Complexity of problems

Grau-AA

Solve problems efficiently

but .... there exist Hard Problems!!!!!

Solve problems efficiently

but .... there exist Hard Problems!!!!!

Algorithmic Solutions

Hardness w.r.t. computational resources required to be solved

Basic concepts of Theory of Computation

- *Algorithms* are modelized as Turing Machines
- *Decision problems* are expressed as formal languages.
- *Functional problems* are expressed as functions.

Types of problems

Decision problem:
Given an *input* x and a *property* Q
we wish to decide if $Q(x)$ is true.

Examples:

Factoring
INPUT: A n-bit integer x, and $k \in \mathbb{N}$ s.t. $(1 \leq k \leq x)$
QUESTION: Decide if there is a prime integer $y \leq k$ that is a factor of x.

Maximum Common Divisor
INPUT: Given two n-bit integers x_1, x_2, and a integer bound $k > 0$
QUESTION: Decide if there is an integer y such that y divides x_1, x_2 with $y \geq k$.

Minimum spanning tree
INPUT: Given $G = (V, E)$, $w : E \to \mathbb{R}^+$ and $k \in \mathbb{N}$.
QUESTION: Decide if there is a spanning tree of weight $\leq k$.

Types of problems

Function problem:
Given an *input* x and a *predicate* Q
we wish to compute y such that $Q(x, y)$.

Examples:

Factoring
INPUT: A n-bit integer x.
QUESTION: Find all the prime factors of x.

Maximum Common Divisor
INPUT: Given two n-bit integers x_1, x_2.
QUESTION: Find the maximum integer y such that y divides x_1, x_2.

Minimum spanning tree
INPUT: Given $G = (V, E)$, $w : E \to \mathbb{R}^+$.
QUESTION: Find a spanning tree with minimum weight.

Paradigmatic example: Satisfiability

SAT

INPUT: A boolean formula $\phi = \bigwedge_{i=1}^{m}(C_i)$ in Conjunctive Normal Form (CNF), over a set of boolean variables $X = \{x_1, \ldots, x_n\}$.

QUESTION: Decide if there is a $A : X \to \{0, 1\}$ such that $A \models \phi$.

i.e. there is a truth assignment $A : X \to \{0, 1\}$ s.t.
$A(\phi) = \bigwedge_{i=1}^{m} \phi(A(C_i)) = 1$.

The input size of a SAT problem is the length of the input formula $|\phi|$.

The 3-SAT problem is the variant of SAT, where each clause has exactly 3 literals.

$$\phi = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3)$$

Basic concepts of Theory of Computation

- *Algorithms* are modelized as Turing Machines
- *Decision problems* are expressed as formal languages.
- *Functional problems* are expressed as functions.
- An *algorithm solves a decision/function problem* is expressed as a TM decides/computes the corresponding language/function
- All reasonable *computational models* are *equivalent*
  That is, any one of them can simulate another.

Important Classes of Decision Problems

- Undecidible: No algorithm solves the problem.
  The halting problem: Given a program P and its description $< P >$, decide if $P(< P >)$ halts.
- Decidible: There is al algorithm which solves the problem.
  (Even if the computation time is $n^{n^{n^{n}}}$ .....).

We focus our study on decidable/computable problems.

Basic concepts of Complexity Theory

- All reasonable *deterministic computational models* are *polynomially equivalent*.

  That is, any one of them can simulate another with only a polynomial increase in running time.
- An *algorithm solves a problem in polynomial/exponential time* is expressed as a *TM decides/computes* the corresponding language/function in *polynomial/exponential time*

- In the classical *Complexity Theory* we focus on aspects of time complexity theory that are unaffected by polynomial differences in running time
- Doing so, allows us to develop the theory in a way that does not depend on the selection of a particular *model of computation*.

  But, you may feel that disregarding polynomial differences is absurd
- *In Algorithmics:* we certainly care about such differences

  The difference between time n and time n^3 is important!

