Com S 311 Section B
Introduction to the Design and Analysis of Algorithms

Lecture One for Week 13

Xiaoqiu (See-ow-chew) Huang

Iowa State University

April 20, 2021

CIRCUIT-SAT is in NP

The 3-CNF-SAT is defined as follows:

3-CNF-SAT
$= \{ < C >$: C is a satisfiable boolean formula in 3-CNF $\}$.

Theorem 34.10
3-CNF-SAT is NP-complete.

Proof

The algorithm for SAT can be used to verify 3-CNF-SAT, so 3-CNF-SAT is in NP.

Next, we show that $\mathrm{SAT} \leq_P \mathrm{3\text{-}CNF\ SAT}$.

Let $\theta(x_1, x_2, ..., x_n)$ be a boolean formula with n variables.

If the formula contains a clause such as the OR of several literals, we use associativity to parenthesize the expression fully so that each operator in the reulsting formula has 1 or 2 operands.

For example,
$\theta(x_1, x_2, x_3, x_4) = (\neg x_1 \vee x_2 \vee \neg x_3 \vee x_4) \wedge ((x_1 \leftrightarrow x_2) \vee x_3)$
$= (\neg x_1 \vee (x_2 \vee (\neg x_3 \vee x_4))) \wedge ((x_1 \leftrightarrow x_2) \vee x_3)$

Proof

As in the proof for Theorem 34.9, we introduce a variable y_i for the output of each operation.

Then we rewrite the formula as the AND of the output variable for the last operation and a conjuction of clauses for each operation in the formula.

Proof

The resulting expression for the above example is

$$\delta(x_1, x_2, x_3, x_4, y_1, y_2, y_3, y_4, y_5, y_6) =$$

$$y_6 \wedge (y_1 \leftrightarrow (\neg x_3 \vee x_4))$$

$$\wedge (y_2 \leftrightarrow (x_2 \vee y_1))$$

$$\wedge (y_3 \leftrightarrow (\neg x_1 \vee y_2))$$

$$\wedge (y_4 \leftrightarrow (x_1 \leftrightarrow x_2))$$

$$\wedge (y_5 \leftrightarrow (y_4 \vee x_3))$$

$$\wedge (y_6 \leftrightarrow (y_3 \wedge y_5)).$$

Proof

Let θ' deonte the resulting boolean formula after applying the above step; each clause in θ' has at most three lierals.

Based on the truth table for the first clause $\theta(1)$, we obtain that

$$\neg\theta(1) = (y_1 \wedge x_3 \wedge \neg x_4) \vee (\neg y_1 \wedge x_3 \wedge x_4) \vee (\neg y_1 \wedge \neg x_3 \wedge x_4) \vee (\neg y_1 \wedge \neg x_3 \wedge \neg x_4).$$

By negating and applying DeMorgan's laws, we obtain the CNF formula

$$\theta(1) = (\neg y_1 \vee \neg x_3 \vee x_4) \wedge (y_1 \vee \neg x_3 \vee \neg x_4) \wedge (y_1 \vee x_3 \vee \neg x_4) \wedge (y_1 \vee x_3 \vee x_4).$$

Now $\theta(1)$ is converted into a CNF formula with each clause containing at most three literals.

Every other clause in $\delta(x_1, x_2, x_3, x_4, y_1, y_2, y_3, y_4, y_5, y_6)$ is converted into a CNF formula in the same way.

y_1	x_3	x_4	$\theta(1) = (y_1 \leftrightarrow (\neg x_3 \vee x_4))$
1	1	1	1
1	1	0	0
1	0	1	1
1	0	0	1
0	1	1	0
0	1	0	1
0	0	1	0
0	0	0	0

Proof

Now each clause C_i in the resulting CNF formula θ'' contains at most three literals.

If C_i contains three distinct literals, then keep it.

If C_i contains two distinct literals, that is, $C_i = (t_1 \vee t_2)$, then replace C_i by the following equivalent formula,

$(t_1 \vee t_2 \vee p) \wedge (t_1 \vee t_2 \vee \neg p)$.

If C_i contains one distinct literal t, then replace C_i by the following equivalent formula,

$(t \vee p \vee q) \wedge (t \vee p \vee \neg q) \wedge (t \vee \neg p \vee q) \wedge (t \vee \neg p \vee \neg q)$.

We conclude that the original boolean formula θ is satisfiable if and only if the resulting 3-CNF formula θ''' is satisfiable.

Proof

Next we show that the construction can be done in polynomial time.

Constructing θ' from θ introduces at most 1 variable and 1 clause per connective in θ.

Constructing θ'' from θ' intruduces at most 8 clauses into θ'' for each clause of θ', which has at most 3 variables and the truth table for each clause has at most $2^3 = 8$ rows.

Constructing θ''' from θ'' introduces at most 4 clauses into θ''' for each clause of θ''.

Thus, the size of the resulting formula θ''' is polynomial in the length of the orignal formula, and can be constructed in polynomial time.

Other NP-Complete Problems

NP-complete problems arise in diverse domains: boolean logic, graph theory, biology, chemistry, physics, and many other areas.

