CS 530: Theory of Computation

Time Complexity (Cont.)

A **clique** in an undirected graph is a subgraph, wherein every two nodes are connected by an edge. A **k-clique** is a clique that contains k nodes. The clique problem is to determine whether a graph contains a clique of a specified size. Let $CLIQUE = \{\langle G\rangle, \langle k\rangle | G$ is an undirected graph with a k-clique $\}$. An **independent set** in in an undirected graph is a subgraph, wherein no two nodes are connected by an edge. The problem is to determine whether a graph contains a clique of a specified size. Let $INDEPENDENT-SET = \{\langle G\rangle, \langle k\rangle | G$ is an undirected graph with an independent set of size $k\}$, where $\langle k\rangle$ is an encoding of a positive integer (base 2).

Theorem 1 (Theorem 7.24, page 268) *$CLIQUE$ is in NP.*

In $SUBSET-SUM$ problem we have a collection of numbers, $x_1, ..., x_k$ and a target number t. We want to determine whether the collection contains a subcollection that adds up to t. Thus, we define the language (where integers are represented in binary): *SUBSET-SUM*$= \{\langle S\rangle, \langle t\rangle | S = \{x_1, ..., x_k\}$ and for some $\{y_1, ..., y_l\} \subseteq \{x_1, ..., x_k\}$, we have $\sum y_i = t$. Notice that $\{y_1, ..., y_l\}$ and $\{x_1, ..., x_k\}$ are considered to be **multisets** and so allow repetition of elements.

Theorem 2 (Theorem 7.25, page 269) *$SUBSET-SUM$ is in NP.*

One important advance on the P versus NP question came in the early 1970s with the work of Stephen Cook and Leonid Levin. They discovered certain problems in NP whose individual complexity is related to that of the entire class. If a polynomial time algorithm exists for any of these problems, all problems in NP would be polynomial time solvable. These problem are called **NP-complete**.

The first NP-complete problem that we present is called the **satisfiability problem**. Recall that variables that can take on the values TRUE and

FALSE are called **Boolean variables**. Usually, we represent TRUE by 1 and FALSE by 0. The Boolean operations AND, OR, and NOT, represented by the symbols $\wedge$, $\vee$, $\neg$ and , respectively. We use the overbar as a shorthand for the $\neg$ symbol, so $\bar{x}$ means $\neg x$. A **Boolean formula** is an expression involving Boolean variables and operations. For example, $\phi = (\bar{x} \vee y) \wedge (x \vee \bar{z})$ is a Boolean formula. A boolean formula is **satisfiable** is some assignment of 0s and 1s to the variables makes the formula evaluate to 1. The preceding formula is satisfiable because the assignment $x = 0$, $y = 1$, and $z = 0$ makes ϕ evaluate to 1. We say the assignment *satisfies* ϕ. The **satisfiability problem** is to test whether a Boolean formula is satisfiable. Let $SAT = \{\langle\phi\rangle |\ \ \phi \text{ is a satisfiable Boolean formula }\}$.

Theorem 3 *(Theorem 7.27, page 272)*
Cook-Levin theorem *$SAT \in P$ iff $P = NP$*

Definition 4 *(Definition 7.28, page 272)*
*A function $f : \Sigma^* \to \Sigma^*$ is a **polynomial time computable function** if some polynomial time Turing machine M exists that halts with just $f(w)$ on its tape, when started on any input w.*

Definition 5 *(Definition 7.29, page 272)*
*Language A is **polynomial time mapping reducible**[1], or simply **polynomial time reducible**, to language B, written $A \leq_P B$, if a polynomial time computable function $f : \Sigma^* \to \Sigma^*$ exists, where for every w,*

$$w \in A \Longleftrightarrow f(w) \in B.$$

*The function f is called the **polynomial time reduction** of A to B.*

Theorem 6 *(Theorem 7.31, page 273)*
For any two languages A, B, if $A \leq_P B$ and $B \in P$, then $A \in P$.

[1]Is is called **polynomial time many-one reducibility** in some other textbooks.

Theorem 7 *(Implicit from the proof above) For any three languages A, B, C, if $A \leq_P B$ and $B \leq_P C$, then $A \leq_P C$.*

A **literal** is a Boolean variable or a negated Boolean variable, as in x and $\bar{x}$. A **clause** is several literals connected with $\vee$s. A Boolean formula is in **conjunctive normal form**, called a **cnf-formula**, if it comprises several clauses connected with $\wedge$s. It is a **3cnf-formula** if all the clauses have three literals, as in

$$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_2} \vee \overline{x_6}) \wedge (x_3 \vee \overline{x_7} \vee x_1) \wedge (x_4 \vee x_5 \vee \overline{x_6}).$$

Let $3SAT = \{\langle\phi\rangle|\ \phi$ is a satisfiable 3cnf-formula formula $\}$. In a satisfiable cnf-formula, each clause must contain at least one literal that is assigned 1.

Theorem 8 *(Theorem 7.32, page 274)*
$3SAT$ is polynomial time reducible $CLIQUE$.

Definition 9 *(Definition 7.34, page 276)*
A language B is ***NP-complete*** *if it satisfies two conditions:*

1. *B is in NP, and*
2. *every A in NP is polynomial time reducible to B.*

Theorem 10 *(Theorem 7.35, page 276)*
If B is NP-complete and $B \in P$, then $P = NP$.

Theorem 11 *(Theorem 7.36, page 276)*
If B is NP-complete and $B \leq_P C$ for C in NP, then C is NP-complete.

Theorem 12 *(Theorem 7.37, page 276)*
SAT is NP-complete.

This theorem restates Theorem 7.27, the Cook-Levin theorem, in another form.

Claim 13 *(Claim 7.41, page 280)*
If the top of the table is the start configuration and every window in the table is legal, each row of the table is a configuration that legally follows the preceding one.

Corollary 14 *(Corollary 7.42, page 282)*
$3SAT$ *is NP-complete.*

Corollary 15 *(Corollary 7.43, page 283)*
$CLIQUE$ *is NP-complete.*

If G is an undirected graph, a **vertex cover** of G is a subset of the nodes where every edge of G touches one of those nodes. The vertex cover problem asks whether a graph contains a vertex cover of a specified size: *VERTEX-COVER*= $\{\langle G\rangle, \langle k\rangle |$ G is an undirected graph that has a k-node vertex cover $\}$.

Theorem 16 INDEPENDENT-SET *is NP-complete.*

Theorem 17 *(Theorem 7.44, page 284)*
VERTEX-COVER *is NP-complete.*

Theorem 18 *(Theorem 7.46, page 286)*
$HAMPATH$ *is NP-complete.*

Next we consider an undirected version of the Hamiltonian path problem, called $UHAMPATH$.

Theorem 19 *(Theorem 7.55, page 291)*
$UHAMPATH$ *is NP-complete.*

Theorem 20 *(Theorem 7.56, page 292)*
$SUBSET - SUM$ *is NP-complete.*