  In Complexity: The *polynomiality* or *non polynomiality* of Travelling Salesman Problem do not depend on polynomial diffferences!

The complexity class P

The class P consists of those *decision* problems that are solvable in polynomial time
(i.e. A *decision problem* $A = \{x | Q(x)\}$ is in P if there exists an algorithm $\mathcal{A}$ and a constant c such that for any x,
$\mathcal{A}$ on input x halts in $O(|x|^c)$ steps and returns YES when $Q(x)$ is true or returns NO otherwise.)

Examples:

- Graph Accessibility Problem
  INPUT: Given a directed graph $G = (V, E)$ and two nodes $s, t \in G$
  QUESTION: Decide if there is a path from s to t.
  (BFS algorithm)
- Shortest path (decision version)
  INPUT: Given a directed graph $G = (V, E)$, two nodes $s, t \in G$ and a natural k
  QUESTION: Decide if there is a path from s to t of length less than or equal to k.
  (Dijkstra's shortest path algorithm)
- Longest Common Subsequence
  INPUT: Given sequences $x_1, x_2, \dots, x_n$ and $y_1, y_2, \dots, y_m$ and an integer $k > 0$, .
  QUESTION: Decide if there is a longest common subsequence with length $\geq k$.
  (Dynamic programming to solve the function version of LCS, and compare the result with k)

More examples in P

The decision versions of
MCD (Euclid),
Minimum Spanning Tree (Jarnik-Prim),
Fractional Knapsack (Greedy).

All these function problems belong to the class FP: The class of function problems that their explicit solution can be found in polynomial time $O(n^c)$ for $c =$constant, where n is the size of the input.

All problems in FP, their decision version is also in P.

Linear Programming.

Linear Programming (LP).
INPUT: $A = (a_{ij})_{1 \leq i \leq m, 1 \leq j \leq n}$, $\vec{c} = \{c_i\}_{i=1}^{n}$ and $\vec{b} = \{b_i\}_{i=1}^{m}$.
QUESTION: Find a set of real variables $\vec{x} = \{x_i\}_{i=1}^{n}$, such that optimize $\vec{c}^T \vec{x}$, subject to some constrain restrictions.

$$
\begin{aligned}
&\max \vec{c}^T \vec{x} \\
&\text{subject to:} \\
&\qquad A\vec{x} \leq \vec{b} \\
&\qquad \vec{x} \geq \vec{0}
\end{aligned}
$$

where A the $m \times n$ matrix of the variables involved in the linear constrains.

Example.

$$\begin{aligned} \max 100x_1 + 600x_2 + 1400x_3 \\ x_1 &\le 200 \\ x_2 &\le 300 \\ x_1 + x_2 + x_3 &\le 400 \\ x_2 + 3x_3 &\le 600 \\ x_1, x_2, x_3 &\ge 0. \end{aligned}$$

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 3 \end{pmatrix} ; \vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} ; \vec{b} = \begin{pmatrix} 200 \\ 300 \\ 400 \\ 600 \end{pmatrix} ; \vec{c} = \begin{pmatrix} 100 \\ 600 \\ 1400 \end{pmatrix}$$

Decision version of LP

d-LP.
INPUT: $m \times n$ matrix A and an m-dimensional vector b.
QUESTION: Decide if there is a vector $x = \{x_1, x_2, \ldots, x_n\}$ ($x_i \in \mathbb{R}^+$) satisfying $Ax \leq b$.
The (real) LP has a polynomial-time solution (interior methods).

- Dantzig's Simplex algorith (1947) although an excellent practical algorithm, uses an exponential number of steps for some particular inputs.
- Kachiyan's algorithm (1979) or Ellipsoid Algorithm, polynomial time, but didn't compete well with Simplex in practice.
- Karmarkar's algorithm (1984) the interior point method. It runs probably in polynomial time, it performs well.

The class NP

The class NP is the class of decision problems such that if we provide a *certificate* of a solution with polynomial size, we can *verify* in polynomial time that the certificate is indeed a solution.

