15-251
Great Ideas in Theoretical Computer Science
Boolean Circuits
\[ P \equiv NP \]
What is P?

P

The set of languages that can be decided in $O(n^k)$ steps for some constant $k$.

The theoretical divide between efficient and inefficient:

$L \in P \quad \rightarrow \quad$ efficiently solvable.

$L \notin P \quad \rightarrow \quad$ not efficiently solvable.
Why P?

- Poly-time is **not** meant to mean “efficient in practice”

- Poly-time: mathematical insight into problem’s structure

- Poly-time: extraordinarily better than brute force search

- Robust to notion of what is an *elementary step*, *what model we use*, *reasonable encoding of input*, *implementation details*.

- Nice closure property:
  Plug in a poly-time alg. into another poly-time alg. $\implies$ poly-time
Why P?

Brute-Force Algorithm: Exponential time

what we care about most in 15-251

Algorithmic Breakthrough: Polynomial time

what we care about more in 15-451

Blood, sweat, and tears: Linear time (or small exponent)
Summary: Poly-time vs not poly-time is a *qualitative* difference, not a *quantitative* one
What is NP?

The set of languages whose solution can be verified in polynomial time

Factoring: Given $N = pq$ where $p$ and $q$ are large primes, output $p,q$

• Considered hard to solve in polynomial time
• Solution easy to verify

Why such strange definition?

• Class of problems we would like to solve in polynomial time
• Problem outside P?
Problems outside P?

Should we care?
Problem beyond NP

Can solve problems beyond NP. Only $100 per problem.

$$$, Problem

#432o~#*&#@)@@!
What is the P vs NP question?

 asks whether these two sets are equal

How would you show $P = NP$?

> Show that every problem in $NP$ can be solved in poly-time.

How would you show $P \neq NP$?

> Show that there is a problem in $NP$ which cannot be solved in poly-time.
P vs NP is on the horizon

Millennium Problems

Yang–Mills and Mass Gap
Experiment and computer simulations suggest the existence of a "mass gap" in the solution to the quantum versions of the Yang–Mills equations. But no proof of this property is known.

Riemann Hypothesis
The prime number theorem determines the average distribution of the primes. The Riemann hypothesis tells us about the deviation from the average. Formulated in Riemann’s 1859 paper, it asserts that all the 'non-obvious' zeros of the zeta function are complex numbers with real part 1/2.

P vs NP Problem
If it is easy to check that a solution to a problem is correct, is it also easy to solve the problem? This is the essence of the P vs NP question. Typical of the NP problems is that of the Hamiltonian Path Problem: given N cities to visit, how can one do this without visiting a city twice? If you give me a solution, I can easily check that it is correct. But I cannot so easily find a solution.

Navier–Stokes Equation
This is the equation which governs the flow of fluids such as water and air. However, there is no proof for the most basic questions one can ask: do solutions exist, and are they unique? Why ask for a proof? Because a proof gives not only certainty, but also understanding.

Hodge Conjecture
The answer to this conjecture determines how much of the topology of the solution set of a system of algebraic equations can be defined in terms of further algebraic equations. The Hodge conjecture is known in certain special cases, e.g., when the solution set has dimension less than four. But in dimension four it is unknown.

Poincaré Conjecture
In 1904 the French mathematician Henri Poincaré asked if the three dimensional sphere is characterized as the unique simply connected three manifold. This question, the Poincaré conjecture, was a special case of Thurston's geometrization conjecture. Perelman's proof tells us that every three manifold is built from a set of standard pieces, each with one of eight well-understood geometries.

