Modern Complexity Theory (CS1.405)

Dr. Ashok Kumar Das

IEEE Senior Member
Web of Science (Clarivate^{TM}) Highly Cited Researcher 2022, 2023
Professor
Center for Security, Theory and Algorithmic Research
International Institute of Information Technology, Hyderabad

E-mail: *ashok.das@iiit.ac.in*
URL: `http://www.iiit.ac.in/people/faculty/ashok-kumar-das`
`https://sites.google.com/view/iitkgpakdas/`

Intractability

RELATIVIZATION

- Assume that we grant the TM the ability to solve the satisfiability problem in a single step, for any size Boolean formula.
- Imagine an attached “black box” that gives the machine this capability. We call the black box an **oracle** to emphasize that it does not necessarily correspond to any physical device.
- The machine could use the oracle to solve any NP problem in polynomial time, regardless of whether P equals NP, because every NP problem is polynomial time reducible to the satisfiability problem (SAT).
- Such a TM is said to be computing ***relative*** to the satisfiability problem (SAT); hence the term **relativization**.

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
HYDERABAD

Definition

An oracle for a language A is a device that is capable of reporting whether any string w is a member of A. An oracle Turing machine M^A is a modified Turing machine that has the additional capability of querying an oracle for A. Whenever M^A writes a string on a special oracle tape, it is informed whether that string is a member of A in a single computation step.

RELATIVIZATION

Definition (The Class P^A)

P^A be the class of languages decidable with a polynomial time oracle deterministic Turing machine that uses oracle A.
In other words, $P^A := \{L | L$ is decided by a polynomial time oracle deterministic Turing machine (ODTM) that uses oracle $A\}$.

Definition (The Class NP^A)

NP^A be the class of languages decidable with a polynomial time oracle non-deterministic Turing machine that uses oracle A.
In other words, $NP^A := \{L | L$ is decided by a polynomial time oracle non-deterministic Turing machine (ONTM) that uses oracle $A\}$.

RELATIVIZATION

Theorem

$NP \subseteq P^{SAT}$.

Theorem

$coNP \subseteq P^{SAT}$.

- Polynomial-time computation relative to the satisfiability problem (SAT) contains all of NP.
  In other words,
  $$NP \subseteq P^{SAT}$$
- Furthermore,
  $$coNP \subseteq P^{SAT}$$
  because P^{SAT}, being a deterministic complexity class, is closed under complementation.

RELATIVIZATION

Theorem

1. An oracle A exists whereby $P^A \neq NP^A$.
2. An oracle B exists whereby $P^B = NP^B$.

PROOF IDEA Exhibiting oracle B is easy. Let B be any PSPACE-complete problem such as *TQBF*.

We exhibit oracle A by construction. We design A so that a certain language L_A in NP^A provably requires brute-force search, and so L_A cannot be in P^A. Hence we can conclude that $\mathrm{P}^A \neq \mathrm{NP}^A$. The construction considers every polynomial time oracle machine in turn and ensures that each fails to decide the language L_A.

PROOF Let B be *TQBF*. We have the series of containments

$$\mathrm{NP}^{TQBF} \overset{1}{\subseteq} \mathrm{NPSPACE} \overset{2}{\subseteq} \mathrm{PSPACE} \overset{3}{\subseteq} \mathrm{P}^{TQBF}.$$

Containment 1 holds because we can convert the nondeterministic polynomial time oracle TM to a nondeterministic polynomial space machine that computes the answers to queries regarding *TQBF* instead of using the oracle. Containment 2 follows from Savitch's theorem. Containment 3 holds because *TQBF* is PSPACE-complete. Hence we conclude that $\mathrm{P}^{TQBF} = \mathrm{NP}^{TQBF}$.

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
HYDERABAD

Problem

If $NP = P^{SAT}$, then $NP = coNP$.

Solution: P^{SAT} is a deterministic class, so it is closed by complementation. We are also given that $NP = P^{SAT}$.
Note that, $A \in P$ if and only if (iff) $A \in P^{SAT}$.
Then, $A \in P^{SAT}$ iff $\bar{A} \in P^{SAT}$.
Since $NP = P^{SAT}$, $\bar{A} \in NP$ iff $A \in coNP$.
As a result, $A \in NP$ iff $A \in coNP$, that is, $NP \subseteq coNP$ and $coNP \subseteq NP$.
Hence, $NP = coNP$.

CIRCUIT COMPLEXITY

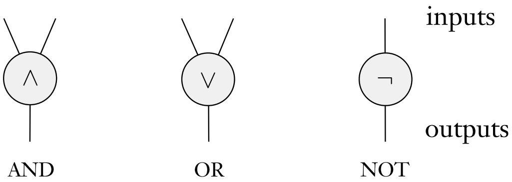
Motivation

- Computers are built from electronic devices wired together in a design called a *digital circuit*.
- We can also simulate theoretical models, such as Turing machines, with the theoretical counterpart to digital circuits, called *Boolean circuits*.
- Two purposes are served by establishing the connection between TMs and Boolean circuits.
  - First, researchers believe that circuits provide a convenient computational model for attacking the P versus NP and related questions.
  - Second, circuits provide an alternative proof of the Cook–Levin theorem that SAT is NP-complete.

