Mathematical Foundation of Computer Sciences VI

Time Complexity

Guoqiang Li
School of Software, Shanghai Jiao Tong University

SHANGHAI JIAO TONG UNIVERSITY

Note

Even when a problem is decidable, it might not be solvable in practice, since the optimal Turing machine which decides this problem could require astronomical time.

Measuring Time

$A = \{0^k 1^k \mid k \geq 0\}$

M_1 on w:

1. Scan across the tape and reject if a 0 is found to the right of a 1.
2. Repeat if both 0s and 1s remain on the tape.
   - Scan across the tape, crossing off a single 0 and a single 1.
3. If 0s still remain after all the 1s have been crossed off, or if 1s still remain after all the 0s have been crossed off, reject.
   Otherwise if neither 0s or 1s remain on the tape, accept.

Time complexity of M_1

1. Analyze the running time of M_1 on every $x \in \Sigma^*$,

$$f_1 : \Sigma^* \to \mathbb{N}$$

2. Analyze the worst-case running time of M_1 on inputs of length $n \in \mathbb{N}$, $f_2 : \mathbb{N} \to \mathbb{N}$. In particular

$$f_2(n) = \max_{x \in \Sigma^n} f_1(x)$$

3. Analyze the average-case running time of M_1 on inputs of length $n \in \mathbb{N}$, $f_3 : \mathbb{N} \to \mathbb{N}$. In particular

$$f_3(n) = \frac{\sum\limits_{x \in \Sigma^n} f_1(x)}{|\Sigma|^n}$$

Worst-case analysis

Definition

Let M be a deterministic Turing machine that halts on all inputs. The running time or time complexity of M is the function $f : \mathbb{N} \to \mathbb{N}$, where $f(n)$ is the maximum number of steps that M uses on any input of length n.

If $f(n)$ is the running time of M, we say that M runs in time $f(n)$ and that M is an $f(n)$ time Turing machine.

Customarily we use n to represent the length of the input.

Big-O Notation

Definition

Let $f, g \in \mathbb{N} \to \mathbb{R}^+$ be two functions. Say that $f(n) = O(g(n))$ if positive integers c and n_0 exist such that for every integer $n \geq n_0$

$$f(n) \leq c \cdot g(n)$$

When $f(n) = O(g(n))$, we say that $g(n)$ is an upper bound for $f(n)$, or more precisely, that $g(n)$ is an asymptotic upper bound for $f(n)$, to emphasize that we are suppressing constant factors.

Examples

SHANGHAI JIAO TONG UNIVERSITY

$5n^3 + 2n^2 + 22n + 6 = O(n^3)$.

Let $b \geq 2$. Then $\log_b n = \dfrac{\log_2 n}{\log_2 b}$ Hence, $\log_b n = O(\log n)$.

$3n \log_2 n + 5n \log_2 \log_2 n + 2 = O(n \log n)$.

$2^{10n^2+7n-6} = 2^{O(n^2)}$.

Poly. vs. Expo.

n^c for $c > 0$ is a polynomial bound.

$2^{(n^\delta)}$ for $\delta > 0$ is an exponential bound.

Small o-notation

Definition

Let $f, g \in \mathbb{N} \to \mathbb{R}^+$ be two functions. Say that $f(n) = o(g(n))$ if

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

In other words, f(n) = o (g(n)) means that for any real number $c > 0$, a number n_0 exists, where $f(n) < c \cdot g(n)$ for all $n \geq n_0$.

Examples

$\sqrt{n} = o(n)$.

$n = o(n \log \log n)$.

$n \log \log n = o(n \log n)$.

$n \log n = o(n^2)$.

$n^2 = o(n^3)$.

$A = \{0^k 1^k \mid k \geq 0\}$

M_1 on w:

1. Scan across the tape and reject if a 0 is found to the right of a 1.
2. Repeat if both 0s and 1s remain on the tape.
   - Scan across the tape, crossing off a single 0 and a single 1.
3. If 0s still remain after all the 1s have been crossed off, or if 1s still remain after all the 0s have been crossed off, reject.
   Otherwise if neither 0s or 1s remain on the tape, accept.

Time analysis

The first stage scans the tape to verify the input is of the form 0^*1^*, taking n steps. Then the machine repositions the head at the left-hand end of the tape, again using n steps. In total $2n = O(n)$ steps.

