The class NP

Review

NTIME(t(n)) = { L | L is decided by a NTM whose maximum running time of a nondeterministic branch of computation is O(t(n)) }

We will now define the special class of problems solved by a nondeterministic TM in polynomial time.

Definition. The class NP.

$$\text{NP} = \bigcup_{k=1}^{\infty} NTIME(n^k)$$

Immediately observe P $\subseteq$ NP, because for every k, TIME(n^k) $\subseteq$ NTIME(n^k).

Example. HAMPATH. A Hamiltonian path in a directed graph is a directed path that goes through each node exactly once. The problem of finding such a path is defined as follows:

HAMPATH = { (G, s, t) | G is a directed graph with a Hamiltonian path from s to t }

Theorem. HAMPATH $\in$ NP.

Proof. "On input (G, s, t),

- Nondeterministically write down all nodes of G except s and t in some order:
  $v_1\ v_2\ \ldots v_{n-2}$. (all possible permutations)
- Check that there are edges (s, v_1), (v_1, v_2), …, (v_{n-2}, t). If yes, accept. Otherwise reject."

How fast does the above algorithm run (in nondeterministic time)?

Step 1 is easy to do in $O(n^2)$ steps where n is the number of nodes. Say that nodes s and t start as marked and all other nodes as unmarked. The TM can go through all the nodes, nondeterministically picking one of the unmarked ones, writing it down, and marking it. Repeating this n-2 times will provide $v_1\ v_2\ \ldots v_{n-2}$.

Step 2 is easy to do in O(nl) steps where n = number of nodes, and l = number of edges.

One very useful expression of the class NP, is as the class of all problems whose solutions are *verifiable* in deterministic polynomial time. For instance, given a potential solution to HAMPATH, i.e. given an order of the nodes s, v_1, …, v_{n-2}, t, it is possible to

check with a deterministic TM in polynomial time whether this is a Hamiltonian path, or not.

It is important to note that the problems in NP are not necessarily solvable in deterministic polynomial time!

A *verifier* for a language A is an algorithm V, where

A = {w | V accepts <w,c> for some string c}

Time of verifier is measured in terms of the length of w – therefore a polynomial time verifier runs in polynomial time in the length of w. A language is **polynomial verifiable** if it has a polynomial time verifier.

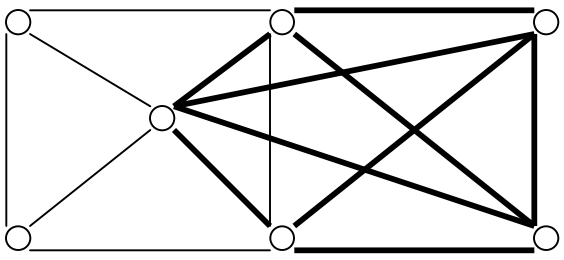
A certificate is information that verifies that a string is a member of A. For example: a certificate for the HAMPATH problem <G,s,t> is a path from s to t.

NP is the class of languages that have polynomial time verifiers

A language is in NP iif it is decided by some nondeterministic polynomial time Turing machine.

Example. CLIQUE. A **clique** in an undirected graph is a subgraph wherein every two nodes are connected by an edge. A k-clique is a clique with k nodes.

A 5-clique:



CLIQUE = { (G, k) | G is an undirected graph and it has some k-clique }

Theorem. CLIQUE ∈ NP.

Proof. "On input (G, k),

1. Nondeterministically select a subset C of k nodes of G.
2. Test whether every two nodes in C are connected by an edge. If yes, accept. Otherwise reject."

The clique C is the certificate.

Example. COMPOSITES.

$$\text{COMPOSITES} = \{ w \mid w \text{ is an integer, and } w = pq \text{ for some } p,q > 1 \}$$

Is COMPOSITES $\in$ P? We don't currently know how to efficiently factor a number; otherwise certain cryptographic protocols in use now would be easy to break.

Theorem. COMPOSITES $\in$ NP. – easily verified

Proof. "On input w,
1. Nondeterministically write down a number $x < w$.
2. Check that x divides w. If yes, accept. Otherwise reject."

Is COMPOSITES $\in$ P? Well how about the following algorithm:

"On input w,
For x = 1 to w check if x divides w. If some x does, accept. Otherwise reject."

The above algorithm runs in time polynomial on w. But it runs in time exponential on |w|, which is the number of digits in the binary representation of w! So the algorithm is not polynomial on the size of the input, where size is taken to be |w| and *not* w.

P Versus NP

Informally,

- P = Languages for which one can **decide** membership in polynomial time.
- NP = Languages for which one can **verify** membership in polynomial time.

The most important open problem of theoretical computer science is perhaps whether these two classes are equal.

$$P \stackrel{?}{=} NP$$

We don't know whether we can find polynomial time algorithms for problems in NP. However most people believe that the two classes are different, because many people have invested enormous effort to find polynomial time algorithms for NP problems.

So far the best methods known for solving problems in NP involves exponential searches.

For example:

- COMPOSITES can be solved by checking ∀x<w whether x divides w.

- CLIQUE can be solved by
  (i) Listing all sets of k nodes of the graph, and
  (ii) Verifying whether one of them forms a clique.

It is clear that:

$$NP \subseteq EXPTIME = \bigcup_k TIME(2^{n^k})$$

But we don't know if NP is some smaller complexity class than EXPTIME. And we don't know if NP is some larger complexity class than P. Of course we know that P is a much smaller complexity class than EXPTIME, but we just have no idea where to place NP between P and EXPTIME!