The Class $\mathcal{NP}$

$\mathcal{NP}$ is the problems that can be solved in polynomial time by a nondeterministic machine.

$\mathcal{NP}$

The ***time*** taken by nondeterministic TM is the length of the *longest* branch.

$$\boxed{\text{\textit{The collection of all problems that can be solved in polynomial time by a nondeterministic machine is called } \mathcal{NP}.}}$$

That is, language $L \in \mathcal{NP}$ if there is k and an NTM that decides L that runs in time $O(n^k)$.

$\mathcal{P}$ *versus* $\mathcal{NP}$

It is immediate that

$$\mathcal{P} \subseteq \mathcal{NP}$$

But does nondeterminism buy one anything?

Example: HAMPATH

$\texttt{HAMPATH} = \{\, \langle G, a, b \rangle : G$ is graph with hamiltonian path from a to $b \,\}$

We do not know how to decide HAMPATH in polynomial time. (Trying all possible paths fails, as there are exponentially many.) But there is a fast nondeterministic program. Guess path node by node, at each stage choosing an unvisited node. Time taken is at most quadratic in the number of nodes. And so HAMPATH is in $\mathcal{NP}$.

One Limit of Nondeterminism

Theorem. *Let L be recursive language. If there is nondeterministic TM for L that runs in time $T(n)$, then there is deterministic TM for L that runs in time $O(C^{T(n)})$ for some constant C.*

Proof. Say L is accepted by NTM N. In the time available, N can access at most $T(n)$ cells. So, the number of configurations of N is at most $X = qng^{T(n)}$ where g is the alphabet size and q the number of states. . .

Proof Continued

Thus, generate all possible configurations. Determine which configurations follow which. Then see if there is a path from the starting configuration to an accepting configuration. The result runs in time polynomial in X, which gives the result.

Certificates

One can reprogram an NTM to start by nondeterministically writing an arbitrary string on a special tape called the ***certificate-tape***. After that, it runs deterministically: when it has to make a nondeterministic choice, it looks up its next move on the certificate-tape.

Example: `HAMPATH`

The certificate for `HAMPATH` is the hamiltonian path. All the program has to do is to check that the edges form a path (end of one is start of next), that the path starts at a and ends at b, and that each node is visited exactly once.

Prover and Verifier

So we have ***Prover/Verifier*** situation. The problem can be decided by a Verifier in polynomial time if given a hint from an omniscient Prover.

Note that the correct answer is made. If there is hamiltonian path, then correct hint will be verified and Verifier will say yes; if there is no hamiltonian path, then Verifier cannot be conned into saying yes.

Example: Satisfiability

In boolean formulas, a ***clause*** is the *or* of literals. A formula is in ***conjunctive normal form*** if the *and* of clauses.

A ***satisfying assignment*** is one that makes the formula TRUE. For example,

$$x \wedge (x \vee y \vee \bar{z}) \wedge (\bar{x} \vee \bar{y})$$

is in conjunctive normal form and is satisfiable: set x to TRUE and y, z to FALSE.

Example: SAT

$\texttt{SAT} = \{ \langle\phi\rangle : \phi$ is a boolean formula in conjunctive normal form having a satisfying assignment $\}$.

$\texttt{SAT} \in \mathcal{NP}$. The certificate is the assignment. However, the number of assignments in exponential, so there is no obvious polynomial-time algorithm.

Primes and Composites

Consider the question of checking whether a number is prime or of finding a factorization.

Note that an integer m is *input in binary*; thus length of input is $\log_2 m$. That is, we want algorithms that run in time polynomial in the number of bits.

`PRIME` is the set of prime numbers (in binary) and `COMPOSITE` is the set of composite numbers.

Primes and Composites

Clearly `COMPOSITE` $\in \mathcal{NP}$: simply guess a split into two factors and then verify by multiplying. From elementary number theory comes a certificate for primeness, so `PRIME` $\in \mathcal{NP}$.

Recently, it was shown that `PRIME` is in $\mathcal{P}$, and hence so is `COMPOSITE`, since $\mathcal{P}$, being deterministic, is closed under complementation. Nevertheless, there is still no polynomial-time algorithm known for determining the factorization of a composite number.

Example: `SUBSET_SUM`

In the `SUBSET_SUM` problem, one is given a collection of numbers in binary, and a target. The question is: is there a subset of the numbers that adds up to the target.

The obvious certificate works: the certificate lists the correct numbers to take. The checker has only to sum the numbers and compare.

$\mathcal{P}$ *versus* $\mathcal{NP}$

Maybe $\mathcal{P}$ and $\mathcal{NP}$ are different sets. However, we do not know.

Conjecture. $\mathcal{P} \neq \mathcal{NP}$

The Clay Institute offers $1 million for a proof or disproof.

Consequences

One can, however, identify problems that are the hardest in $\mathcal{NP}$, called $\mathcal{NP}$-***complete problems***. They have the property that, if there is a polynomial-time algorithm for any one of them, then there is a polynomial-time algorithm for all of $\mathcal{NP}$.

There are numerous $\mathcal{NP}$-complete problems that industry would love to solve quickly. But, almost all cryptography assumes that decoding without the secret key is harder than decoding with the secret key, which might not be true if $\mathcal{P} = \mathcal{NP}$.

Summary

The class $\mathcal{NP}$ is the set of all languages that are decidable by a nondeterministic TM running in polynomial time. Such a machine is equivalent to a deterministic machine that is handed a certificate to verify the answer.