Sections 7.4 – NP-Completeness

CSC 320

NP-Completeness

A language L is *NP-complete* if

- L is in NP
- L is NP-*hard*: there is a *polynomial time reduction* from any other language in NP to L.

The *Cook-Levin Theorem* states that SAT is NP-complete. This implies $SAT \in$ P iff P $=$ NP, i.e., SAT is at least as hard as any problem in NP and $SAT \in$ NP.

Poly-time Reductions

- We want a definition of reduction so that if (1) L is polynomial time reducible to L' and (2) L' is in P, then L is in P.

- A function $f : \Sigma^* \rightarrow \Sigma^*$ is a *polynomial time computable function* if some polynomial time deterministic TM exists which when any input w is input, the TM halts with just $f(w)$ on its tape.

- A language A is *polynomial time reducible* to language B, $A \leq_P B$ if there is a polynomial time computable function $f : \Sigma^* \rightarrow \Sigma^*$ such that:

$$w \in A \Leftrightarrow f(w) \in B$$

The function f is a *polynomial time reduction* from A to B.

The SAT Problem

We are now going to take a slight detour, and focus on a particular computational problem, known as *Boolean satisfiability* or *SAT* for short.

Why study SAT?

1. A powerful technique for solving problems – translate a problem instance as an instance of SAT, and then use a *SAT-solver*

2. A way to understand hardness of problems. We will show that the SAT problem is "as hard" as any problem in the complexity class *NP* (*nondeterministic polynomial time*). (Cook-Levin Theorem)

   Then, to show another problem Π is hard, we just need to show that we could use an efficient solver for Π to build and efficient solver for SAT.

Boolean Formulas

- A *Boolean variable* can take the values 1 (TRUE) or 0 (FALSE)
- A *Boolean formula* expression made up of Boolean variables using the *Boolean operations* AND, OR, and NOT, which are usually written $\wedge$, $\vee$ and $\neg$
- For a variable x, we often write $\overline{x}$ instead of $\neg x$
- A *truth assignment* T for a Boolean expression assigns each variable x the value 0 or 1. This is denoted $T(x)$.
- The value of an formula φ under a particular truth assignment T is the result of evaluating φ with each variable x replaced by its value $T(x)$, in the standard way
- E.g., Suppose $\varphi = (\overline{x} \wedge y) \vee (\overline{y} \wedge z)$, and T is defined by $T(x) = 0$, $T(y) = 1$ and $T(z) = 0$.
- Then under T ,

$$\begin{aligned} \varphi &\equiv (\overline{0} \wedge 1) \vee (\overline{1} \wedge 0) \\ &\equiv (1 \wedge 1) \vee (0 \wedge 0) \\ &\equiv 1 \vee 0 \equiv 1 \end{aligned}$$

Satisfiability of Boolean Formulas

- A formula φ is *satisfiable* if it evaluates to 1 under some truth assignment
- A *truth table* summarizes the values given to a formula by all possible truth assignments.
- So a formula φ is satisfiable if at least one row of the table gives it the value 1
- $\mathrm{SAT} = \{\langle\varphi\rangle \mid \varphi \text{ is a satisfiable Boolean expression}\}.$

Poly-time Reductions

Theorem: If $A \leq_P B$ and $B \in \mathsf{P}$ then $A \in \mathsf{P}$.

Proof: Suppose M is a polytime alg. for deciding B and f is a polytime reduction from A to B. Here is a polytime TM to decide A:

1. Compute $f(w)$

2. Run M on $f(w)$ and output whatever M outputs

Why won't the old definition of reduction work?

NP-completeness

Theorem: If B is NP-complete and $B \in \mathsf{P}$ then $\mathsf{P} = \mathsf{NP}$.

Proof: Suppose that B is NP-complete and $B \in \mathsf{P}$. Let L be any language in NP. Since B NP-complete, $L \leq_P B$. Since, $B \in \mathsf{P}$ we use the previous theorem to conclude that $L \in \mathsf{P}$.

So the assumptions imply $\mathsf{NP} \subseteq \mathsf{P}$. Since it is clear that $\mathsf{P} \subseteq \mathsf{NP}$, we conclude that $\mathsf{P} = \mathsf{NP}$.

Proving a Language L is NP-complete

Theorem: A language L is NP-complete if

1. L is in NP and

2. there is an NP-complete language B and $B \leq_P L$.

Proof: Let L_A be any language in NP. Since B is NP-complete, $L_A \leq_P B$. By assumption (2), $L_A \leq_P L$. Combining this with assumption (1), we can conclude that L is NP-complete.

NP-complete languages

1. SAT (Cook-Levin)
2. 3SAT (SAT $\leq_P$ 3SAT)
3. CLIQUE (3SAT $\leq_P$ CLIQUE)
4. HAMPATH (3SAT $\leq_P$ HAMPATH)
5. SUBSET-SUM (3SAT $\leq_P$ SUBSET-SUM)
6. VERTX-COVER (3SAT $\leq_P$ VERTX-COVER)

- A Boolean formula is in *conjunctive normal form (CNF)*, if it comprises several conjunctions ($\wedge$) of disjunctive ($\vee$) clauses.
- It is a *3CNF-formula* is all the disjunctive clauses have exactly 3 literals.
  - $(x_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (x_3 \vee \bar{x}_5 \vee x_6) \wedge (x_3 \vee \bar{x}_6 \vee x_4) \wedge (x_4 \vee x_5 \vee x_6)$