Computational Complexity Theory

Lecture 12: Boolean Circuits: class P/poly; Karp-Lipton theorem

Department of Computer Science,
Indian Institute of Science

Boolean Circuits

An algorithm for every input length?

- *"One might imagine that $P \neq NP$, but SAT is tractable in the following sense: for every ℓ there is a very short program that runs in time ℓ^2 and correctly treats all instances of size ℓ."* --- Karp and Lipton (1982).

An algorithm for every input length?

- *"One might imagine that P ≠ NP, but SAT is tractable in the following sense: for every ℓ there is a very short program that runs in time ℓ^2 and correctly treats all instances of size ℓ."* --- Karp and Lipton (1982).

- P ≠ NP rules out the existence of a single efficient algorithm for SAT that handles all input lengths. But, it doesn't rule out the possibility of having a sequence of efficient SAT algorithms – one for each input length.

Lesson learnt from Cook-Levin

- Locality of computation implies that an algorithm A working on inputs of some fixed length n and running in time $T(n)$ can be viewed as a Boolean circuit φ of size $O(T(n)^2)$ s.t. $A(x) = \varphi(x)$ for every $x \in \{0,1\}^n$.
- On the other hand, a circuit on inputs of length n and of size S can be viewed as an algorithm working on length n inputs and running in time S.

Lesson learnt from Cook-Levin

- Locality of computation implies that an algorithm A working on inputs of some fixed length n and running in time $T(n)$ can be viewed as a Boolean circuit φ of size $O(T(n)^2)$ s.t. $A(x) = \varphi(x)$ for every $x \in \{0,1\}^n$.
- On the other hand, a circuit on inputs of length n and of size S can be viewed as an algorithm working on length n inputs and running in time S.

- To rule the existence of a sequence of algorithms – one for each input length – we need to rule out the existence of a sequence of (i.e., a family of) circuits.

Boolean circuits

- A Boolean circuit is a directed acyclic graph whose nodes/gates are labelled as follows:

➢ A node with in-degree zero is labelled by an input variable, and it outputs the value of the variable.

➢ Any other node is labelled by one of the three operations $\wedge$, $\vee$, $\neg$, and it outputs the value of the operation on its input.

Nodes with out-degree zero are the output gates.

Boolean circuits

- A <u>Boolean circuit</u> is a directed acyclic graph whose nodes/gates are labelled as follows:
  - ➢ A node with in-degree zero is labelled by an input variable, and it outputs the value of the variable.
  - ➢ Any other node is labelled by one of the three operations $\wedge$, $\vee$, $\neg$, and it outputs the value of the operation on its input.

  Nodes with out-degree zero are the output gates.

- Typically, we'll consider circuits with one output gate, and with nodes having in-degree at most two.

Boolean circuits

- A <u>Boolean circuit</u> is a directed acyclic graph whose nodes/gates are labelled as follows:
  - A node with in-degree zero is labelled by an input variable, and it outputs the value of the variable.
  - Any other node is labelled by one of the three operations $\wedge$, $\vee$, $\neg$, and it outputs the value of the operation on its input.

  Nodes with out-degree zero are the output gates.

- **<u>Size</u>** of circuit is the no. of edges in it. **<u>Depth</u>** is the length of the longest path from an i/p to o/p node.

Boolean circuits

- A <u>Boolean circuit</u> is a directed acyclic graph whose nodes/gates are labelled as follows:
  - A node with in-degree zero is labelled by an input variable, and it outputs the value of the variable.
  - Any other node is labelled by one of the three operations $\wedge$, $\vee$, $\neg$, and it outputs the value of the operation on its input.

  Nodes with out-degree zero are the output gates.

θ(no. of nodes)

- **<u>Size</u>** of circuit is the <u>no. of edges</u> in it. **<u>Depth</u>** is the length of the longest path from an i/p to o/p node.

Boolean circuits

- A <u>Boolean circuit</u> is a directed acyclic graph whose nodes/gates are labelled as follows:
  - A node with in-degree zero is labelled by an input variable, and it outputs the value of the variable.
  - Any other node is labelled by one of the three operations $\wedge$, $\vee$, $\neg$, and it outputs the value of the operation on its input.

  Nodes with out-degree zero are the output gates.