$A \in \mathrm{NP}$ iff there exists $B \in \mathrm{P}$ and a polynomial p such that
$A = \{x | \exists y \, |y| \leq p(|x|) \, \langle x, y \rangle \in B\}$
y is the *certificate*
B is the *verifier* (in fact the algorithm deciding B)

NP stands for Nondeterministic Polynomial time.

Examples

- Maximum Cut.

INPUT: $G = (V, E)$, $|V| = n$ and $k \in \mathbb{N}$.
QUESTION: Decide if there is a partition V_1, V_2 of V, such that the number of edges between V_1 and V_2 is greater than or equal to k.

Certificate: The partition V_1, V_2. *To verify*, find all the edges between V_1 and V_2 (time $O(n^2)$) sum them and check that $\geq k$.

- Minimum Cut.

INPUT: $G = (V, E)$, $|V| = n$ and $k \in \mathbb{N}$.
QUESTION: Decide if there is a partition V_1 and V_2 of V, such that the number of edges between V_1 and V_2 is less than or equal to k.

Certificate: The partition V_1, V_2 (same as above).

Examples

- Composite.

INPUT: $x \in \mathbb{N}$.
QUESTION: Decide if x composite.
Certificate: $p, q \in \mathbb{N}$. *To verify* check in $O(|x|^2)$ that $x = p \cdot q$, (notice in this problem the length of input x is the number of bits to represent x)

- 3-SAT.

INPUT: $\phi = \bigwedge_{i=1}^{m} (C_i)$, on $X = \{x_1, \ldots, x_n\}$, where each $|C_i| = 3$.
QUESTION: Decide if there is $A : X \to \{0, 1\}$ s.t. $A(\phi) = 1$.
Certificate: $A : X \to \{0, 1\}$. *To verify* check in $O(3m)$ that at least one literal of every C_i is set to 1.

- Subset Sum.

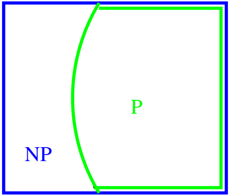
INPUT: sequence of positive integers $\{a_1, \ldots, a_n\}$ and $k \in \mathbb{Z}$.
QUESTION: Decide if it exists a $A \subseteq \{1, \ldots, n\}$ s.t. $\sum_{i \in A} a_i = k$.
Certificate: The set A.

Examples

•Integer Linear Programming.
INPUT:$A = (a_{ij})_{1\leq i\leq m, 1\leq j\leq n}$, $\vec{b} = \{b_i\}_{i=1}^m$, $\vec{c} = \{c_i\}_{i=1}^n$, , and finally a goal k.
QUESTION: Decide if there is an integer vector $\vec{x} = \{x_i\}_{i=1}^n$ such that $\sum_{i=1}^n c_i x_i \geq k$ subject to the constrains $A\vec{x} \leq \vec{b}$.
Certificate: $\vec{x} = \{x_i\}_{i=1}^n$. *To verify* check in $O(n^2)$ that the constrains are satisfied and the objective function $\sum_{i=1}^n c_i x_i \geq k$.

• Minimum spanning tree.
INPUT: $G(V, E)$ with $w : E \to \mathbb{Z}$ and $k \in \mathbb{Z}$.
QUESTION: Decide if there is a spanning tree T with total weight $\leq k$.
Certificate: T. *To verify* check in $O(n^2)$ that T does not contain cycles, $V(T) = V(G)$ and the sum of the weights of the edges is $\leq k$.

Notice: P $\subseteq$ NP.



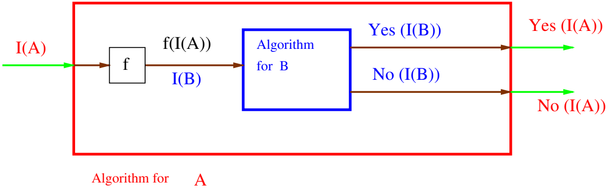
The US\$ 10^6 Question: Is $P \neq NP$ or $P = NP$?
http://www.claymath.org/prizeproblems/pvsnp.html

Reducibility

Let A, B be languages.
A is reducible or can be reduced to B ($A \leq_m^p B$)
if there exists a polynomial-time computable function f (reduction function) which transforms inputs of A to *equivalent* inputs of B.
That is,

- $\forall x$, $x \in A$ iff $f(x) \in B$.
- *Extra-requirement:* For every x of size n, f should be computed in polynomial time (in n).

