P and NP

Definition: P is the class of languages that are decidable in polynomial time on a deterministic Turing machine,

$$P = \bigcup_k TIME(n^k), \text{ for } k \geq 0.$$

Definition:

$$NP = \bigcup_k NTIME(n^k), \text{ for } k \geq 0.$$

Observation: In order to prove that a language is a member of a particular complexity class we simply have to demonstrate than an appropriate algorithm exists.

We have seen this in the case of the directed path in a graph.

Properties of P and NP

Theorem: The complexity class P is closed under complementation.

Proof: Any language $L \in P$ can be decided in deterministic polynomial time. Let M be such a decider for L. To show that P is closed under complementation we show that we can construct a deterministic polynomial time decider M' for $\overline{L}$,

M' = "On input w, where w is a string:

1. Run M on w.
2. If M accepts, *reject*; if M rejects, *accept*."

It is easy to see that this machine runs in deterministic polynomial time. □

Properties of P and NP

Theorem: The complexity class NP is closed under the Kleene-closure.

Proof: Any language L in NP is decided by some nondeterministic polynomial time TM. Let M be such a decider for L. To show that NP is closed under the Kleene-closure we need to show that $L^* \in NP$, where

$$L^* = \{w | w = \emptyset \text{ or } w = w_1 w_2 \ldots w_k \text{ for } k \geq 1 \text{ and each } w_i \in L\}.$$

We construct a nondeterministic polynomial time TM M' that decides L^*,

M' = "On input w, where w is a string:

1. If $w = \emptyset$, then *accept*.
2. Nondeterministically split w into the strings $w_1 w_2 \ldots w_k$ for $k \geq 1$.
3. Run M on each string w_i.
4. If M accepts all w_i's, *accept*; otherwise, *reject*."

By realizing that the number of times the machine M is invoked is bounded by $O(n)$ (each $|w_i| = 1$ with $n = |w|$) and the fact that M is a nondeterministic polynomial time TM, say $O(n^m)$, then the total nondeterministic polynomial runtime is $O(n^{m+1})$. Therefore, $L^* \in NP$. □

Hamiltonian Path

A Hamiltonian path in a directed path is a directed path that goes through each node exactly once. Formally, $HAMPATH = \{\langle G, s, t\rangle | G$ is a directed graph with a Hamiltonian path from s to $t\}$.

No deterministic polynomial time algorithms are know that decide this language.

Theorem:

$$HAMPATH \in NP$$

Proof: We construct an nondeterministic Turing machine that decides $HAMPATH$ in polynomial time.

M = "On input $\langle G, s, t\rangle$:

1. Nondeterministically generate a permutation of m numbers $p_1, \ldots, p_m$ such that $1 \leq p_i \leq m$ where m is the number of nodes in graph G.
2. Check whether $p_1 = s$ and $p_m = t$. If either test fails, *reject*.
3. For each i between 1 and $m - 1$, check wether (p_i, p_{i+1}) is an edge in G. If any are not, *reject*. Otherwise, the generated list of numbers represents a Hamiltonian path, *accept*."

Analysis. It is easy to see that all the stages run in polynomial time. □

Verifiers

We can define the class NP in an alternative manner using deterministic polynomial time verifiers.

Definition: A ***verifier*** for a language A is a deterministic TM V, where

$$A = \{w | V \text{ accepts } \langle w, c \rangle \text{ for some string } c\}.$$

Here the string c is called a ***certificate***.

We measure the time of a verifier in terms of the length of w, that is, a polynomial time verifier run in polynomial time in the length of w.

A language is ***polynomially verifiable*** if it has a (deterministic) polynomial time verifier.

Definition: NP is the class of languages that have (deterministic) polynomial time verifiers.

Hamiltonian Path (revisited)

Theorem:

$$HAMPATH \in NP$$

Proof #2: This time we show that a polynomial time verifier exists for a Hamiltonian path.

Let c be a Hamiltonian path $\langle p_1 \rightsquigarrow p_m \rangle$, the we construct the verifier V as follows:

V = "On input $\langle \langle G, s, t \rangle, c \rangle$:

1. Verify that $|p_1 \rightsquigarrow p_m| = m - 1$. If not, *reject*.
2. Verify that $p_1 \rightsquigarrow p_m$ does not have any repetitions. If any are found, *reject*.
3. Check wether $p_1 = s$ and $p_m = t$. If either fails, *reject*.
4. For each i between 1 and $m - 1$, check wether (p_i, p_{i+1}) is an edge in G. If any are not, *reject*.
5. All test have passed, *accept*."

Deciding vs. Verifying

Theorem: A language is decided by a nondeterministic polynomial time TM iff it can be verified by a deterministic polynomial time verifier.

Proof: We show that nondeterministic deciders can be constructed from verifiers and vice versa.

(a) For the '$\Rightarrow$' direction: Let N be the nondeterministic TM that decides the language, then we can construct a corresponding verifier V as follows,

V = " On input $\langle w, c\rangle$, where w and c are strings:

1. Simulate N on w but only follow computations that are described in c (this means N will only have a single branch of computation).
2. If this branch of computation accepts, *accept*; otherwise, *reject*."

