

Notes taken on Stephen Cook's lecture by Eugenia Ternovskaia  
on February 22, 1999.

## NP-completeness: Classes P and NP

Recall our informal definition of class **P**: this is a class of decision problems that can be solved in polytime.

The reason for introducing Turing machines (TMs) is to make the notion of **P** formal.

Let  $M$  be a TM. The language accepted by  $M$  is  $L(M) = \{x \in \Sigma^* \mid M \text{ accepts } x\}$ , where  $\Sigma$  is the input alphabet for  $M$ ,  $x$  is a string over this alphabet, i.e.,  $x = x_1, x_2, \dots, x_n$ ,  $x_i \in \Sigma$ . We have  $\dots bbx_1x_2\dots x_nbbb\dots$  on the input of  $M$ , and  $q_0$  is the start state of this machine, by convention.

Last time, we have considered the language  $PAL = \{yy^R \mid y \in \{0,1\}^*\}$  of palindromes over  $\{0,1\}^*$  and designed a TM  $M_{PAL}$  s.t.  $L(M_{PAL}) = PAL$ .

Let us define a notion of worst case time complexity of TMs, and then return to the palindrome example.

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  $M$ ).**

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

Let us return to our palindrome example. Since the TM accepting palindromes over  $\{0,1\}^*$  scans  $x$  symbol by symbol and reduces the number of scanned symbols by one on every iteration, the worst case time complexity of  $M_{PAL}$  is

$$T_{M_{PAL}}(n) = n + (n - 1) + (n - 2) + \dots + 1 = \Omega(n^2).$$

Notice that on some strings  $M_{PAL}$  performs much faster than on others. For example, if  $x = 10000000$ , then  $t_M(x) = 9$ . In general, if the first and the last symbols of  $x$  differ, then  $t_M(x) = n + 1$ .

Now we are ready to define the class **P** formally.

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

**Definition 3 (Formal definition of P).**

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

It is important to understand how this formal definition corresponds to the informal definition we mentioned at the beginning of this lecture. On one hand, a decision problem  $D$  assigns “yes” or “no” to each input string. On the other,  $L_D$  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

$$\begin{aligned} L_D &= \{x \mid x \text{ codes a “yes” input to } D\} \\ &= \{e(w) \mid w \text{ is a “yes” input to } D\}. \end{aligned}$$

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

**Polytime Thesis** *If a decision problem  $D$  can be solved by a polytime algorithm, then  $L_D$  can be recognized by some polytime TM.*

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

## Examples of languages in P

**Example 1: SQ** Let  $\bar{m}$  is the binary notation for  $m$ .

$$SQ = \{\bar{k}^2 \mid k \in \mathbb{N}\}.$$

**Claim**  $SQ \in \mathbf{P}$ .

To show that this language is in  $\mathbf{P}$ , 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. It takes a polynomial time in the length of the input string. We could also use Newton’s method that works even faster.

**Example 2: MSTD** (Minimum Spanning Tree Decision Problem).

Input: A connected graph  $G = (V, E)$ ,  $c(e) \in \mathbb{N}$ ,  $e \in E$ ,  $B \in \mathbb{N}$ .

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

**Claim**  $MSTD \in \mathbf{P}$ .

To construct a polytime algorithm, use Kruskal to find a MST  $T$ . If  $c(T) \leq B$ , 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  $SKSD$  is in  $\mathbf{P}$ .

**Conjecture:**  $SKSD \notin \mathbf{P}$ .

Later we shall see that  $SKSD$  is NP-complete.

We introduce a new class of languages  $\mathbf{NP}$  (for Nondeterministic Polytime).

**Definition 4 (Informal Definition of NP).**  $\mathbf{NP}$  is a class of languages such that  $L \in \mathbf{NP}$  if and only if  $L$  has a polytime “guess-and-check” algorithm.

**Example 3: SKSD**

Input:  $d, w_1, \dots, w_d, C, B$ .

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

To show  $SKSD \in \mathbf{NP}$ , 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  $S \subseteq \{1, \dots, d\}$ .
- 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).**  $\mathbf{NP}$  is a class of languages such that  $L \in \mathbf{NP}$  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 $\mathbf{P}$ and $\mathbf{NP}$

**Claim 1.**

$$\mathbf{P} \subseteq \mathbf{NP}.$$

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

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

Note that  $R(x, y)$  is polytime simply because we assume that  $L$  is in  $\mathbf{P}$ , so the question  $x \in L?$  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 SKSD example, string  $x = e(d, w_1, \dots, w_d, C, B)$  encodes the input, string  $y$  encodes a subset  $S \subseteq \{1, \dots, d\}$ ,  $R(x, y)$  holds if and only if  $B \leq K(S) \leq C$ .

We have shown  $\mathbf{P} \subseteq \mathbf{NP}$ .

**Conjecture:**  $\mathbf{P} \neq \mathbf{NP}$ .

We have shown  $SKSD \in \mathbf{NP}$ . Thus the above conjecture follows from our earlier conjecture that  $SKSD \notin \mathbf{P}$ .