We will use the reduction methodology to show that several problems in graph theory are NP-complete.

Boolean formulas and graphs are powerful systems for expressing various types of computation and relationship.

The Clique Problem

A **clique** in an undirected graph $G = (V, E)$ is a complete subgraph $G' = (V', E')$, with $V' \subseteq V$ and $E' \subseteq E$.

The property that G' is a complete subgraph means that for each pair of vertices u, v in V', the edge (u, v) is in E'.

The **size** of a clique is the number of vertices it contains.

The **clique problem** is the optimization problem of finding a clique of maximum size in a graph.

A decision version of this problem is to determine whether a clique of a given size k exists in a graph.

The problem is defined as the formal language
$\mathrm{CLIQUE} = \{< G, k >:$
$\quad G \text{ is a graph containing a clique of size } k\}.$

The Clique Problem is in NP

We show that CLIQUE is in NP.

We design a polynomial-time for verifying whether a given graph $G = (V, E)$ contains a clique of size k.

A certificate to the algorithm is a k-vertex subset V' of V.

The algorithm determines whether V' is a clique by checking if, for each pair of vertices u, v in V', the edge (u, v) is in E.

If so, the algorithm reports 1. Otherwise, it reports 0.

The Clique Problem is NP-Hard

Next, we show that $\mathrm{3\text{-}CNF\ SAT} \leq_P \mathrm{CLIQUE}$.

Let $\theta = C_1 \wedge C_2 \wedge \ldots \wedge C_k$ be a boolean formula in 3-CNF with k clauses, an instance of $\mathrm{3\text{-}CNF\ SAT}$.

Our reduction algorithm transforms θ into a graph $G = (V, E)$ such that θ is satisfiable if and only if G contains a clique of size k.

For $r = 1, 2, \ldots, k$, each clause C_r has three literals, that is, $C_r = t_1^r \vee t_2^r \vee t_3^r$.

For each C_r, the reduction algorithm adds a triple of three vertices v_1^r, v_2^r, and v_3^r to V.

The Clique Problem is NP-Hard

The reduction algorithm creates an edge between two vertices v_i^r and v_j^s if

they are in different triples, that is, $r \neq s$, and

their corresponding literals are consistent, that is, t_i^r is not the negation of t_j^s.

The reduction algorithm constructs the graph in polynomial time.

The Clique Problem is NP-Hard

For example, consider a boolean formula with three clauses in 3-CNF:
$\theta = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)$.

The vertex set V contains 3 triples of 3 vertices each (9 vertices):

$V = \{v_1^1, v_2^1, v_3^1, v_1^2, v_2^2, v_3^2, v_1^3, v_2^3, v_3^3\}$

The edge set E contains the following edges:

$$\begin{aligned} E = \{&(v_1^1, v_2^2), (v_1^1, v_3^2), (v_1^1, v_2^3), (v_1^1, v_3^3), \\ &(v_2^1, v_1^2), (v_2^1, v_2^2), (v_2^1, v_3^2), (v_2^1, v_1^3), (v_2^1, v_3^3), \\ &(v_3^1, v_1^2), (v_3^1, v_2^2), (v_3^1, v_1^3), (v_3^1, v_2^3), \\ &(v_1^2, v_1^3), (v_1^2, v_2^3), (v_1^2, v_3^3), \\ &(v_2^2, v_1^3), (v_2^2, v_1^3), (v_2^2, v_3^3), \\ &(v_3^2, v_1^3), (v_3^2, v_2^3), (v_3^2, v_3^3)\}. \end{aligned}$$

The Clique Problem is NP-Hard

The example boolean formula is true under assignment: $x_1 = 1$, $x_2 = 0$, and $x_3 = 0$.

The example graph G contains a clique of 3 vertices corresponding to the 3 literals:

$V' = \{v_1^1, v_2^2, v_3^3\}$, and

$E' = \{(v_1^1, v_2^2), (v_1^1, v_3^3), (v_2^2, v_3^3)\}$.

The Clique Problem is NP-Hard

We show that the contruction is a reduction.

Suppose that θ has a satisfying assignment.

Then each clause contains at least one literal t_i^r that is assigned 1, and this literal corresponds to a vertex v_i^r.

Selecting one such true literal from each of the k clauses yields a set V' of k vertices.

For any two vertices v_i^r, v_j^s in V', where $r \neq s$, both corresponding literals t_i^r and t_j^s map to 1 under the satisfying assignment.

Thus, the literals cannot be complements. By the construction of G, the edge (v_i^r, v_j^s) is in E. Thus, V' is a clique.

The Clique Problem is NP-Hard

Next, suppose that G has a clique V' of size k.

Because no edges in G connect vertices in the same triple, V' contains one vertex per triple.

Thus, each clause in θ contains a literal t_i^r that corresponds to a vertex v_i^r in V'.

Because G contains no edges between inconsistent literals, we can assign 1 to each such literal t_i^r.

Under this assignment, each clause is satisfied, and θ is satisfied.