In stages 2 and 3, the machine repeatedly scans the tape and crosses off a 0 and 1 on each scan. Each scan uses $O(n)$ steps. Because each scan crosses off two symbols, at most $n = 2$ scans can occur. So the total time taken by stage 2 and 3 is $(n = 2)O(n) = O(n^2)$.

In stage 4, the machine makes a single scan to decide whether to accept or reject, hence require time $O(n)$.

The overall running time

$$O(n) + O(n^2) + O(n) = O(n^2)$$

Time classes

SHANGHAI JIAO TONG UNIVERSITY

Definition

Let $t : \mathbb{N} \to \mathbb{R}^+$ be a function. Define the time complexity class

$$TIME(t(n))$$

to be the collection of all languages that are decidable by an $O(t(n))$ time Turing machine.

Example

$\{0^k1^k \mid k \geq 0\} \in TIME(n^2)$.

A better algorithm

M_2 on w:

1. Scan across the tape and reject if a 0 is found to the right of a 1.
2. Repeat as long as some 0s and 1s remain on the tape.
    1. Scan across the tape, checking whether the total number of 0s and 1s remaining is even or odd. If it is odd, then reject.
    2. Scan again across the tape, crossing off every other 0 starting with the first 0, and then crossing off every other 1 starting with the first 1.
3. If no 0s and no 1s remain on the tape, then accept. Otherwise, reject.

Time analysis

Every stage takes $O(n)$ time.

Stage 1 and 3 are executed once, hence total $O(n)$ time.

Stage 2.2 crosses off at least half of the 0s and 1s each time it is executed, hence at most $1 + \log_2 n$ iterations.

Thus the total time of stages 2, 3 and 4 is $(1 + \log_2 n)O(n) = O(n \log n)$.

The overall running time of M_2 is

$$O(n) + O(n \log n) = O(n \log n)$$

Can we do even better than $O(n \log n)$?

Theorem

Every language that can be decided in $o(n \log n)$ time on a single-tape Turing machine is regular.

$\{0^k 1^k \mid k \geq 0\}$ in linear time on a 2-tape TM

M_3 on w:

1. Scan across tape 1 and reject if a 0 is found to the right of 1.
2. Scan across the 0s on tape 1 until the first 1. At the same time copy the 0s onto tape 2.
3. Scan across the 1s on tape 1 until the end of the input. For each 1 read on tape 1, cross off a 0 on tape 2. If all 0s are crossed off before all the 1s are read, then reject.
4. If all the 0s have now been crossed off, then accept. If any 0s remain, then reject.
5. If no 0s and no 1s remain on the tape, then accept.
   Otherwise, reject.

Complexity relationships among models

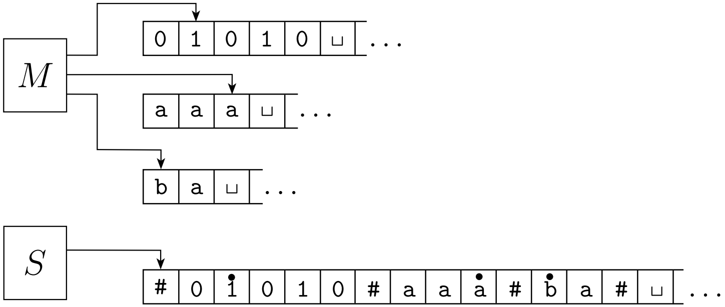
Theorem

Let $t(n)$ be a function with $t(n) \geq n$. The every $t(n)$ time multitape Turing machine has an equivalent $O(t^2(n))$ time single-tape Turing machine.

Proof (1)

We simulate an M with k tapes by a single-tape S.

- S uses $\#$ to separate the contents of the different tapes.
- S keeps track of the locations of the heads by writing a tape symbol with a dot above it to mark the place where the head on that tape would be.