Birch and Swinnerton-Dyer Conjecture
Supported by much experimental evidence, this conjecture relates the number of points on an elliptic curve mod p to the rank of the group of rational points. Elliptic curves, defined by cubic equations in two variables, are fundamental mathematical objects that arise in many areas: Wiles' proof of the Fermat Conjecture, factorization of numbers into primes, and cryptography, to name three.
Boolean circuits are related to the P vs NP question in multiple ways
Boolean Circuits
Some preliminary questions

What is a Boolean circuit?
- It is a computational model for computing decision problems (or computational problems)

We already have TMs. Why Boolean circuits?
- The definition is simpler
- Easier to understand, usually easier to reason about
- Boolean circuits can efficiently simulate TMs
  (efficient decider TM $\implies$ efficient/small circuits.)
- Circuits are good models to study parallel computation
- Real computers are built with digital circuits
Sounds AWESOME!
So why didn’t we just learn about circuits first?

An inspirational quote from a famous person:

An algorithm is a **finite** answer to **infinite** number of questions.

Stephen Kleene
(1909 - 1994)
Sounds AWESOME!
So why didn’t we just learn about circuits first?

A non-inspirational quote from a non-famous person:

Circuits are an infinite answer to infinite number of questions.
<table>
<thead>
<tr>
<th><strong>Dividing a problem according to length of input</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Sigma = {0, 1} )</td>
</tr>
<tr>
<td>( L \subseteq {0, 1}^* )</td>
</tr>
<tr>
<td>( f : {0, 1}^* \rightarrow {0, 1} )</td>
</tr>
<tr>
<td>( {0, 1}^n = \text{all strings of length } n )</td>
</tr>
<tr>
<td>( L_n = {w \in L :</td>
</tr>
<tr>
<td>( f^n : {0, 1}^n \rightarrow {0, 1} )</td>
</tr>
<tr>
<td>for ( x \in {0, 1}^n ), ( f^n(x) = f(x) )</td>
</tr>
<tr>
<td>( L = L_0 \cup L_1 \cup L_2 \cup \cdots )</td>
</tr>
<tr>
<td>( f = (f^0, f^1, f^2, \ldots) )</td>
</tr>
</tbody>
</table>
Dividing a problem according to length of input

A TM is a finite object (finite number of states) but can handle any input length.

Imagine a model where we allow the TM to change with input length.

\[ \text{input} \rightarrow \text{TM} \rightarrow \text{output} \]

computes \( L \)

\[ \text{TM}_0 \quad \text{TM}_1 \quad \text{TM}_2 \quad \text{TM}_3 \quad \ldots \]

\[ L_0 \quad L_1 \quad L_2 \quad L_3 \quad \ldots \]
Dividing a problem according to length of input

So one machine does not compute $L$.

You use a family of machines:

$$(M_0, M_1, M_2, \ldots)$$

(Imagine having a different Python function for each input length.)

Is this a reasonable/realistic model of computation?!?

Boolean circuits work this way

Need a separate circuit for each input length
Boolean Circuit Definition
Picture of a circuit
Picture of a circuit
Picture of a circuit

- Binary OR gate
- Binary AND gate
- Unary NOT gate
- Input gate
- Output gate
Picture of a circuit

- Binary OR gate
- Binary AND gate
- Unary NOT gate
- Input gate
- Output gate
Information flows from input gates to the output gate. No feedback loops allowed! It is a directed acyclic graph.
Computes a function $f : \{0, 1\}^n \rightarrow \{0, 1\}$
So how does it compute $f(x_1, x_2, \ldots, x_n)$?
Computes a function $f : \{0, 1\}^n \rightarrow \{0, 1\}$
So how does it compute $f(x_1, x_2, \ldots, x_n)$?
Computes a function $f : \{0, 1\}^n \rightarrow \{0, 1\}$

So how does it compute $f(x_1, x_2, \ldots, x_n)$?
Computes a function $f : \{0, 1\}^n \rightarrow \{0, 1\}$
So how does it compute $f(x_1, x_2, \ldots, x_n)$?
Computes a function \( f : \{0, 1\}^n \rightarrow \{0, 1\} \)
So how does it compute \( f(x_1, x_2, \ldots, x_n) \)?
Computes a function $f : \{0, 1\}^n \rightarrow \{0, 1\}$
So how does it compute $f(x_1, x_2, \ldots, x_n)$?
Picture of a circuit

