NP-completeness II

The class P

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 $T_M(n) \in O(n^2)$. Recall that $L(M)$ is the language consisting of those strings of Σ^* for which M halts in the state q_{accept}.

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

Now we are ready to define the class P formally.

Definition 3 (Formal definition of P)

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

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

Claim: 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: 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: SKD $\notin$ P

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

The class NP

We introduce a new class of languages NP (NP stands for Non-deterministic Polytime, since NP can be defined in terms of non-deterministic Turing machines; we use a different characterization of NP).

An **informal** definition of NP: a decision problem is in the class NP if, given an input instance and an alleged solution, it can be verified in polytime whether the solution is a correct one.

For example, SKD (Simple Knapsack Decision) is in NP because, given an input instance $\langle w_1, \ldots, w_d, B, C\rangle$ to SKD and a set $S \subseteq \{1, \ldots, d\}$, we can verify efficiently (i.e., in polytime) whether $B \leq K(S) \leq C$.

A **working** definition of NP (i.e., a definition to use when showing that a given decision problem is in NP): a decision problem is in NP if it has an efficient "guess-and-check" algorithm. The guess-and-check algorithm works as follows: given an input instance I to our decision problem, we *guess* a solution S, and we *check* (in polytime) that S is indeed a correct solution.

For example, MSTD (Minimum Cost Spanning Tree Decision) is in NP because it has the following guess-and-check algorithm: given an input instance $G = (V, E)$ with cost function $c : E \longrightarrow N$ and parameter B, we guess a set $T \subseteq E$ and we check that $c(T) \leq B$.

The guess-and-check procedure is not as vague as it seems; we are now going to give an example of a guess-and-check procedure that would not work. Consider one more time the SKD problem, and suppose that we aim to prove that SKD is in NP by employing the following guess-and-check algorithm: we guess $S_1, S_2, \ldots, S_{2^d}$, which contains all the subsets of $\{1, \ldots, d\}$, and we check that for some $i \in \{1, \ldots, 2^d\}$, $B \leq K(S_i) \leq C$. This particular guess-and-check algorithm does not do it because it is *not efficient*; the checking part is exponential in the size of the input!

However, if we just guess one subset $S \subseteq \{1, \ldots, d\}$, and check that $B \leq K(S) \leq C$, we have an efficient guess-and-check algorithm that shows that SKD is indeed in NP.

We finally give a **formal** definition of NP: we first define what it means for a relation $R(x, y)$ to be polytime in the length of x. It means that there exists a Turing machine M_R and a polynomial $p(n)$ with the following properties: when M_R is presented with the input:

$$\underbrace{x_1 x_2 \ldots x_n}_{x} \,\#\, \underbrace{y_1 y_2 \ldots y_m}_{y}$$

M_R takes at most $p(n)$ steps to halt, and it halts in state q_{accept} iff $R(x, y)$ holds.

The only subtle thing about this definition is that M_R runs in time polynomial in x, while the input consists of two arguments (presented to M_R as a single string, consisting of a segment of 0s and 1s—binary encoding of x, separated by # from a second segment of 0s and 1s—binary encoding of y.) This means that if m is very large, M_R will ignore part of y; in fact, M_R is going to read at most $p(n)$ bits of y (why?). Of course, there is nothing to consider if y is "short".

So now that we have a precise definition of what it means for $R(x, y)$ to be polytime in the length of x (i.e., polytime in $|x|$), we can define the class NP precisely. We say that a language L (since we are being rigorous, we talk in terms of languages, rather than decision problems) is in NP iff there exists a relation $R(x, y)$, polytime in $|x|$, such that $x \in L \iff \exists y R(x, y)$.

Let $p(n)$ be the polynomial that bounds the time of the computation of M_R (as defined above). By the above discussion, we know that we can replace $\exists y R(x, y)$ in the definition by:

$$\exists y (|y| \leq p(|x|) \text{ and } R(x, y))$$

Make sure that you understand this subtle point.

So how does this formal definition correspond to our guess-and-check method of determining if a language is in NP? Well, what we guess is the y and R is the checking relation; let's see this with an example. We are going to show that SKD is in NP by using the formal definition of NP. Define R as follows:

$$R(\underbrace{e(\langle w_1, \ldots, w_d, B, C\rangle)}_{x}, \underbrace{e(S)}_{y}) \text{ holds iff } B \leq K(S) \leq C$$

Verifying that $B \leq K(S) \leq C$ can be certainly done in polytime, while given the above definition of R, the statement $\exists y R(x, y)$ holds iff there exists a set $S \subseteq \{1, \ldots, d\}$ such that $B \leq K(S) \leq C$. Note that $|y| \leq O(d)$.

Lemma 1 P $\subseteq$ NP.

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

$$R(x, y) \text{ to be } (x \in L)$$

That is, $R(x, y)$ holds iff $x \in L$ (note that y does not appear on the right-hand side of the definition).

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.

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. Again, note that the polytime TM which determines whether $R(x, y)$ holds, ignores its input y. $\square$

We have shown that 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.

Many-one reducibility

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

Input: $\cdots \mid b \mid b \mid x_1 \mid \ldots \mid x_n \mid b \mid b \mid \cdots$ where $x = x_1, \ldots, x_n$ is the input.

Output: $\cdots \mid b \mid b \mid y_1 \mid \ldots \mid y_m \mid b \mid b \mid \cdots$ where $f(x) = y = y_1, \ldots, y_m$ is the output.

Definition 4 Suppose that $L_1 \subseteq \Sigma_1^*$, $L_2 \subseteq \Sigma_2^*$. We say L_1 is polytime-many-one-reducible to L_2 (written as $L_1 \leq_p L_2$) iff there is a function $f : \Sigma_1^* \longrightarrow \Sigma_2^*$ such that f is computable by a polytime TM and $x \in L_1$ iff $f(x) \in L_2$ for all $x \in \Sigma_1^*$

We also say L_1 is many-one polytime reduces to L_2, or L_1 p-transforms to L_2. This is a special case of polytime reductions $L_1 \xrightarrow{p} L_2$ (the "black box" approach that we have seen already).

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

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

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

Claim: PARTITION $\leq_p$ SKD.

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

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

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 $S \subseteq \{1, \ldots d\}$ 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.

NP-completeness

Definition 5 (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 6 (NP-complete) A language L_0 is NP-complete if:

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

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 2 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.