Polynomial-time reducibility



Reduction f from problem A to problem B

If B can be recognized by a polynomial time algorithm, and there exists a polynomial time computable reduction from $f : A \leq_m^p B$, then we also have a polynomial-time algorithm recognizing A.

Therefore, *if $A \leq_m^p B$, then A is not harder than B*

Polynomial-time reducibility

Lemma

Let A, B be languages.

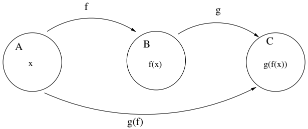
If $A \leq_m^p B$ and $B \in P$, then $A \in P$.

If $A \leq_m^p B$ and $B \in NP$, then $A \in NP$.

P and NP are closed under $\leq_m^p$

Lemma

For languages A, B, C, if $A \leq_m^p B$ and $B \leq_m^p C$, then $A \leq_m^p C$



Sketch: A composition of $\mathcal{A}$ with the algorithms computing f and g, $\mathcal{A}(g(f(x)))$.
Notice that $|g(f(x))| = poly(|x|)$

SAT $\leq_m^p$ 3SAT

Recall that SAT is a set of n variables and formula $\phi = C_1 \wedge C_2 \wedge \cdots C_m$ where every clause i is the $\vee$ of k_i literals $(1 \leq k_i \leq n)$. The problem consists in finding an assignment $A : X \to \{0, 1\}$ s.t. $A(\phi) = 1$.
The 3SAT problem is restricted version of SAT where each clause has exactly 3 literals.
Given an instance ϕ for SAT, we have to *reduce* it into an instance ϕ' for 3SAT: There is an assignment $A(\phi) = 1$ iff there is an assignment $A'(\phi') = 1$.

SAT $\leq_m^p$ 3SAT

Given $\phi = \bigwedge_{i=1}^{m}(C_i)$ and X for SAT, let z_i be a literal.
The reduction f:

- If $C_j = (z_j)$ f adds variables $Y_j = \{y_j^1, y_j^2\}$ and forms the clauses
$C_j' = (z_j \vee y_j^1 \vee y_j^2) \wedge (z_j \vee \bar{y}_j^1 \vee y_j^2) \wedge (z_j \vee y_j^1 \vee \bar{y}_j^2) \wedge (z_j \vee \bar{y}_j^1 \vee \bar{y}_j^2)$.
Notice that there is an assignment $A(C_j) = 1$ iff in the assignment A' for 3SAT, $A'(z_j = 1)$ so $A'(C_j') = 1$.
- If $C_j = (z_1 \vee z_2)$, adds variable $Y_j = \{y_j^1\}$ and forms the clauses $C_j' = (z_1 \vee z_2 \vee y_j^1) \wedge (z_1 \vee z_2 \vee \bar{y}_j^1)$.
An assignment $A'(C_j') = 1$ iff $A(C_j) = 1$, i.e. $\Rightarrow A(z_1) = 1$ or/and $A(z_2) = 1$.
- If $C_j = (z_1 \vee z_2 \vee z_3)$ then $C_j' = C_j$