Proof (2)

On input $w = w_1 \dots w_n$:

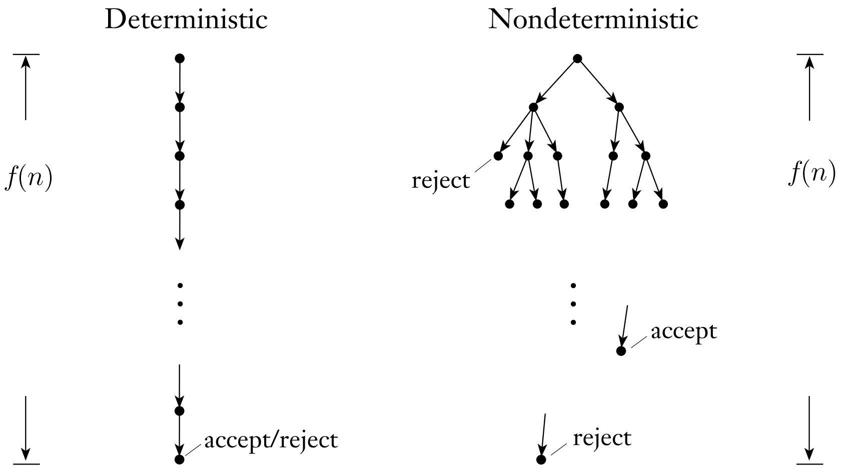
1. First S puts its tape into the format that represents all k tapes of M:
$$\#\dot{w}_1 w_2 \dots w_n \# \dot{\sqcup} \# \dot{\sqcup} \dots \# \dot{\sqcup} \#$$
Time: $O(n) = O(t(n))$.
2. To determine the symbols under the virtual heads, S scans its tape from the first $\#$, which marks the left-hand end, to the $(k+1)$st $\#$, which marks the right-hand end.
Time: $O(t(n))$.
3. Then S makes a second pass to update the tapes according to the way that Ms transition function dictates. If S makes one of the virtual heads to the right onto a $\#$, then S writes $\sqcup$ on this tape cell and shifts the tape contents, from this cell until the rightmost $\#$, one unit to the right.
Time: $O(k \cdot t(n)) = O(t(n))$.
4. Go back to 2.

Nondeterministic machines

Definition

Let N be a nondeterministic Turing machine that is a decider. The running time of N is the function $f : \mathbb{N} \to \mathbb{N}$, where $f(n)$ is the maximum number of steps that N uses on any branch of its computation on any input of length n.

Deterministic vs. nondeterministic



Complexity relationships among models (cont'd)

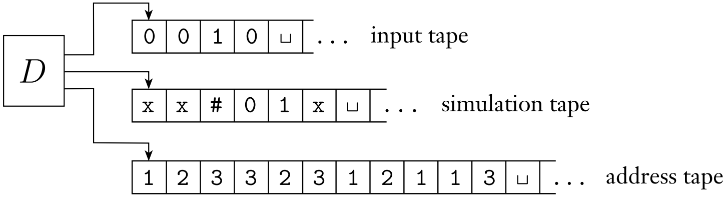
Theorem

Let $t(n)$ be a function with $t(n) \geq n$. The every $t(n)$ time nondeterministic single-tape Turing machine has an equivalent $2^{O(t(n))}$ time deterministic single-tape Turing machine.

Proof (1)

We simulate a nondeterministic N by a deterministic D.

- D try all possible branches of N's nondeterministic computation.
- If D ever finds the accept state on one of these branches, it accepts.



Proof (2)

On an input of length n, every branch of N's nondeterministic computation tree has a length of at most $t(n)$.

Every node in the tree can have at most b children, where b is the maximum number of legal choices given by N's transition function. Thus, the total number of leaves in the tree is at most $bt(n)$.

The total number of the nodes in the tree is less than twice the maximum number of leaves, hence $O(b^{t(n)})$. The time it takes to start from the root and travel down to a node is $O(t(n))$. Hence the total running time of D is $O(t(n)b^{t(n)}) = 2^{O(t(n))}$.

D has 3 tapes, thus can be simulated by a single-tape TM in time

$$(2^{O(t(n))})^2 = 2^{O(t(n))}$$

The Class P

P

Definition

P is the class of languages that are decidable in polynomial time on a deterministic single-tape Turing machine. In other words:

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

P

P is invariant for all models of computation that are polynomially equivalent to the deterministic single-tape machine.

P roughly corresponds to the class of problems that are realistically solvable on a computer.

Examples of Problems in P

Reasonable encodings

We continue to use $\langle\cdot\rangle$ to indicate a reasonable encoding of one or more objects into a string.

Unary encoding of n as $\underbrace{11\ldots11}_{n \text{ times}}$ is exponentially larger than the standard binary encoding of n, hence not reasonable.

A graphs can be encoded either by listing its nodes and edges, i.e., its adjacency list, or its adjacency matrix, where the (i,j)th entry is 1 if there is an edge from node i to node j and 0 if not.

