$ on Boolean matrices with usual matrix addition and multiplication respectively.

**6.** $B^k$ corresponds to the existence of a path of length at most k. If there is a path between $x$ and $\neg x$, then there is a path of length $\leqslant n$, where $n$ is the number of vertices. This computation is of complexity order $O(n^3 logn)$.

**7.** Both the computations for resolution method and Adjacency matrices, $O(n+m)+O(n^3 logn)$ is of the format $cn^d$. Thus we can say 2-CNF can be computed in polynomial time.

**3.3.** Suppose we have an NP -*oracle* – a magic device that can immediately solve any instance of SAT problem for us. In other words, for any propositional formula the oracle tells whether it is satisfiable or not. Prove that there is a polynomial-time algorithm that finds a satisfying assignment to a given formula by making a polynomial number of queries to the oracle. (A similar statement is true for the Hamiltonian cycle: finding a Hamiltonian cycle in a graph is at most polynomially harder than checking for its existence)

**1.** We have the NP - *oracle* with us which tells us if a given SAT propositional formula, $F$, is satisfiable or not. Let the set of variables in our formula be $x_1, x_2, ...., x_n$ and $x_i$ be any $i$th variable in the given set. Below are the steps we follow to get the assignment for a satisfiable forumula.

**Step 0:** Verify the formula $F$ is Satisfiable or not. If not, we do not have to look for assignment. Stop looking. If yes proceed.

**Step 1:** If $F$ is satisfiable, select the variable $x_i$ where $i = 0$. Assign $x_0 = True$ and substitute in the formula $F$. Let $\overline{F}$ be the new formula where $\overline{F} = F \wedge x_i$. Check $\overline{F}$ for satisfiablity using $NP - oracle$. If $\overline{F}$ is satisfiable let $F = \overline{F}$ and repeat the step for next variables $x_1, x_2...x_n$. If $\overline{F}$ is not satisfiable, proceed to step2.

**Step 2:** If $\overline{F}$, from step1, is not satisfiable, assign $x_0 = False, \overline{F} = F \wedge \neg x_i$ and $F = \overline{F}$ and proceed for next variables $x_1, x_2...x_n$.

**Step 3:** Using above steps, we determine values for all the variables such that $F$ is satisfiable, to get the final satisfying assignment for $F$.

**2.** We can see from above steps, we query $(n + 1)$ times which is of the polynomial time.

**3.8.** Prove that the predicate "x is the binary representation of a composite integer" belongs to NP.

**1.** A number $S$ is composite if it has a factor $t$ and $t < s$. A non-deterministic Turing machine can choose factors such that there exists $1 < t < S$ whose multiplication of the factors is our number S.

**2.** As multiplication can be done in polynomical time, we can say this process is of the below form and can say that this predicate belongs to class NP

$$L(x) = \exists y((|y| < q(|x|)) \wedge R(x, y))$$
$$L(x) \in NP$$