- If $k > 3$, $C_j = (z_1 \vee z_2 \vee \cdots \vee z_k)$, then f adds variables $Y_j = \{y_j^1, y_j^2, \ldots, y_j^{k-3}\}$ and the clauses $C'_j = (z_1 \vee z_2 \vee y_j^1) \wedge (\bar{y}_j^1 \vee z_3 \vee y_j^2) \wedge \cdots (\bar{y}_j^{k-4} \vee z_{k-2} \vee y_j^{k-3}) \wedge (\bar{y}_j^{k-3} \vee z_{k-1} \vee z_k)$
  An assignment $A(C_j) = 1$ must have $A(z_i) = 1$ for at least a z_i then $A'(y_j^1) = \cdots = A'(y_j^{i-2}) = 1$ and
  $A'(y_j^{i-1}) = \cdots = A'(y_j^{k-3}) = 0$
  On the other hand, if $A'(C'_j) = 1 \Rightarrow \exists z_i$ s.t. $A'(z_i) = 1$ (else there would be a y_j^i: $A'(y_j^i) = 1 = A'(\bar{y}_j^i)$)

Take the input to 3SAT: $\phi' = \bigwedge_{i=1}^{m}(C'_i)$
and $Y = X \cup (\cup_{i=1}^{m} Y_i)$.

Example

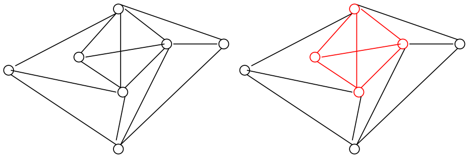
Input SAT: $\phi = (\bar{x_1})(\bar{x_1}, \bar{x_2})(\bar{x_1}, x_3, \bar{x_4})(x_1, x_2, \bar{x_3}, x_4, x_5)$
$C_1' = (\bar{x_1}, y_1, y_2)(\bar{x_1}, \bar{y}_1, y_2)(\bar{x_1}, y_1, \bar{y}_2)(\bar{x_1}, \bar{y}_1, \bar{y}_2)$
$C_2' = (\bar{x_1}, \bar{x_2}, y_3)(\bar{x_1}, \bar{x_2}, \bar{y}_3)$
$C_3' = (\bar{x_1}, x_3, \bar{x_4})$
$C_4' = (x_1, x_2, y_4)(\bar{y}_4, \bar{x}_3, y_5)(\bar{y}_5, x_4, x_5)$
Then $f(\phi) = C_1' \wedge C_2' \wedge C_3' \wedge C_4'$
with $Y = \{x_1, x_2, x_3, x_4, x_5, y_1, y_2, y_3, y_4, y_5\}$
Notice:
$A(x_1) = A(x_4) = A(x_5) = 0 \; A(x_2) = A(x_3) = 1$
$A'(x_1) = A'(x_4) = A'(x_5) = 0 \; A'(x_2) = A'(x_3) = 1$
$A'(y_1) = A'(y_2) = A'(y_3) = A'(y_4) = A'(y_5) = 0$

Complexity of f

We must prove $\phi' = f(\phi)$ can be constructed in poly($|\phi|$).
Input to SAT: The input ϕ must not have repeated clauses, or repeated literals inside the same clause ($(x_i \vee x_i \vee \bar{x}_j)$) or complementary variables inside the same clause ($(x_i \vee x_j \vee \bar{x}_j)$).
If ϕ has m clauses and n variables, $|\phi| \leq nm$.
$\forall C_i \in \phi$, $|C_i| = 1$ add 2 variables to Y and 4 clauses in ϕ'.
$\forall C_i \in \phi$, $|C_i| = 2$ add 1 variable to Y and 2 clauses in ϕ'.
$\forall C_i \in \phi$, $|C_i| = k$ add $k-3$ variables to Y and $k-2$ clauses in ϕ'.
So $|Y| = O(n)$ and $|\phi| = O(nm)$

Example: The CLIQUE

CLIQUE.
INPUT: $G(V, E)$ and $k \in \mathbb{Z}$.
QUESTION: Decide if there is a complete subgraph of G with at least k vertices.
The functional version: Given G, find the maximum complete subgraph.



3-SAT$\leq_m^p$ CLIQUE

Given an input F for 3-SAT, we have to construct in polynomial time an input G, k for CLIQUE, such that *F is sat iff G has a CLIQUE*.

Let $X = \{x_1, \dots, x_n, \bar{x_1}, \dots, \bar{x_n}\}$, and $F = C_1 \wedge \cdots \wedge C_m$.