The path problem

$$PATH = \{\langle G, s, t\rangle \mid \quad G \text{ is a directed graph that has a directed path from } s \text{ and } t\}$$

Theorem

$PATH \in P$.

Testing relative prime

$$RELPRIMIE = \{\langle x, y\rangle \mid x \text{ and } y \text{ are relatively prime}\}$$

Theorem

$RELPRIMIE \in P$.

The Euclidean Algorithm

Recall the greatest common divisor $gcd(x, y)$ is the largest integer that divides both x and y.

E on $\langle x, y \rangle$

```
1 Repeat until $y = 0$:
2         Assign $x \leftarrow x \pmod y$.
3 Exchange $x$ and $y$.
4 Output $x$.
```

Testing relative prime

R on $\langle x, y \rangle$

1. Run E on $\langle x, y \rangle$.
2. If the result is 1, then accept. Otherwise, reject.

Time analysis

We show that E runs in polynomial time

- Every execution of stage 2 with $y \leq x$ cuts the value x at least by half.
- Thus, the maximum number of times that stage 2 and 3 are executed is the lesser of $2\log_2 x$ and $2\log_2 y$.

Testing context-freeness

Theorem

Every context-free language is a member of P.

Recall (1)

SHANGHAI JIAO TONG UNIVERSITY

Definition

A context-free grammar is in Chomsky normal form if every rule is of the form

$$A \to BC \text{ and } A \to a$$

where a is any terminal and A, B and C are any variables, except that B and C may be not the start variable. In addition, we permit the rule $S \to \epsilon$, where S is the start variable.

Theorem

Any context-free language is generated by a context-free grammar in Chomsky normal form.

Theorem

Let G be CFG in Chomsky normal form, and G generates w with $w \neq \epsilon$. Then any derivation of w has $2|w| - 1$ steps.

Recall (2)

S on $\langle G, w\rangle$

1. Convert G to an equivalent grammar in Chomsky normal form.
2. List all derivations with $2|w| - 1$ steps; except if $|w| = 0$, then instead check whether there is a rule $S \to \epsilon$.
3. If any of these derivations generates w, then accept; otherwise reject.

The running time of S is $2^{O(n)}$.

Dynamic programming

Let w be an input string and $n := |w|$.

For every $i \leq j \leq n$ we will compute

$table(i, j)$ = the collection of variables that can generate the substring $w_i w_{i+1} \dots w_j$.

Dynamic Programming (cont'd)

D on $w = w_1 \dots w_n$:

```
1  For $w = \epsilon$, if $S \to \epsilon$ is a rule, then accept; else reject.
2  For $i = 1$ to $n$:
3      For each variable $A$:
4          Test whether $A \to b$ is a rule, where $b = w_i$.
5          If so, place $A$ in table $(i, i)$.
6  For $\ell = 2$ to $n$:
7      For $i = 1$ to $n - \ell + 1$:
8          Let $j = i + \ell - 1$
9          For $k = i$ to $j - 1$:
10             For each rule $A \to BC$:
11             If $B \in table(i, k)$ and $C \in table(k, j)$,
               then put $A$ in table$(i, j)$.
12 If $S \in table(1, n)$, then accept; else reject.
```

The Class NP

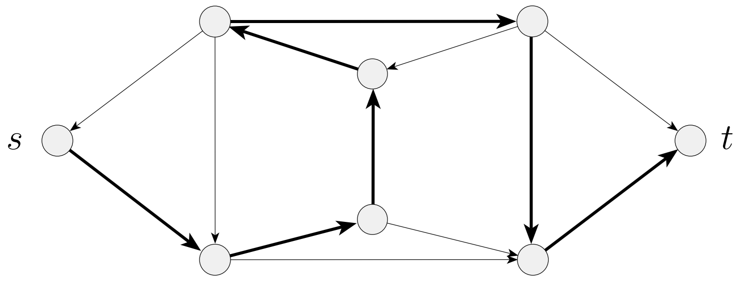
Hamiltonian path

Definition

A Hamiltonian path in a directed graph G is a directed path that goes through each node exactly once.

$$HAMPATH = \{\langle G, s, t\rangle \mid G \text{ is a directed graph with a Hamiltonian path from } s \text{and } t\}$$

Hamiltonian path (cont'd)



Polynomial verifiability

Even though we don't know how to determine fast whether a graph contains Hamiltonian path, if such a path were discovered somehow (perhaps using the exponential time algorithm), we could easily convince someone else of its existence simply by presenting it.

