Recitation 06: P and NP

This recitation covers some basic definitions and tools of complexity theory.

In the first half of the semester, we learned about computability theory, where we placed languages in classes (regular, context-free, decidable, T-recognizable) based on whether they could be solved using certain models of computation. We now move on to complexity theory, where we restrict our attention to *decidable* languages and first focus on determining how much *time* is required to decide them.

Time complexity for deterministic models

To determine the time complexity of a language, we must first define what it means for a Turing machine to run in $t(n)$ time, where $t : \mathbb{N} \to \mathbb{N}$.

Definition 1. A single-tape deterministic TM M runs in time $t(n)$ if M halts in at most $t(n)$ steps on all inputs of length n.

Definition 2. $TIME(t(n)) = \{B \mid$ some single-tape deterministic TM M runs in $O(t(n))$ time and $L(M) = B\}$.

A TM runs in $O(t(n))$ time if it runs in $\leq ct(n)$ time for some constant $c > 0$ independent of n.

We can now define our first time complexity class.

Definition 3. $P = \bigcup_{k \in \mathbb{N}} TIME(n^k)$.

In words, P is the set of languages that can be decided in cn^k time for constants c, k. These are the languages that can be decided in *polynomial time*. To show that a language is in P, we typically must construct a TM that decides that language, then argue that the TM halts in polynomial time.

Model independence

While Definitions 1 and 2 require a single-tape deterministic TM, Definition 3 is *model independent*. This means that for all reasonable deterministic models of computation, P defines the same class of languages. This is useful because we are not restricted to any specific deterministic model when proving languages are in P.

Time complexity for nondeterministic models

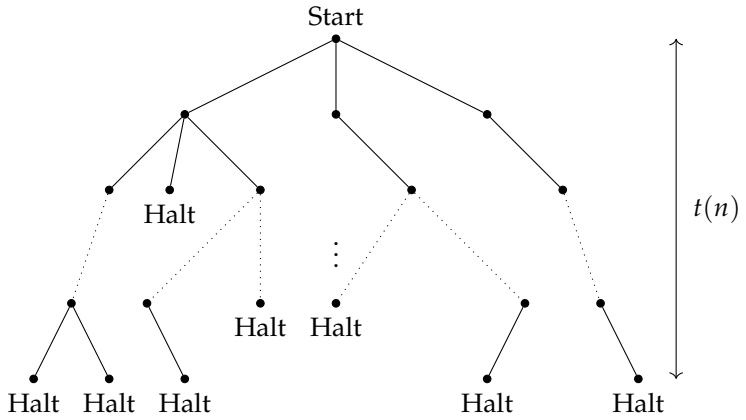
We define the analogous terms for nondeterministic TMs.

Definition 4. A nondeterministic TM N runs in time $t(n)$ if all of the threads of N halt in at most $t(n)$ steps on all inputs of length n.

Note the difference between acceptance and runtime: for an NTM to accept an input, it is enough for one thread to accept. For an NTM to run in time $t(n)$, *all* threads need to halt in that time.

Definition 5. $NTIME(t(n)) = \{B \mid \text{some NTM } N \text{ runs in } O(t(n)) \text{ time and } L(N) = B\}$.

Intuitively, this means that the tree consisting of all the branches of N's computation can have *height* at most $t(n)$, as shown below. Note however that these definitions do not limit the *width* of the tree, and there could be a non-polynomial number of branches, as long as each branch has polynomial length.



Definition 6. $NP = \bigcup_{k \in \mathbb{N}} NTIME(n^k)$.

In words, NP is the set of languages decided by some NTM in polynomial time. This definition is again model independent for all reasonable nondeterministic models of computation.

Certificates for NP

Intuitively, NP consists of languages L for which we can *verify membership quickly*. For an input $x \in L$, there is some short "certificate" c such that if given c, it is easy to confirm that x is in L. Here are some examples of certificates for instances of languages in NP:

Here, "short," "quickly," and "easy" all mean polynomial in $|x|$.

- If a graph $G \in HAMPATH$, the certificate would be the sequence of nodes corresponding to the Hamiltonian path in G. It is certainly polynomial time to check that each node in the sequence is in G, all nodes in G appear in the sequence exactly once, and all pairs of adjacent nodes in the sequence are connected by an edge in G.

- For the *SUBSET-SUM* language, defined as $\{\langle S, t\rangle \mid S = \{x_1, ..., x_k\}$ and $\exists\{y_1, ..., y_l\} \subseteq S$ such that $\sum_{i=1}^{l} y_i = t\}$, the certificate would be $c = \{y_1, ..., y_l\}$. Given c, it is easy to check that $c \subseteq S$ and that the sum of the elements of c is t.

This concept can be formalized with the following theorem.

Theorem 7. *$L \in NP \iff$ there exists verifier TM V such that $L(V) \in P$ and ($x \in L \iff$ there exists c such that $|c| = O(poly(|x|))$ and $\langle x, c\rangle \in V$).*

Proof. Informally, we want to show that $L \in NP \iff$ strings in L have short and quickly checkable certificates, and strings not in L don't.