Define $G = (V, E)$ and k:

$V = \{(x, C_j) \,|x \in C_j\}$

$E = \{((x, C_i), (y, C_j)) \,|i \neq j,\ x \neq \bar{y}, \}$

$k = m$

CLIQUE$\leq_m^p$ INDEPENDENT SET.

INDEPENDENT SET.
INPUT: $G(V, E)$ and $k \in \mathbb{Z}$.
QUESTION: Decide if there is a $S \subseteq G$ with have no edges among them and with $|S| \geq k$.

The reduction between CLIQUE and INDEPENDENT SET is very easy:
Any clique on $G = (V, E)$ is an independent set on $\bar{G} = (V, \bar{E})$ (with the same size k).
Hence, a reduction function f can be defined as
$f(\langle G, k \rangle) = \langle \bar{G}, k \rangle$.

3-SAT $\leq_m^p$ Integer Linear Programing.

Given any instance ϕ on X for 3-SAT we have to reduce it into a specific instance M, b of LP, such that there is an A with $A(\phi) = 1$ iff there is a solution $\vec{x}$ to the LP instance.
The reduction follows the fact that any clause $C_i = (x, \bar{y}, z)$ of ϕ can be expressed as a constrain $x + (1 - y) + z \geq 1$, with integers $x, y, z \in \{0, 1\}$. With a complexity of $O(nm)$.

Example: Let $X = \{x_1, x_2, x_3\}$ with

$$\phi = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3)$$

which transform into

$$\begin{aligned} x_1 + x_2 + (1 - x_3) &\geq 1 \\ (1 - x_1) + (1 - x_2) + x_3 &\geq 1 \\ (1 - x_1) + (1 - x_2) + (1 - x_3) &\geq 1 \\ 0 \leq x_1, x_2, x_3 &\leq 1. \end{aligned}$$

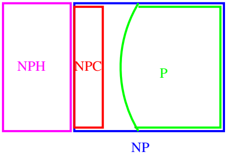
$A(x_1) = 1, A(x_2) = 0, A(x_3) = 0$ is solution to 3SAT instance iff $x_1 = 1, x_2 = 0, X_3 = 0$ is solution to ILP instance.

NP-completeness

A problem A is NP-complete if:

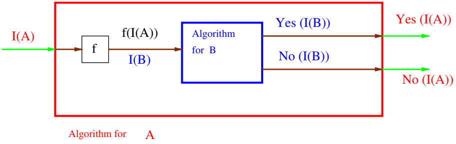
1. $A \in$ NP, and
2. for every $B \in$ NP, $B \leq_m^p A$.

If for every $B \in$ NP, $B \leq_m^p A$, then A is said to be NP-hard.



Lemma

Let A be NP-complete. Then A is in P iff P=NP.



Reduction f from problem A to problem B

So, once we prove that a problem is NP-complete, either A has no efficient algorithm or all NP problems are in P.
Moreover, if the decision version of a problem is NP-complete, then the functional version is NP-hard

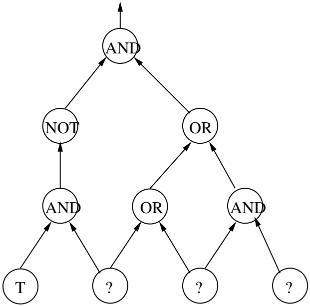
Majority conjecture: P $\neq$ NP

To prove a problem is NP-complete, we just have to find a reduction from a problem known to be NP-complete, and we need a first NP-complete problem.

We need a first problem in NP-complete: SAT or CIRCUIT-SAT

CIRCUIT SAT.

CIRCUIT-SAT.
INPUT: a boolean circuit with gates: *AND*, *OR*, *NOT*, and the input gates *T*, *F* and *?*.
QUESTION: Decide if it exists an assignment to the input gates (?), such that the circuit evaluates to T.



The seminal theorem.

Theorem (Cook's theorem)

SAT is NP-complete.

