NP-completeness II

Let M be a TM, and let $t_M(x)$ be the number of steps required by M to halt (i.e., terminate in one of the two final states) on input x, where $x \in \Sigma^*$. Each step is an execution of one instruction of the TM. We define $t_M(x) = \infty$ if M never halts on input x. Let $|x|$ be the length of string X, let $n \in \mathbb{N}$

Definition 1 (Worst case time complexity of M)

$$T_M(n) = \max\{t_M(x) \mid |x| = n\}.$$

Exercise: Design a TM M such that $L(M) = \text{PAL}$, and $T_M(n) \in O(n^2)$.

Definition 2 (Polytime TM) A TM M is *polytime* if and only if $T_M(n) = O(n^k)$ for some $k \in \mathbb{N}$.

Now we are ready to define the class P formally.

Definition 3 (Formal definition of P)

$$P = \{L \mid L = L(M) \text{ for some polytime TM } M\ \}.$$

It is important to understand how this formal definition corresponds to the informal definition we mentioned before. On one hand, a decision problem D assigns "YES" or "NO" to each input string. On the other hand, L_D is the language (or "concrete problem", according to Cormen) associated with D. Code each input w of the decision problem D by a string $e(w) \in \Sigma^*$. Define

$$L_D = \{x \mid x \text{ codes a "YES" input to } D\} = \{e(w) \mid w \text{ is a "YES" instance of } D\}$$

Thus we formalize the notion "decision problem D" by a language L_D, and we formalize the notion "algorithm" by Turing machine.

Polytime Thesis: A decision problem D can be solved by a polytime algorithm **iff** L_D can be recognized by some polytime TM, i.e. $L_D = L(M)$ where M is some polytime TM.

Note that one direction of the Polytime Thesis is obviously true: If L_D can be recognized by some polytime TM, then the decision problem D can be solved by a polytime algorithm. The algorithm is the TM.

Note that the definition of the class P is sensitive to how the input is represented (in binary or unary notation). For example, if the input numbers to SKD are expressed in unary notation, then the dynamic programming problem discussed earlier in the course solves SKD in polytime. However, if the numbers are expressed in binary notation (which is the default case), then SKD is NP-complete.

Example: $\mathrm{SQ} = \{\bar{k^2} \mid k \in \mathbb{N}\}$, where $\bar{m}$ denotes the binary expression for $m \in \mathbb{N}$.

Claim: $\mathrm{SQ} \in P$.

To show that this language is in P, we have to come up with a polytime algorithm for the corresponding decision problem. Take $\bar{m}$ as an input to this algorithm, and then apply binary search to find an $\bar{n}$ such that $n^2 = m$. Binary search takes a polynomial time in the length of the input string. We could also use Newton's method that works even faster.

Example: MSTD (Minimum Spanning Tree Decision Problem).

Input: A connected graph $G = (V, E)$, a cost function $c : E \longrightarrow \mathbb{N}$, and $B \in \mathbb{N}$.

Question: Does G have a spanning tree T s.t. $c(T) \leq B$?

Claim: $\mathrm{MSTD} \in P$.

To construct a polytime algorithm, use Kruskal to find a min cost ST $T_{\min}$. If $c(T_{\min}) \leq B$, output "YES", else, output "NO".

If you can solve a search problem in polytime, you can solve the corresponding decision problem in polytime.

An interesting question is whether SKD is in P.

Conjecture: $\mathrm{SKD} \notin P$

Later we shall see that SKD is NP-complete, and hence it is very unlikely that SKD is in P.

We introduce a new class of languages NP (for Nondeterministic Polytime).

Definition 4 (Informal Definition of NP) NP is a class of languages such that $L \in NP$ iff L has a polytime "guess-and-check" algorithm.

Example: SKD

Input: $\langle d, w_1, \ldots, w_d, C, B \rangle$

Question: Is there $S \subseteq \{1, \ldots, d\}$ s.t. $B \leq K(S) \leq C$, where $K(S) = \sum_{i \in S} w_i$.

To show $\mathrm{SKD} \in NP$, we have to come up with a polytime "guess-and-check" algorithm for the decision problem above.

This algorithm would be as follows:

1. Guess $S \subseteq \{1, \ldots, d\}$.
2. Check $B \leq K(S) \leq C$.

The checking must be done in polytime. The set S is called a *certificate* for this input instance.

Definition 5 (Formal definition of *NP*) *NP* is a class of languages such that $L \in NP$ if and only if there is a relation $R(x, y)$, where $x, y \in \Sigma^*$ which is recognized by some two-input TM M which runs in time polynomial in $|x|$ s.t. $L = \{x \mid R(x, y) \text{holds for some } y\}$. If relation $R(x, y)$ holds, string y is called a *certificate* for x.

A guess-and-check algorithm works as follows:

1. Guess a certificate y.
2. Check whether $R(x, y)$ holds.

Formally,

$$x \in L \text{ iff } \exists y R(x, y)$$

This is a short way of writing the definition above. Note that if x has a certificate y, then x has a certificate y that is not too long; i.e. $|y| \leq |x|^k$ for some k. This is because the TM checking $R(x, y)$ must complete its computation within this many steps.