- **<u>Size</u>** corresponds to "sequential time complexity". **<u>Depth</u>** corresponds to "parallel time complexity".

Boolean circuits

- A <u>Boolean circuit</u> is a directed acyclic graph whose nodes/gates are labelled as follows:
  - A node with in-degree zero is labelled by an input variable, and it outputs the value of the variable.
  - Any other node is labelled by one of the three operations $\wedge$, $\vee$, $\neg$, and it outputs the value of the operation on its input.

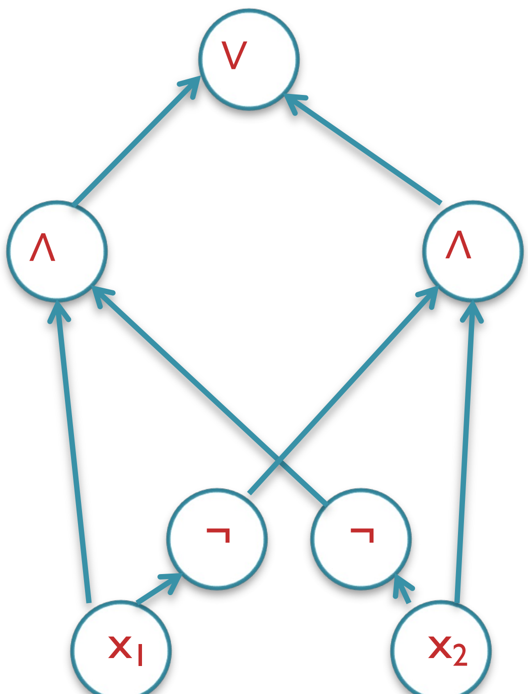
  Nodes with out-degree zero are the output gates.

- If every node in a circuit has out-degree at most one, then the circuit is called a <u>**formula**</u>.

A circuit for Parity

- $\text{PARITY}(x_1, x_2, \ldots, x_n) = x_1 \oplus x_2 \oplus \ldots \oplus x_n$.

$$x_1 \oplus x_2 = (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$$



φ

Size(φ) = |φ| = 8
Depth(φ) = 3

Circuit family

- Let $T: \mathbb{N} \to \mathbb{N}$ be some function.
- Definition: A $T(n)$-size circuit family is a set of circuits $\{C_n\}_{n \in \mathbb{N}}$ such that C_n has n inputs and $|C_n| \leq T(n)$.

Class P/poly

- Let $T: \mathbb{N} \rightarrow \mathbb{N}$ be some function.
- Definition: A $T(n)$-size circuit family is a set of circuits $\{C_n\}_{n\in\mathbb{N}}$ such that C_n has n inputs and $|C_n| \leq T(n)$.
- Definition: A language L is in $SIZE(T(n))$ if there's a $T(n)$-size circuit family $\{C_n\}_{n\in\mathbb{N}}$ such that

$$x \in L \Leftrightarrow C_n(x) = 1, \text{ where } n = |x|.$$

- Defintion: Class $P/poly = \bigcup_{c \geq 1} SIZE(n^c)$.

Class P/poly

- Let $T: \mathbb{N} \rightarrow \mathbb{N}$ be some function.
- Definition: A $T(n)$-size circuit family is a set of circuits $\{C_n\}_{n\in\mathbb{N}}$ such that C_n has n inputs and $|C_n| \leq T(n)$.

- Definition: A language L is in $SIZE(T(n))$ if there's a $T(n)$-size circuit family $\{C_n\}_{n\in\mathbb{N}}$ such that

$$x \in L \iff C_n(x) = 1, \text{ where } n = |x|.$$

The circuit family $\{C_n\}_{n\in\mathbb{N}}$ *decides* L, i.e., C_n *decides* $L \cap \{0,1\}^n$.

- Defintion: Class $P/poly = \bigcup_{c \geq 1} SIZE(n^c)$.

Class P/poly

- Let $T: \mathbb{N} \rightarrow \mathbb{N}$ be some function.
- Definition: A $T(n)$-size circuit family is a set of circuits $\{C_n\}_{n \in \mathbb{N}}$ such that C_n has n inputs and $|C_n| \leq T(n)$.