CIRCUIT COMPLEXITY

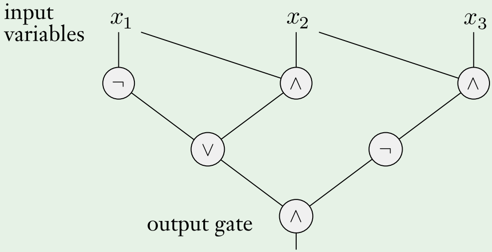
Definition (Boolean circuit)

A ***Boolean circuit*** is a collection of **gates** and **inputs** connected by wires. Cycles are not permitted. Gates take three forms: 1) AND gates, 2) OR gates, and 3) NOT gates



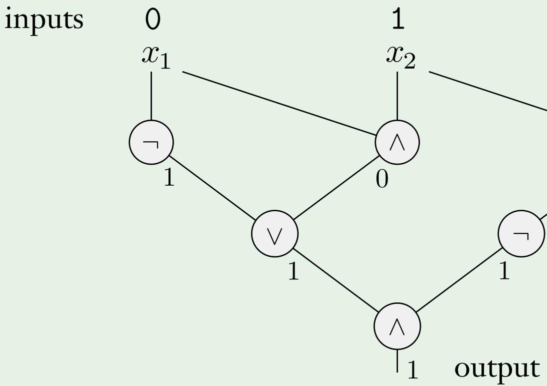
Example (A Boolean circuit)

The wires in a Boolean circuit carry the Boolean values 0 and 1. The gates are simple processors that compute the Boolean functions AND, OR, and NOT. The AND function outputs 1 if both of its inputs are 1 and outputs 0 otherwise. The OR function outputs 0 if both of its inputs are 0 and outputs 1 otherwise. The NOT function outputs the opposite of its input; in other words, it outputs a 1 if its input is 0 and a 0 if its input is 1. The inputs are labeled $x_1, \ldots, x_n$. One of the gates is designated the ***output gate***. The following figure depicts a Boolean circuit.



CIRCUIT COMPLEXITY

Example (A Boolean circuit computing)



CIRCUIT COMPLEXITY

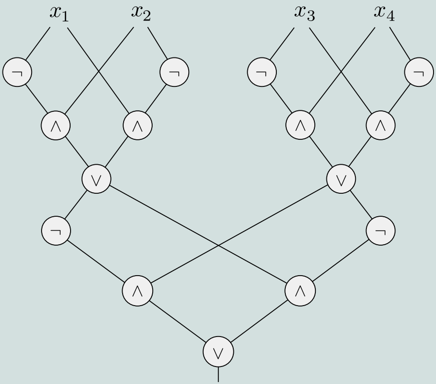
Boolean circuit computing

We use functions to describe the input/output behavior of Boolean circuits. To a Boolean circuit C with n input variables, we associate a function $f_C \colon \{0,1\}^n \longrightarrow \{0,1\}$, where if C outputs b when its inputs $x_1, \ldots, x_n$ are set to $a_1, \ldots, a_n$, we write $f_C(a_1, \ldots, a_n) = b$. We say that C computes the function f_C. We sometimes consider Boolean circuits that have multiple output gates. A function with k output bits computes a function whose range is $\{0,1\}^k$.

Boolean circuit computing

Example (A Boolean circuit that computes the parity function)

The n-input ***parity function*** $parity_n : \{0,1\}^n \to \{0,1\}$ outputs 1 if an odd number of 1s appear in the input variables.



CIRCUIT COMPLEXITY

Definition (Circuit family)

A ***circuit family*** C is an infinite list of circuits, say $(C_0, C_1, C_2, \ldots)$, where C_n has n input variables. Then, C decides a language A over the alphabet $\{0, 1\}$ if for every string w,

$$w \in A \text{ iff } C_n(w) = 1,$$

where n is the length of w.

- The ***size*** of a circuit is the number of gates that it contains.
- Two circuits are ***equivalent*** if they have the same input variables and output the same value on every input assignment.
- A circuit is ***size minimal*** if no smaller circuit is equivalent to it.
- The problem of minimizing circuits has obvious engineering applications but is very difficult to solve in general.

CIRCUIT COMPLEXITY

- A circuit family is minimal if every C_i on the list is a minimal circuit.
- The ***size complexity*** of a circuit family $(C_0, C_1, C_2, \ldots)$ is the function $f : N \rightarrow N$, where $f(n)$ is the size of C_n.
- The ***depth*** of a circuit is the length (number of wires) of the longest path from an input variable to the output gate.
- We define ***depth minimal*** circuits and circuit families, and the depth complexity of circuit families, as we did with circuit size.