Instead of reducing any NP problem A to SAT as in the original result, we give a sketch of the proof that $A \leq_m^p$ CIRCUIT-SAT and after, prove that CIRCUIT-SAT $\leq_m^p$ SAT.

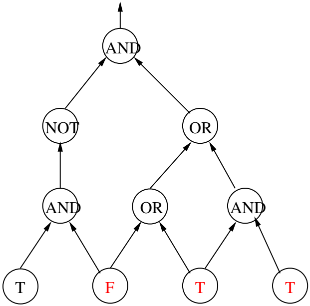
Any problem $A \in NP$ can be reduced to SAT.

Let A be a NP problem.
We want to show that $A \leq_m^P$ CIRCUIT-SAT.
Since $A \in NP$, we have that $x \in A$ iff $\exists y$, $|y| \leq p(|x|) \wedge \langle x, y \rangle \in B$ where p is a poly and $B \in P$. Let $\mathcal{A}$ be a polynomial-time algorithm that recognizes B.
Given any polynomial-time algorithm (in particular $\mathcal{A}$) its computation on any input of length can n be expressed as a polynomial-size boolean circuit C_n where its input gates encode the input to the algorithm.

Therefore, given any instance x for A, we can construct in poly-time an instance C' of CIRCUIT-SAT (whose known inputs are the bits of x and whose unknown inputs are the witness y, and such that $C'(y) = 1$ iff $C_{n+p(n)}(\langle x, y \rangle) = 1$ iff $\mathcal{A}$ on input $\langle x, y \rangle$ outputs YES.

CIRCUIT-SAT $\leq_m^p$ SAT.

CIRCUIT-SAT.
INPUT: a boolean circuit with gates: *AND*, *OR*, *NOT*, and the input gates T, F and $?$.
QUESTION: Decide if it exists an assignment to the input gates (?), such that the circuit evaluates to T.

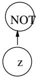


CIRCUIT SAT $\leq_m^p$ SAT

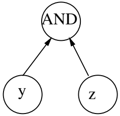
Given any circuit, we can rewrite it as a CNF: for each gate we associate a variable x and we model the effect of the gate using at most three clauses.

x

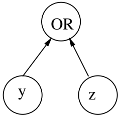
$\bar{x}$



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

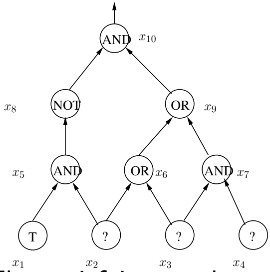


$(\bar{x} \vee y) \wedge (\bar{x} \vee z) \wedge (x \vee \bar{y} \vee \bar{z})$



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

Example.



$$
\begin{aligned}
&(\bar{x}_5 \vee x_2) \wedge (x_5 \vee \bar{x}_2) \wedge (\bar{x}_2 \vee x_6) \wedge \\
&(\bar{x}_3 \vee x_6) \wedge (\bar{x}_6 \vee x_2 \vee x_3) \wedge (\bar{x}_7 \vee x_3) \wedge \\
&(\bar{x}_7 \vee x_4) \wedge (\bar{x}_3 \vee \bar{x}_2 \vee x_7) \wedge (x_8 \vee x_5) \wedge \\
&(\bar{x}_8 \vee \bar{x}_5) \wedge (x_9 \vee \bar{x}_6) \wedge (x_9 \vee \bar{x}_7) \wedge \\
&(\bar{x}_9 \vee x_6 \vee x_7) \wedge (x_8 \vee \bar{x}_{10}) \wedge (x_9 \vee \bar{x}_{10}) \wedge \\
&(\bar{x}_9 \vee \bar{x}_8 \vee x_{10}) \wedge x_{10}
\end{aligned}
$$

The satisfying truth assignment of the resulting SAT formula is in 1-to-1 correspondence with the assignments on the gates in the given instance of CIRCUIT-SAT.

Therefore, CIRCUIT-SAT $\leq_m^p$ SAT.