- Definition: A language L is in $SIZE(T(n))$ if there's a $T(n)$-size circuit family $\{C_n\}_{n \in \mathbb{N}}$ such that

$$x \in L \Leftrightarrow C_n(x) = 1, \text{ where } n = |x|.$$

Alternatively, we say C_n *computes* the characteristic function of $L \cap \{0,1\}^n$.

- Defintion: Class $P/poly = \bigcup_{c \geq 1} SIZE(n^c)$.

Class P/poly

- Observation: $P \subseteq P/poly$.
- Proof. If $L \in P$, then there's a n^c-time TM that decides L for some constant c. By Cook-Levin, there's a $O(n^{2c})$-size circuit family $\{C_n\}_{n\in N}$ such that

$$x \in L \iff C_n(x) = 1, \text{ where } n = |x|.$$

Class P/poly

- Observation: $P \subseteq P/poly$.
- Proof. If $L \in P$, then there's a n^c-time TM that decides L for some constant c. By Cook-Levin, there's a $O(n^{2c})$-size circuit family $\{C_n\}_{n\in N}$ such that

  $$x \in L \iff C_n(x) = 1, \text{ where } n = |x|.$$

  (Note: C_n is poly(n)-time computable from 1^n.)

- Is $P = P/poly$?

Class P/poly

- Observation: $P \subseteq P/poly$.
- Proof. If $L \in P$, then there's a n^c-time TM that decides L for some constant c. By Cook-Levin, there's a $O(n^{2c})$-size circuit family $\{C_n\}_{n\in N}$ such that

  $x \in L \iff C_n(x) = 1$, where $n = |x|$.

  (Note: C_n is poly(n)-time computable from 1^n.)

- Is $P = P/poly$? No! P/poly contains undecidable languages.

Class P/poly

- Let HALT = {(M,y) : M halts on input y}. HALT is an undecidable language.
- Notation. #(M,y) = number corresponding to the binary string (M,y).
- Let UHALT = $\{1^{\#(M,y)} : (M,y) \in \text{HALT}\}$. Then, UHALT is also an undecidable language.

Class P/poly

- Let HALT = {(M,y) : M halts on input y}. HALT is an undecidable language.
- Notation. #(M,y) = number corresponding to the binary string (M,y).
- Let UHALT = $\{1^{\#(M,y)} : (M,y) \in \text{HALT}\}$. Then, UHALT is also an undecidable language.

- Obs. Any unary language is in P/poly. (*Homework*)

  Hence, P $\subsetneq$ P/poly .

Class P/poly

- What makes P/poly contain undecidable languages? *Ans:* $L \in$ P/poly implies that L is decided by a circuit family $\{C_n\}$, where $|C_n| = n^{O(1)}$. We don't require that C_n is poly-time computable from 1^n.

Class P/poly

- What makes P/poly contain undecidable languages? *Ans:* $L \in$ P/poly implies that L is decided by a circuit family $\{C_n\}$, where $|C_n| = n^{O(1)}$. We don't require that C_n is poly-time computable from 1^n.
- P/poly is a *non-uniform class* as a language in this class is allowed to have different algorithms/circuits for different input lengths.
- P is a *uniform class* as a language in this class has one algorithm for all inputs.

Class P/poly

- What makes P/poly contain undecidable languages? *Ans:* $L \in$ P/poly implies that L is decided by a circuit family $\{C_n\}$, where $|C_n| = n^{O(1)}$. We don't require that C_n is poly-time computable from 1^n.
- P/poly is a *non-uniform class* as a language in this class is allowed to have different algorithms/circuits for different input lengths.
- P is a *uniform class* as a language in this class has one algorithm for all inputs.

Model	What it captures
TM (uniform)	An algo for all inputs
Circuits (non-uniform)	An algo per i/p length

Class P/poly

- What makes P/poly contain undecidable languages? *Ans:* $L \in$ P/poly implies that L is decided by a circuit family $\{C_n\}$, where $|C_n| = n^{O(1)}$. We don't require that C_n is poly-time computable from 1^n.
- P/poly is a *non-uniform class* as a language in this class is allowed to have different algorithms/circuits for different input lengths.
- P is a *uniform class* as a language in this class has one algorithm for all inputs.
- Is SAT $\in$ P/poly? In other words, is NP $\subsetneq$ P/poly?