Computes a function \( f : \{0, 1\}^n \rightarrow \{0, 1\} \)

So how does it compute \( f(x_1, x_2, \ldots, x_n) \)?
Computes a function $f : \{0, 1\}^n \rightarrow \{0, 1\}$

So how does it compute $f(x_1, x_2, \ldots, x_n)$?
Picture of a circuit

Computes a function \( f : \{0, 1\}^n \rightarrow \{0, 1\} \)
So how does it compute \( f(x_1, x_2, \ldots, x_n) \)?
Computes a function $f : \{0, 1\}^n \rightarrow \{0, 1\}$

So how does it compute $f(x_1, x_2, \ldots, x_n)$?
Truth Table of a Circuit

• A table of (input, output) pairs

<table>
<thead>
<tr>
<th>Input 1</th>
<th>Input 2</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Truth table of AND gate

• If n bit input, number of possible inputs is $2^n$

• Hence, rows in the table is $2^n$
Question: What does this circuit compute?

(sometimes circuits are drawn upside down)
Question: What does this circuit compute?

(sometimes circuits are drawn upside down)

parity of $x_1 + x_2$

$x_1 \oplus x_2$

parity of $x_3 + x_4$

$x_3 \oplus x_4$

$x_1 \oplus x_2 \oplus x_3 \oplus x_4$
How does a circuit **decide/compute** a language?

How do we measure the **complexity** of a circuit?
How can a circuit decide a language?

Given \( f : \{0, 1\}^* \rightarrow \{0, 1\} \), write

\[
f = (f^0, f^1, f^2, \ldots)
\]

where \( f^n : \{0, 1\}^n \rightarrow \{0, 1\} \)

Construct a circuit for each input length.

\[
\begin{align*}
C_0 & \quad f^0 \\
C_1 & \quad f^1 \\
C_2 & \quad f^2 \\
C_3 & \quad f^3 \\
\ldots
\end{align*}
\]

A circuit family \( C \) is a collection of circuits \((C_0, C_1, C_2, \ldots)\) where each \( C_n \) has \( n \) input gates.
How can a circuit decide a language?

We say that a circuit family \( C = (C_0, C_1, C_2, \ldots) \) decides/computes \( f : \{0, 1\}^* \to \{0, 1\} \) if \( C_n \) computes \( f^n \) for every \( n \).
Circuit size and complexity

Definition (size of a circuit): The size of a circuit is the total number of gates (counting the input variables as gates too) in the circuit.

Definition (size of a circuit family): The size of a circuit family \( C = (C_0, C_1, C_2, \ldots) \) is a function \( s : \mathbb{N} \to \mathbb{N} \) such that \( s(n) = \text{size of } C_n \).

Definition (circuit complexity): The circuit complexity of a decision problem is the size of the minimal circuit family that decides it.
<table>
<thead>
<tr>
<th>Circuits vs TMs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>size of a circuit family</strong></td>
</tr>
<tr>
<td>$s(n)$</td>
</tr>
<tr>
<td><strong>circuit complexity</strong></td>
</tr>
<tr>
<td>of a decision problem</td>
</tr>
</tbody>
</table>
Intrinsic complexity

Time complexity

algs. with complexity worse than $\Theta(n^2)$.

best alg. that solves $L$ $\Theta(n^2)$

algs. with complexity better than $\Theta(n^2)$

nothing here solves $L$.

some alg. here solve $L$. 
Let \( f : \{0, 1\}^* \rightarrow \{0, 1\} \) be the parity decision problem.

\[
f(x) = x_1 + \ldots + x_n \mod 2 \quad \text{(where } n = |x|)\]

\[
f(x) = x_1 \oplus \cdots \oplus x_n\]

What is the circuit complexity of this function?
Poll

\[ s(n) = 2s(n/2) + 5 \]
\[ s(1) = 1 \]
\[ \implies s(n) = O(n). \]
Fundamental Facts Regarding Circuits (Big picture)
The big picture

Computability with respect to circuits

**Theorem 1:** Any decision problem $f : \{0, 1\}^* \rightarrow \{0, 1\}$ can be computed by a circuit family of size $O(2^n)$.