In other words, verifying the existence of a Hamiltonian path may be much easier than determining its existence.

Testing composite

Definition

A natural number is composite if it is the product of two integers > 1.

$$COMPOSITES = \{x \mid x = pq \text{ for integers } p, q > 1\}$$

Verifiers

Definition

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

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

We measure the time of a verifier only in terms of the length of w, so a polynomial time verifier runs in polynomial time in the length of w. A language A is polynomial verifiable if it has a polynomial time verifier.

The string c in the above definition is a certificate, or proof, of membership in A. For polynomial verifiers, the certificate has polynomial length (in the length of w).

Certificates

For $HAMPATH$, a certificate for $\langle G, s, t\rangle \in HAMPATH$ is a Hamiltonian path from s to t.

For $COMPOSITES$, a certificate for x is one of its divisors.

The class NP

Definition

NP is the class of languages that have polynomial time verifiers.

Nondeterministic polynomial Turing machines

Theorem

A language is in NP if and only if it is decided by some nondeterministic polynomial time Turing machines.

Proof (1)

Assume that the verifier V is a TM that runs in time n^k.

N on w with $n = |w|$

1. Nondeterministically select string c of length at most n^k.
2. Run V on $\langle w, c\rangle$.
3. If V accepts, then accept; otherwise, reject.

Proof (2)

Assume that A is decided by a polynomial time NTM N.

V on $\langle w, c \rangle$

1. Simulate N on input w, treating each symbol of c as a description of the nondeterministic choice to make at each step.
2. If this branch of N's computation accepts, then accept; otherwise, reject.

Nondeterministic time complexity classes

Definition

$$NTIME(t(n)) = \{L \mid L \text{ is a language decided by an } O(t(n)) \text{ time nondeterministic Turing machine}\}$$

Corollary

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

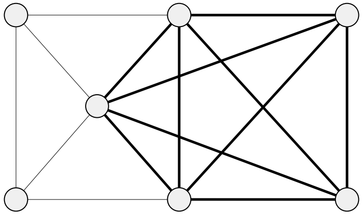
Examples of problems in NP

The clique problem

SHANGHAI JIAO TONG UNIVERSITY

Definition

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 contains k nodes.



The clique problem (cont'd)

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

Theorem

$CLIQUE$ is in NP.

Proof (1)

SHANGHAI JIAO TONG UNIVERSITY

V on $\langle\langle G, k\rangle, c\rangle$:

1. Test whether c is a subgraph with k nodes in G.
2. Test whether G contains all edges connecting nodes in c.
3. If both pass, then accept; otherwise reject.

Proof (2)

N on $\langle G, k\rangle$:

1. Nondeterministically select a subset c of k nodes in G.
2. Test whether G contains all edges connecting nodes in c.
3. If yes, then accept; otherwise reject.

The subset-sum problem

$$SUBSET\text{-}SUM = \{\langle S, t\rangle \mid \begin{array}{l} S = \{x_1, \ldots x_k\} \text{ and for some} \\ \{y_1 \ldots, y_\ell\} \subseteq S, we have \sum_{i\in[\ell]} y_i = t\} \end{array}$$

Theorem

$SUBSET\text{-}SUM$ is in NP.

The P versus NP question

P = the class of languages for which membership can be decided quickly

NP = the class of languages for which membership can be verified quickly

A remark

P versus NP, a gift to mathematics from computer science.

– S. Smale.

The NP-Completeness

Remark

In 1970s, Stephen Cook and Leonid Levin discovered certain problems in NP whose individual complexity is related to that of the entire class.

If a polynomial time algorithm exists for any of these problems, all problems in NP would be polynomial time solvable.

These problems are called NP-complete.

Satisfiability Problem

Boolean variables are assigned to $TRUE(1)$ or $FALSE(0)$.

Boolean operations are AND, OR, and NOT.

A Boolean formula is an expression involving Boolean variables and operations.

A Boolean formula is satisfiable if some assignment makes the formula evaluate to 1.

The satisfiability problem is to test whether a Boolean formula is satisfiable, i.e.,

$$SAT = \{\langle\varphi\rangle \mid \varphi \text{ is a satisfiable Boolean formula}\}$$

Satisfiability Problem

SHANGHAI JIAO TONG UNIVERSITY

Theorem

$SAT \in P$ *if and only if* $P = NP$.