Karp-Lipton theorem

- Theorem *(Karp & Lipton 1982)*. If NP $\subsetneq$ P/poly then PH $= \Sigma_2$.
- Proof. We'll show that NP $\subsetneq$ P/poly implies $\Pi_2 = \Sigma_2$. It's sufficient to show that $\Pi_2 \subseteq \Sigma_2$.

Karp-Lipton theorem

- Theorem *(Karp & Lipton 1982)*. If $NP \subsetneq P/poly$ then $PH = \Sigma_2$.
- Proof. Let $L \in \Pi_2$. There's a polynomial function $q(.)$ and a poly-time TM M s.t.

$$x \in L \Longleftrightarrow \forall u_1 \in \{0,1\}^{q(|x|)}\ \exists u_2 \in \{0,1\}^{q(|x|)}\ M(x,u_1,u_2) = 1.$$

Karp-Lipton theorem

- Theorem *(Karp & Lipton 1982)*. If NP $\subsetneq$ P/poly then PH = Σ_2 .
- Proof. Let L $\in \Pi_2$. There's a polynomial function q(.) and a poly-time TM M s.t.

  $x \in L \Leftrightarrow \forall u_1 \in \{0,1\}^{q(|x|)} \, \exists u_2 \in \{0,1\}^{q(|x|)} \; M(x,u_1, u_2) = 1.$

- Goal. Come up with a polynomial function p(.) and a poly-time TM N s.t.

  $x \in L \Leftrightarrow \exists v_1 \in \{0,1\}^{p(|x|)} \, \forall v_2 \in \{0,1\}^{p(|x|)} \; N(x,v_1, v_2) = 1.$

- Think about designing such a TM N.

Karp-Lipton theorem

- Theorem *(Karp & Lipton 1982)*. If NP $\subsetneq$ P/poly then PH = Σ_2 .
- Proof. Let $L \in \Pi_2$. There's a polynomial function $q(.)$ and a poly-time TM M s.t.

by Cook-Levin

$x \in L \iff \forall u_1 \in \{0,1\}^{q(|x|)}\ \exists u_2 \in \{0,1\}^{q(|x|)}\ \varphi(x,u_1, u_2) = 1.$

Karp-Lipton theorem

- Theorem *(Karp & Lipton 1982)*. If NP $\subsetneq$ P/poly then PH = Σ_2 .
- Proof. Let L $\in \Pi_2$. There's a polynomial function q(.) and a poly-time TM M s.t.

by Cook-Levin

$$x \in L \Leftrightarrow \forall u_1 \in \{0,1\}^{q(|x|)} \exists u_2 \in \{0,1\}^{q(|x|)} \varphi(x,u_1, u_2) = 1.$$

- If M runs in time $T(n) = n^{O(1)}$ on (x,u_1, u_2), where $|x| = n$, then $|\varphi| = O(T(n)^2)$. Let m = #(bits to write φ).
- N can compute φ from M in poly(|x|) time.

Karp-Lipton theorem

- Theorem *(Karp & Lipton 1982)*. If NP $\subsetneq$ P/poly then PH = Σ_2 .
- Proof. Let $L \in \Pi_2$. There's a polynomial function $q(.)$ and a poly-time TM M s.t.

  by Cook-Levin

  $x \in L \Leftrightarrow \forall u_1 \in \{0,1\}^{q(|x|)}\ \exists u_2 \in \{0,1\}^{q(|x|)}\ \varphi(x,u_1, u_2) = 1.$

- If M runs in time $T(n) = n^{O(1)}$ on (x,u_1, u_2), where $|x| = n$, then $|\varphi| = O(T(n)^2)$. Let m = length of φ .
- N can compute φ from M in poly(|x|) time.

Karp-Lipton theorem

- Theorem *(Karp & Lipton 1982)*. If NP $\subsetneq$ P/poly then PH = Σ_2 .
- Proof. Let L $\in \Pi_2$. There's a polynomial function q(.) and a poly-time TM M s.t.

$$x \in L \Leftrightarrow \forall u_1 \in \{0,1\}^{q(|x|)} \exists u_2 \in \{0,1\}^{q(|x|)} \varphi(x,u_1,u_2) = 1.$$

$\varphi(x,u_1,u_2)$ as a function of u_2 is satisfiable. Wlog φ is a CNF (why?).