CIRCUIT COMPLEXITY

Definition

- The ***circuit complexity*** of a language is the size complexity of a minimal circuit family for that language.
- The ***circuit depth complexity*** of a language is defined similarly, using depth instead of size.

Theorem

Let $t : N \rightarrow N$ be a function, where $t(n) \geq n$. If a language $A \in TIME(t(n))$, then A has circuit complexity $O(t^2(n))$.

CIRCUIT COMPLEXITY

We say that a Boolean circuit is ***satisfiable*** if some setting of the input causes the circuit to output 1.
The ***circuit-satisfiable*** problem tests whether a circuit is satisfiable.
Define

$$\text{CIRCUIT-SAT} := \{\langle C\rangle | C \text{ is a satisfiable Boolean circuit}\}.$$

Theorem

CIRCUIT-SAT is NP-complete.

Proof.

We must demonstrate the following two things:

1. CIRCUIT-SAT is in NP.
2. (NP-hard) any language A in NP is reducible to CIRCUIT-SAT.

□

Part 1. CIRCUIT-SAT is in NP.

We design the following polynomial-time verifier (DTM, V) that can decide CIRCUIT-SAT.

Algorithm: Polynomial-time verifier (DTM, V) for CIRCUIT-SAT

Input: $(\langle C \rangle, \beta)$, where β is a certificate (an assignment of the setting of the n Boolean variables, say $x_1, x_2, \cdots, x_n$)

Output: Accept/Reject

```
if β does not contain n Boolean variables assignment then
    return "reject"
end if
Evaluate the circuit C on β.
if the circuit output is 1 then
    return "accept"
else
    return "reject"
end if
```

Part 2. CIRCUIT-SAT is NP-hard.

We show that any language A in NP is poly-time reducible to CIRCUIT-SAT. We must give a polynomial time reduction $f : \Sigma^* \to \Sigma^*$ that maps strings to circuits, where $f(w) = \langle C \rangle$ implies that $w \in A$ iff Boolean circuit C is satisfiable.

Because A is in NP, it has a polynomial time verifier (DTM) V whose input has the form $\langle x, c \rangle$ where c may be the certificate showing that the assignment x is in A. In order to construct f, we need to obtain the circuit simulating V.

We now fill in the inputs to the circuit that correspond to x with the symbols of w. The only remaining inputs to the circuit correspond to the certificate c. We call this circuit C and output it.

If C is satisfiable, a certificate exists, so w is in A. Conversely, if w is in A, a certificate exists, so C is satisfiable.

Time Complexity: If the running time of the polynomial-time verifier V is n^k for some positive integer k, so the size of the circuit constructed is $O((n^k)^2) = O(n^{2k})$ by the circuit complexity theorem. Since the structure of the circuit is quite simple (actually, it is highly repetitious), so the running time of the reduction is $O(n^{2k})$, which is poly-time.

CIRCUIT COMPLEXITY

Definition (P-complete)

A language B is P-complete if

1. $B \in P$, and
2. every A in P is log space reducible to B, that is,

$$A \leq_L B, \forall A \in P.$$

[P-hard]

CIRCUIT COMPLEXITY

P-completeness

For a circuit C and input setting x, we represent $C(x)$ to be the value of C on x. Define

$$\text{CIRCUIT-VALUE} := \{\langle C, x\rangle \mid C \text{ is a Boolean circuit and } C(x) = 1\}.$$

Theorem

CIRCUIT-VALUE is P-complete.

Proof.

Part 1. CIRCUIT-VALUE is in P. We need to design a DTM, say, M that can decide CIRCUIT-VALUE in poly-time.
Part 2. Any A in P is log space reducible to CIRCUIT-VALUE, that is,

$$A \leq_L \text{CIRCUIT-VALUE}, \forall A \in P.$$

$\square$

CIRCUIT COMPLEXITY

Part 1.

Algorithm: Polynomial-time DTM, M for CIRCUIT-VALUE

Input: $\langle C, x \rangle$, where C is a Boolean circuit and x an assignment of the setting of the n Boolean variables, say $x_1, x_2, \cdots, x_n$)

Output: Accept/Reject

```
Evaluate the circuit C on x.
if the circuit output is 1 then
    return "accept"
else
    return "reject"
end if
```

Part 2. Required to Prove (RTP):

$$A \leq_L \text{ CIRCUIT-VALUE}, \forall A \in P$$

Let A be any language in P. We convert an input string $w \in A$ to a Boolean circuit C with w such that $w \in A$ if and only if $f(w) = \langle C \rangle$ is satisfiable Boolean circuit on w, where $f : \Sigma^* \rightarrow \Sigma^*$ is log-space reduction function.

On input w, the reduction produces a circuit C that simulates the polynomial time deterministic Turing machine (DTM) for A. The input to the circuit C is w itself. The reduction can be carried out in log space because the circuit C it produces has a simple and repetitive structure.

Thank You!!!