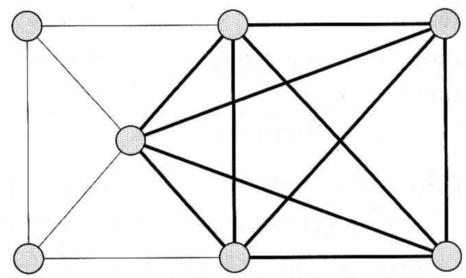
(b) For the '$\Leftarrow$' direction: Let V be a polynomial time verifier with runtime n^k, then we can construct the corresponding nondeterministic TM N,

N = "On input w of length n:

1. Nondeterministically generate a string c with $|c| \leq n^k$.
2. Run V on $\langle w, c\rangle$.
3. If V accepts, *accept*; otherwise, *reject*."

Cliques

Example: Clique in a graph. A clique in an undirected graph is a subgraph wherein every two nodes are connected by an edge. A k-clique is a clique that has k node.



A 5-clique.

Formally, expressed as a language,

$$CLIQUE = \{\langle G, k\rangle | G \text{ is an undirected graph with a } k\text{-clique } \}.$$

$CLIQUE \in NP$

Theorem:

$$CLIQUE \in NP.$$

Proof #1: We construct a nondeterministic polynomial time decider.

N = "On input $\langle G, k \rangle$:

1. Nondeterministically select a set Q of k nodes where each node is in G.
2. Test whether G contains all edges connecting nodes in Q.
3. If yes, *accept*; otherwise, *reject*."

Stage 2 runs in $O(n^2)$ with $n = |\langle G, k \rangle|$. Therefore, there whole machine runs in nondeterministic polynomial time.

$CLIQUE \in NP$

Proof #2: Let c be a k-clique on G, then construct a verifier,

V = "On input $\langle\langle G, k\rangle, c\rangle$:

1. Test whether c is a set of k nodes in G.
2. Test whether G contains all edges connecting nodes in Q.
3. If all tests pass, *accept*; otherwise, *reject*."

Here, stage 1 and 2 run in $O(n^2)$ time, therefore the verifier runs in deterministic polynomial time.

P vs NP

Since a TM is considered a special case of an nondeterminisic TM we have,

$$P \subset NP$$

It is still an open question whether $P = NP$, since currently the best known deterministic algorithms for NP problems use exponential time,

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

(Remember: to simulate a nondeterministic TM on a TM we need exponential time.)

NP-Completeness

Definition: A function $f: \Sigma^* \longrightarrow \Sigma^*$ is a ***polynomial time computable function*** if some (deterministic) polynomial time Turing machine M, on every input w, halts with just $f(w)$ on its tape.

Definition: Language A is ***polynomial time mapping reducible***, or simply ***polynomial time reducible***, to language B, written $A \leq_p B$, if a polynomial time computable function $f: \Sigma^* \longrightarrow \Sigma^*$ exists, where for every w,

$$w \in A \Leftrightarrow f(w) \in B.$$

The function f is call the ***polynomial time reduction*** from A to B.

NP-Completeness

Theorem: If $A \leq_p B$ and $B \in P$, then $A \in P$.

Proof: Let M be a polynomial time decider for B and let f be a polynomial time reduction from A to B, then we can construct a polynomial time decider N for A as follows:

N = "On input w:

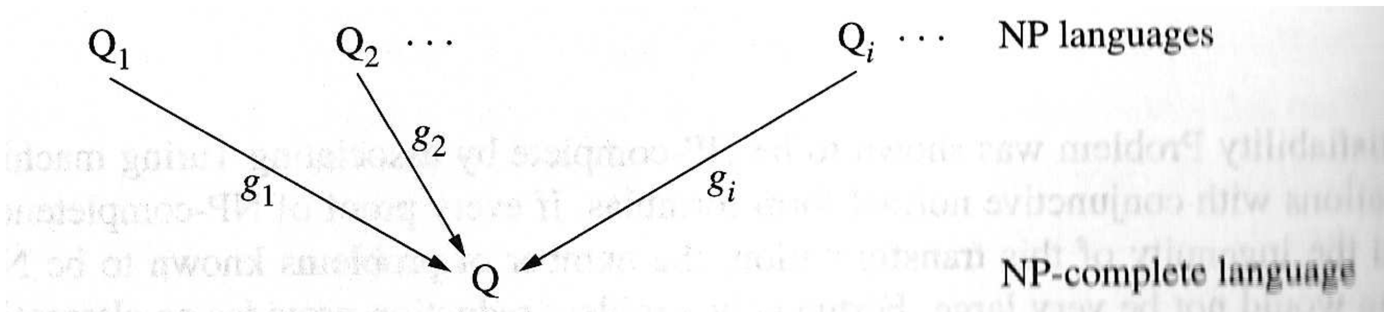
1. Compute $f(w)$.
2. Run M on $f(w)$ and output whatever M outputs."

Clearly, if $w \in A$ then $f(w) \in B$ since f is a reduction. It is easy to see that N runs in polynomial time.□

NP-Completeness

Definition: A language Q is ***NP-complete*** if it satisfies two conditions:

1. $Q \in NP$, and
2. every $Q_i \in NP$ is polynomial time reducible to Q.



NP-Completeness

Theorem: If B is NP-complete and $B \in P$, then $P = NP$

This theorem highlights the importance of NP-complete problems, should a deterministic polynomial time solutions be found to an NP-complete problem, then the NP complexity class will collapse into the P complexity class.

NP-Completeness

Theorem: If B is NP-complete and $B \leq_p C$ for $C \in NP$, then C is NP-complete.

Proof: Let g_i be a polynomial time reduction from any language $A_i \in NP$ to B and let f be the polynomial time reduction from B to C. We know that g_i has to exist for all languages $A_i \in NP$ since B is NP-complete. This gives us a polynomial time reduction $f \circ g_i$ from any language $A_i \in NP$ to C. $\square$