Karp-Lipton theorem

- Theorem *(Karp & Lipton 1982)*. If NP $\subsetneq$ P/poly then PH = Σ_2 .
- Proof. Let $L \in \Pi_2$. There's a polynomial function q(.) and a poly-time TM M s.t.

  $x \in L \Leftrightarrow \forall u_1 \in \{0,1\}^{q(|x|)} \; \varphi(x,u_1, u_2) \in SAT.$

- By assumption, SAT $\in$ P/poly, i.e., there's a circuit C_m of size $p(m) = m^{O(1)}$ that correctly decides satifiability of all input circuits φ of length m.

Karp-Lipton theorem

- Theorem *(Karp & Lipton 1982)*. If NP $\subsetneq$ P/poly then PH = Σ_2 .
- Proof. Let L $\in \Pi_2$. There's a polynomial function q(.) and a poly-time TM M s.t.

  $x \in L \Leftrightarrow \forall u_1 \in \{0,1\}^{q(|x|)}\ \varphi(x,u_1,u_2) \in SAT.$

- First attempt. A Σ_2 statement to capture membership of strings in L.

  $x \in L \Leftrightarrow \exists C_m \in \{0,1\}^{p(m)}\ \forall u_1 \in \{0,1\}^{q(|x|)}\ C_m(\varphi(x,u_1,u_2))=1.$

Karp-Lipton theorem

- Theorem *(Karp & Lipton 1982)*. If NP $\subsetneq$ P/poly then PH = Σ_2 .
- Proof. Let L $\in \Pi_2$. There's a polynomial function q(.) and a poly-time TM M s.t.

  $x \in L \Leftrightarrow \forall u_1 \in \{0,1\}^{q(|x|)} \ \varphi(x,u_1, u_2) \in SAT.$
- First attempt. A Σ_2 statement to capture membership of strings in L.

  $x \in L \Leftrightarrow \exists C_m \in \{0,1\}^{p(m)} \ \forall u_1 \in \{0,1\}^{q(|x|)} \ C_m(\varphi(x,u_1, u_2))=1.$

- Wrong! Think about a C_m that always outputs 1.

Karp-Lipton theorem

- Theorem *(Karp & Lipton 1982)*. If NP $\subsetneq$ P/poly then PH = Σ_2 .
- Proof. Let L $\in \Pi_2$. There's a polynomial function q(.) and a poly-time TM M s.t.

  $x \in L \Leftrightarrow \forall u_1 \in \{0,1\}^{q(|x|)}\ \varphi(x,u_1, u_2) \in SAT.$
- First attempt. A Σ_2 statement to capture membership of strings in L.

  $x \in L \Leftrightarrow \exists C_m \in \{0,1\}^{p(m)}\ \forall u_1 \in \{0,1\}^{q(|x|)}\ C_m(\varphi(x,u_1, u_2))=1.$

- Need to be sure that C_m is the right circuit.

Karp-Lipton theorem

- Theorem *(Karp & Lipton 1982)*. If NP $\subsetneq$ P/poly then PH = Σ_2.
- Proof. Let L $\in \Pi_2$. There's a polynomial function q(.) and a poly-time TM M s.t.

  $x \in L \Leftrightarrow \forall u_1 \in \{0,1\}^{q(|x|)}\ \varphi(x,u_1, u_2) \in SAT.$
- If there's a circuit C_m of size $m^{O(1)}$ that correctly decides satifiability of all input circuits φ of length m, then by self-reducibility of SAT, there's a multi-output circuit D_m of size $r(m) = m^{O(1)}$ that outputs a *satisfying assignment* for input φ if $\varphi \in$ SAT. *(Homework)*

Karp-Lipton theorem

- Theorem *(Karp & Lipton 1982)*. If $NP \subsetneq P/poly$ then $PH = \Sigma_2$.
- Proof. Let $L \in \Pi_2$. There's a polynomial function $q(.)$ and a poly-time TM M s.t.

$$x \in L \Leftrightarrow \forall u_1 \in \{0,1\}^{q(|x|)} \ \varphi(x,u_1,u_2) \in SAT.$$