A universal exponential upper bound for all decision problems

(We know this is not true in the TM model)
The big picture

Limits of efficient computability with respect to circuits

**Theorem 2 (Shannon’s Theorem):**
There is a decision problem such that any circuit family computing it must have size at least $\frac{2^n}{5n}$

Less exciting than it seems: consider a “random” decision problem. You can’t even write it down.

In fact, **almost all** (!!!) decision problems require exponential size circuits.
The big picture

Circuits can efficiently “simulate” TMs

**Theorem 3:** Let $f : \{0, 1\}^* \rightarrow \{0, 1\}$ be a decision problem which can be decided in time $O(T(n))$. Then it can be computed by a circuit family of size $O(T(n)^2)$.

\[
\text{poly-time TM} \quad \iff \quad \text{poly-size circuits}
\]
Consequence of Theorem 3

poly-time TM $\iff$ poly-size circuits
no poly-size circuits $\implies$ no poly-time TM

To show $P \neq NP$:
Find $h$ in NP whose circuit complexity is “super-polynomial”.

\[ h \in NP \cap \neg P \]
Consequence of Theorem 3

So we can just work with circuits instead

This is awesome in 2 ways:

1. **Circuits:** clean and simple definition of computation
   “Just” a composition of **AND**, **OR**, **NOT** gates

2. Restrict the circuit
   Make it less powerful
   e.g. (i) restrict *depth*
   (ii) restrict types of gates
Circuit lower bounds

Shannon: practically all decision problems require exponential size circuits.

**Question:** find one with a small (poly-size) description (non cheating version of Shannon’s theorem)?

1976: Schnorr gives $3n-3$ lower bound.


2001: Lachish-Raz give $4.5n$ - lower bound.

2002: Iwama-Morizumi give $5n$ - lower bound.

**Great News!**

2015: Demenkov et al. give a simpler proof of $5n$ - lower bound.
Circuit lower bounds

Exciting progress was made in the 1980s for restricted circuits.

People thought $P \neq NP$ would be proved soon.

After 60 years of research and lots of funding, best lower bound on circuit size for an explicit function ("small description"): 

$$5n - \text{peanuts}$$
The big picture

Theorem 1:
Any decision problem can be computed by an exponential size circuit family.

Theorem 2:
Almost all decision problems require exponential size circuit families.

Theorem 3:
poly-time TM $\implies$ poly-size circuits
no poly-size circuits $\implies$ no poly-time TM
Proof of Theorem 2
(Informal) Poll

How many different functions $f : \{0, 1\}^n \to \{0, 1\}$ are there?

- $n$
- $2n$
- $n^2$
- $2^n$
- $2^{2^n}$
- none of the above
- beats me
Theorem 2: Some functions are hard

**Theorem:** There exists a decision problem such that any circuit family computing it must have size at least $2^n/5n$.

**Proof:**

Want to show: there is a function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ that cannot be computed by a circuit of size $< 2^n/5n$.

**Observation:** # possible functions is $2^{2^n}$

**Claim 1:** # circuits of size at most $s$ is $\leq 2^{5s \log s}$

**Claim 2:** For $s \leq 2^n/5n$, $2^{5s \log s} < 2^{2^n}$

# circuits $< \#$ functions
Theorem 2: Some functions are hard

**Theorem:** There exists a decision problem such that any circuit family computing it must have size at least $2^n/5n$.

**Proof:**

Want to show: there is a function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ that cannot be computed by a circuit of size $< 2^n/5n$.