Relation between classes P and *NP*:

Claim: $P \subseteq NP$.

Proof of Claim. We need to show that for any language L, if $L \in P$, then $L \in NP$. Define

$$R(x, y) \equiv (x \in L)$$

Note that $R(x, y)$ is polytime simply because we assume that L is in P, so the question: "is $x \in L$?" can be answered in polytime. Notice that if $x \in L$, then according to our definition of R, *any* string y will do as a certificate for x. On the other hand, if $x \notin L$, then *no* string y will serve as a certificate for x. Note that the polytime TM which determines whether $R(x, y)$ holds, ignores its input y.

Returning to our SKD example, string $x = e(\langle d, w_1, \ldots, w_d, C, B \rangle)$ encodes the input, string y encodes a subset $S \subseteq \{1, \ldots, d\}$, and $R(x, y)$ holds if and only if $B \leq K(S) \leq C$.

We have shown $P \subseteq NP$, however

Conjecture: $P \neq NP$.

We have shown SKD $\in NP$. Thus the above conjecture follows from our earlier conjecture that SKD $\notin P$.

Turing machines as functions

TMs compute functions from input to output: $f : \Sigma_1^* \longrightarrow \Sigma_2^*$.

Input: $\dots bbbx_1 \dots x_n bbbb \dots$, where $x = x_1, \dots, x_n$ is an input string.

Output: $\dots bbby_1 \dots y_m bb \dots$ where $f(x) = y = y_1, \dots, y_m$ is an output string.

Suppose $L_1 \subseteq \Sigma_1^*$, $L_2 \subseteq \Sigma_2^*$,

Definition 6 We say L_1 is polytime-many-one-reducible to L_2 ($L_1 \leq_p L_2$) iff there is a function $f : \Sigma_1^* \longrightarrow \Sigma_2^*$ s.t. f is computable by a polytime TM and $x \in L_1$ iff $f(x) \in L_2$ for all $x \in \Sigma_1^*$

Example: PARTITION $\leq_p$ SKD, where PARTITION is defined as follows:

Input: $\langle d, w_1, \dots, w_d \rangle$ all in $\mathbb{N}$.

Question: Is there $S \subseteq \{1, \dots, d\}$ s.t. $\sum_{i \in S} w_i = \sum_{i \in \bar{S}} w_i$, where $\bar{S} = \{1, \dots, d\} - S$, the complement of S.

Claim: PARTITION $\leq_p$ SKD.

Proof of Claim. Given an input $I = \langle d, w_1, \dots, w_d \rangle$ to PARTITION, compute

$$I' = f(\langle d, w_1, \dots, w_d \rangle) = \langle d', w'_1, \dots, w'_{d'}, C, B \rangle$$

such that $I \in$ PARTITION iff $I' \in$ SKD.

What is I'? Take $d' = d$, $w'_i = w_i$, $1 \leq i \leq d$. Take $C = B = \frac{1}{2} \sum_{i=1}^{d} w_i$. If $I' \in$ SKD, then there is an $S \subseteq \{1, \dots d\}$ such that $\sum_{i \in S} w_i = \frac{1}{2} \sum_{i=1}^{d} w_i$. Hence, $\sum_{i \in \bar{S}} w_i = \sum_{i \in S} w_i$. So $I \in$ PARTITION.

Exercise: Conversely, if $I \in$ PARTITION then $I' \in$ SKD.

Thus, if we could solve SKD easily, we could solve PARTITION easily.

A decider for L is a TM that halts in state q_{accept} on input x iff $x \in L$. The decider for L_2 does not have to be polytime.

Suppose $L_1 \leq_p L_2$, i.e., $x \in L_1$ iff $f(x) \in L_2$, for all $x \in \Sigma_1^*$. How does the polytime algorithm from the "black box" work?

1. Input x.
2. Compute $f(x)$.
3. Ask $f(x) \in L_2$?.
4. Output this answer.

Definition 7 (*NP*-hard) A language L_0 is *NP*-hard if $L \leq_p L_0$ for every language $L \in NP$.

Note that there are *NP*-hard languages that are not in *NP*.

Definition 8 (*NP*-complete) A language L_0 is *NP*-complete if

1. L_0 is *NP*-hard,
2. $L_0 \in NP$.

An example of an *NP*-complete language is SKD.

Theorem 1 If L_0 is any language such that L_0 is *NP*-hard and $L_0 \in P$, then $P = NP$

The theorem states that if we ever find a polytime algorithm for an *NP*-complete problem, then every *NP* problem has a polytime algorithm.

Exercise: Prove this theorem as follows:

First prove the following lemma:

Lemma 1 If $L_1 \leq_p L_2$ and $L_2 \in P$, then $L_1 \in P$.

We already know that $P \subseteq NP$, so we only need to show the other direction, i.e. that $NP \subseteq P$. For this, use the fact that L_0 is *NP*-hard and $L_0 \in P$, and the lemma.