Polynomial time reduction

Definition

A function $f : \Sigma^* \to \Sigma^*$ is a polynomial time computable function if some polynomial time Turing machine exists that halts with just $f(w)$ on its tape, when started on any input w.

Definition

Let $A, B \subseteq \Sigma^*$. Then A is polynomial time mapping reducible, or simply polynomial time reducible, to B, written $A \leq_P B$, if a polynomial time computable function $f : \Sigma^* \to \Sigma^*$ exists, where for every w

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

The function f is called the polynomial time reduction of A to B.

Polynomial time reduction

Theorem

If $A \leq_P B$ *and* $B \in P$, *then* $A \in P$.

3SAT

A literal is a Boolean variable or a negated Boolean variable.

A clause is several literals connected with $\vee$s.

A Boolean formula is in conjunctive normal form, called a cnf-formula, if it comprises several clauses connected with $\wedge$s.

A Boolean formula is a 3CNF-formula if all the clauses have three literals.

$$3SAT = \{\langle\varphi\rangle \mid \varphi \text{ is a satisfiable 3CNF-formula}\}$$

Result (1)

Theorem

$3SAT$ *is polynomial time reducible to* $CLIQUE$.

Proof (1)

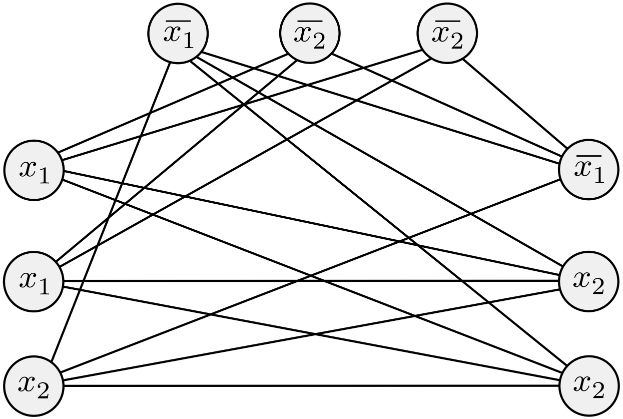
Let φ be a formula with k clauses such as

$$\varphi(a_1 \vee b_1 \vee c1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \ldots \wedge (a_k \vee b_k \vee c_k)$$

The reduction generates a string $\langle G, k\rangle$.

1. The nodes in G are organized into k groups of three nodes $t_1, \ldots, t_k$. Each triple corresponds to one of the clauses, and each node in a triple corresponds to a literal in the associated clauses.
2. The edges of G connect all but two types of pairs of nodes in G.
    - No edge is present between nodes in the same triple.
    - No edge is present between two nodes with contradictory labels, e.g., x_2 and $\overline{x_2}$.

Proof (2)



$$\varphi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$

NP-complete

SHANGHAI JIAO TONG UNIVERSITY

Definition

A language B is NP-complete if it satisfies two conditions:

1. B is in NP, and
2. every A in NP is polynomial time reducible to B.

Relative Results

Theorem

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

Theorem

If B is NP-complete and $B \leq_P C$ for C in NP, then C is NP-complete.

The first NPC problem

Theorem

SAT is NP-complete.

Proof (1)

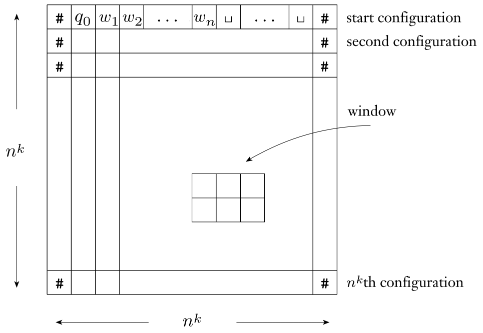
SAT is in NP, since a nondeterministic polynomial time Turing machine can

1. guess an assignment to a given formula φ,
2. accept if the assignment satisfies φ.

Proof (2)

Let N be an NTM that decides a language A in time n^k for some $k \in \mathbb{N}$. We show $A \leq_P SAT$.

A tableau for N on w is an $n^k \times n^k$ table whose rows are the configurations of the branch of the computation of N on input w.



Proof (3)

We assume that each configuration starts and ends with a $\#$ symbol. Therefore, the first and last columns of a tableau are all $\#$s.