**Observation:** # possible functions is $2^{2^n}$

**Claim 1:** # circuits of size at most $s$ is $\leq 2^{5s \log s}$

**Claim 2:** For $s \leq 2^n/5n$, $2^{5s \log s} < 2^{2^n}$

We are done once we prove Claim 1. (Claim 2 is easy/uninteresting.)
Theorem 2: Some functions are hard

Proof (continued):

Claim 1: \# circuits of size at most \( s \) is \( \leq 2^{5s \log s} \)

Proof of claim:

 Encoding a circuit with a binary string of length \( 5s \log s \):

Number the gates: 1, 2, 3, 4, \ldots, \( s \)

For each gate in the circuit, write down:
- type of the gate (3 bits)
- from which gates the inputs are coming from (2 \( \log s \) bits)

Total: \( s(3 + 2 \log s) \) bits

(3\( s \) + 2\( s \log s \)) bits \( \leq (5s \log s) \) bits
Theorem 2: Some functions are hard

That was due to Claude Shannon (1949).

Father of *Information Theory*.

A non-constructive argument.

In fact, it is easy to show that **most** functions require exponential size circuits.
Back to P vs NP

Boolean circuits: another model of computation. (arguably simpler definition, easier to reason about)

\[ \text{no poly-size circuits} \implies \text{no poly-time TM} \]
(can attack P vs NP problem with circuits)

CIRCUIT-SAT decision problem:

Given as input the description of a circuit, output True if the circuit is “satisfiable”.

Whether CIRCUIT-SAT is in P or not is intimately related to the P vs NP question.
Theorem 1: Max circuit size of a function

**Theorem:** Any decision problem \( f : \{0, 1\}^* \rightarrow \{0, 1\} \) can be computed by a circuit family of size \( O(2^n) \).

**Proof:**

**Goal:** Construct a circuit of size \( O(2^n) \) for \( f^n : \{0, 1\}^n \rightarrow \{0, 1\} \).

**Observation:**

\[
 f^n(x_1, x_2, \ldots, x_n) = (x_1 \land f^n(1, x_2, \ldots, x_n)) \lor \\
(\neg x_1 \land f^n(0, x_2, \ldots, x_n))
\]
**Theorem 1: Max circuit size of a function**

**Theorem:** Any decision problem $f : \{0, 1\}^* \rightarrow \{0, 1\}$ can be computed by a circuit family of size $O(2^n)$.

**Proof:**

**Goal:** Construct a circuit of size $O(2^n)$ for $f^n : \{0, 1\}^n \rightarrow \{0, 1\}$.

**Observation:**

$$f^n(x_1, x_2, \ldots, x_n) = \begin{cases} 1 & \text{if } x_1 = 1 \\ x_1 \land f^n(1, x_2, \ldots, x_n) \lor \neg x_1 \land f^n(0, x_2, \ldots, x_n) & \text{if } x_1 = 0 \end{cases}$$
Theorem 1: Max circuit size of a function

**Theorem:** Any decision problem \( f : \{0, 1\}^* \to \{0, 1\} \) can be computed by a circuit family of size \( O(2^n) \).

**Proof:**

**Goal:**
Construct a circuit of size \( O(2^n) \) for \( f^n : \{0, 1\}^n \to \{0, 1\} \)

**Observation:**
\[
f^n(x_1, x_2, \ldots, x_n) = (x_1 \land f^n(1, x_2, \ldots, x_n)) \lor (\neg x_1 \land f^n(0, x_2, \ldots, x_n))
\]

if \( x_1 = 0 \)
Theorem 1: Max circuit size of a function

Proof (continued):

\[ s(n) = \text{max size of a circuit computing } n\text{-variable function} \]

\[ s(n) \leq 2s(n-1) + 5, \quad s(1) \leq 3 \implies s(n) = O(2^n) \]
Theorem: Let \( f : \{0, 1\}^* \rightarrow \{0, 1\} \) be a decision problem which can be decided in time \( O(T(n)) \). Then it can be computed by a circuit family of size \( O(T(n)^2) \).

How can you prove such a theorem?

If you like a challenge, try to prove it yourself.

If you don’t like a challenge, but still curious, see the course notes for a sketch of the proof.
$P \equiv NP$