($\Longrightarrow$) Let $L \in NP$. Then, there exists NTM N that decides L in polynomial time. We will show that strings in L have short and quickly checkable certificates. Construct TM V which takes in $\langle x, c\rangle$ and accepts iff c is an accepting computation history of N on x. From our previous algorithms for checking that computation histories are valid and accepting, we know that V runs in polynomial time. Then, for all $x \in L$, let c be the computation history of N on x. We know that c is short since N runs in polynomial time.

Intuitively, think of the computation history as all the nondeterministic choices that N made on an input. These choices specify a thread, and V just needs to check whether this thread leads to an accept state.

($\Longleftarrow$) Let language L have short and quickly checkable certificates for strings in L. We will show that $L \in NP$. Let V be the verifier for L that runs in time n^k for inputs of length n, for some constant k. For an input x of length n, since V accepts $\langle x, c\rangle$ for some certificate c in time n^k, we know that $|c|$ won't exceed n^k. We can thus build NTM N: on input x, nondeterministically guess certificate c of length at most n^k. Run V on $\langle x, c\rangle$ and accept if and only if V accepts. N will accept x if and only if there exists a c such that V accepts $\langle x, c\rangle$, so $L(N) = L$. All threads will halt in polynomial time since $|c|$ is polynomial, and V runs in polynomial time. □

A small detail on guessing: since a TM a fixed number of states, we can't guess the entire c all at once (since the number of possibilities may depend on n). Instead we can guess c bit-by-bit.

Given Theorem 7, it is now easy to show that a language $L \in NP$. To construct an NTM running in polynomial time that decides L, we can

1. Think of a (short, easy to check) certificate for strings in L.
2. Build the following NTM. On input x:
   (a) Guess the certificate c.
   (b) Check whether or not c is a valid certificate for x. Accept if so. Else, reject.

Some examples

$HAMPATH \in NP$

Recall that $HAMPATH = \{\langle G, s, t\rangle : \text{there is a Hamiltonian path from } s \text{ to } t\}$. Here, G is a directed graph, and s and t are two vertices in G. A Hamiltonian path from s to t is a sequence of vertices $s = v_1, v_2, \ldots, v_n = t$ where every pair of vertices (v_i, v_{i+1}) are connected by a (directed) edge. Let us show that $HAMPATH \in NP$.

Intuitively, the idea is to exhibit an easy-to-verify, polynomial-sized certificate for the existence of a Hamiltonian path between s and t. What could such a certificate be? Well, if you think about it, the Hamiltonian path itself is such a certificate! If you already know the sequence of vertices $s = v_1, v_2, \ldots, v_n = t$, then it's easy to check that this sequence is a Hamiltonian path.

Let us formalize this intuition into an non-deterministic polynomial time Turing machine. The NTM is going to guess (nondeterministically) the sequence $v_1, \ldots, v_n$ and then check whether it is a valid Hamiltonian path, that $s = v_1$ and $v_n = t$, and that every vertex is used exactly once.

Let us check that both guessing the sequence and checking that it is a valid Hamiltonian path take polynomial time. The sequence $v_1, \ldots, v_n$ is polynomial in size: particularly, each vertex has a $O(\log n)$ bit representation, so the total number of bits to represent $v_1, \ldots, v_n$ is $O(n \times \log n)$. Then, nondeterministically guessing these $O(n \times \log n)$ bits takes only $O(n \times \log n)$ time. Secondly, once we have $v_1, \ldots, v_n$, we need to check that each $v_i \to v_{i+1}$ is an edge in the graph. To do this, we can scan through the description of G and see if $(v_i, v_{i+1}) \in E(G)$. This will take polynomial time per check, for a total of polynomial time to verify that each pair of consecutive vertices is connected by an edge. Next, checking $s = v_1$ and $t = v_n$ is easy. Finally, we need to check that every vertex is used exactly once, which can be done by checking off the vertices from a list as they appear in the path.

$COMPOSITES \in NP$

Let us define $COMPOSITES = \{m \in \mathbb{N} : n \text{ is composite}\}$. Recall that a composite number is one that is not prime, i.e. $m = pq$ where $p, q \in \mathbb{N}$ and $p, q \neq 1$.

Let us show that $COMPOSITES \in NP$. Again, the idea is to identify a certificate for $m \in COMPOSITES$. Here, we can take the certificate to be the factorization $m = pq$. Our NTM will non-deterministically guess the factorization (p, q), and then check that $m = pq$ and $p, q \in \mathbb{N}\backslash\{1\}$.

Let us check that both the guessing of (p, q) and checking that it

is a proper factorization take polynomial time. First, what is the size of (p, q) in terms of the size of m? Note that to represent m, we need to use $O(\log m)$ bits to represent n in binary. So, the size of m (i.e., the input size) is $O(\log m)$. Representing p and q as bit strings also takees $O(\log m)$ bits, since $p, q < m$. Thus, we can guess (p, q) in time polynomial in the input size. Next, checking $m = pq$ can be done by multiplying pq via long multiplication and checking equality, both of which can be done in polynomial time. Also, checking $p, q \neq 1$ can easily be done.