The first row of the tableau is the starting configuration of N on w, and each row follows the previous one according to N's transition function.

A tableau is accepting if any row of the tableau is an accepting configuration.

Every accepting tableau for N on w corresponds to an accepting computation branch of N on w. Thus, the problem of determining whether N accepts w is equivalent to the problem of determining whether an accepting tableau for N on w exists.

Proof (4)

On input w, the reduction produces a formula φ.

1. Let Q and Γ be the state set and tape alphabet of N. We set

$$C = Q \cup \Gamma \cup \{\#\}$$

2. For each $i, j \in [n^k]$ and for each $s \in C$, we have a variable $x_{i,j,s}$.
3. Each of the $(n^k)^2$ entries of a tableau is called a cell.
4. If $x_{i,j,s}$ takes on the value 1, it means that the cell in row i and column j contains an s.

We represent the contents of the cells with the variables of φ.

Proof (5)

We design φ so that a satisfying assignment to the variables does correspond to an accepting tableau for N for w

$$\varphi_{cell} \wedge \varphi_{start} \wedge \varphi_{move} \wedge \varphi_{accept}$$

Proof (6)

$$\varphi_{cell} = \bigwedge_{i,j\in[n^k]} \left[\left(\bigvee_{s\in C} x_{i,j,s} \right) \left(\bigwedge_{\substack{s,t\in C,\\ s\neq t}} (\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}}) \right) \right]$$

Proof (7)

$$\begin{aligned}\varphi_{start} = \quad & x_{1,1,\#} \wedge x_{1,2,q_0} \wedge \\ & x_{1,3,w_1} \wedge x_{1,4,w_1} \wedge \ldots \wedge x_{1,n+2,w_n} \wedge \\ & x_{1,n+3,\sqcup} \wedge \ldots \wedge x_{1,n^k-1,\sqcup} \wedge x_{1,n^k,\#}\end{aligned}$$

Proof (8)

SHANGHAI JIAO TONG UNIVERSITY

$$\varphi_{accept} = \bigvee_{i,j \in [n^k]} x_{i,j,q_{accept}}$$

Proof (9)

Finally, formula φ_{move} guarantees that each row of the tableau corresponds to a configuration that legally follows the preceding row's configuration according to N's rules.

It does so by ensuring that each 2×3 window of cells is legal.

We say that a 2×3 window is legal if that window does not violate the actions specified by N's transition function.

Proof (10)

Assume that:

- When in state q_1 with the head reading an a, N writes a b, stays in state q_1, and moves right.
- When in state q_1 with the head reading a b, N nondeterministically either
  - writes a c, enters q_2, and moves to the left, or
  - writes an a, enters q_2, and moves to the right.

(a)

a	q_1	b
q_2	a	c

(b)

a	q_1	b
a	a	q_2

(c)

a	a	q_1
a	a	b

(d)

#	b	a
#	b	a

(e)

a	b	a
a	b	q_2

(f)

b	b	b
c	b	b

Legal moves

Proof (11)

Shanghai Jiao Tong University

Assume that:

- When in state q_1 with the head reading an a, N writes a b, stays in state q_1, and moves right.
- When in state q_1 with the head reading a b, N nondeterministically either
  - writes a c, enters q_2, and moves to the left, or
  - writes an a, enters q_2, and moves to the right.

(a)

a	b	a
a	a	a

(b)

a	q_1	b
q_2	a	a

(c)

b	q_1	b
q_2	b	q_2

Illegal moves

Proof (12)

If the top row of the tableau is the start configuration and every window in the tableau is legal, each row of the tableau is a configuration that legally follows the preceding one.

Proof (13)

$$\varphi_{move} = \bigwedge_{1 \leq i,j < n^k} \text{the } (i,j)\text{-window is legal}$$

We replace the (i,j)-window is legal by

$$\bigvee_{\substack{a_1,\ldots,a_6 \\ \text{is a legal window}}} x_{i,j-1,a_1} \wedge x_{i,j,a_2} \wedge x_{i,j+1,a_3} \wedge x_{i+1,j-1,a_4} \wedge x_{i+1,j,a_5} \wedge x_{i+1,j+1,a_6}$$

Corollary

Corollary

3SAT is NP-complete.

Additional NP-complete problems

Corollary

Corollary

CLIQUE is NP-complete.