- A Σ_2 statement to capture membership in L.

$$x \in L \Leftrightarrow$$

$$\exists D_m \in \{0,1\}^{r(m)} \ \forall u_1 \in \{0,1\}^{q(|x|)} \ \varphi(x,u_1,\underbrace{D_m(\varphi(x,u_1,u_2))}_{\text{assignment to the } u_2 \text{ variables}}) = 1.$$

Karp-Lipton theorem

- Theorem *(Karp & Lipton 1982)*. If NP $\subsetneq$ P/poly then PH = Σ_2 .
- Proof. Let $L \in \Pi_2$. There's a polynomial function $q(.)$ and a poly-time TM M s.t.

  $x \in L \Leftrightarrow \forall u_1 \in \{0,1\}^{q(|x|)} \ \varphi(x,u_1, u_2) \in SAT.$

- A Σ_2 statement to capture membership in L.

  $x \in L \Leftrightarrow$

  $\exists D_m \in \{0,1\}^{r(m)} \ \forall u_1 \in \{0,1\}^{q(|x|)} \ \underbrace{\varphi(x,u_1, D_m(\varphi(x,u_1, u_2)) = 1}_{\text{Can be checked by a poly-time TM } N.}.$

Can be checked by a poly-time TM N.

Karp-Lipton theorem

- Theorem *(Karp & Lipton 1982)*. If NP $\subsetneq$ P/poly then PH = Σ_2 .
- Proof. Let L $\in \Pi_2$. There's a polynomial function q(.) and a poly-time TM M s.t.

  $x \in L \Leftrightarrow \forall u_1 \in \{0,1\}^{q(|x|)}\ \varphi(x,u_1, u_2) \in SAT.$
- A Σ_2 statement to capture membership in L.

  $x \in L \Leftrightarrow$

  $\exists D_m \in \{0,1\}^{r(m)}\ \forall u_1 \in \{0,1\}^{q(|x|)}\ N(x, D_m, u_1) = 1.$

Karp-Lipton theorem

- Theorem *(Karp & Lipton 1982)*. If $NP \subsetneq P/poly$ then $PH = \Sigma_2$.

- If we can show $NP \not\subset P/poly$ assuming $P \neq NP$, then

  $NP \not\subset P/poly \Leftrightarrow P \neq NP$.

- Karp-Lipton theorem shows $NP \not\subset P/poly$ assuming the stronger statement $PH \neq \Sigma_2$.

Functions outside P/poly

- Are there Boolean functions (i.e., languages) outside P/poly?

Functions outside P/poly

- Are there Boolean functions (i.e., languages) outside P/poly? Yes! There are many. Let $\exp(m) = 2^m$.
- Theorem. $1 - \exp(-2^{n-1})$ fraction of Boolean functions on n variables **do not** have circuits of size $2^n/(22n)$.
- Proof. Follows from a counting argument.

Functions outside P/poly

- Are there Boolean functions (i.e., languages) outside P/poly? Yes! There are many. Let $\exp(m) = 2^m$.
- Theorem. $1 - \exp(-2^{n-1})$ fraction of Boolean functions on n variables **do not** have circuits of size $2^n/(22n)$.
- Proof. Let $s = 2^n/(22n)$. A circuit of size s has at most s internal nodes. It can be specified by giving the labels of the internal nodes and the adjacency lists.

Functions outside P/poly

- Are there Boolean functions (i.e., languages) outside P/poly? Yes! There are many. Let $\exp(m) = 2^m$.
- Theorem. $1 - \exp(-2^{n-1})$ fraction of Boolean functions on n variables **do not** have circuits of size $2^n/(22n)$.
- Proof. Let $s = 2^n/(22n)$. A circuit of size s has at most s internal nodes. It can be specified by giving the labels of the internal nodes and the adjacency lists.
- Number of bits required to write the adjacency lists it at most $s(\log s + 3) + 4(s + n) \leq 9s.\log s$.

Functions outside P/poly

- Are there Boolean functions (i.e., languages) outside P/poly? Yes! There are many. Let $\exp(m) = 2^m$.
- Theorem. $1 - \exp(-2^{n-1})$ fraction of Boolean functions on n variables **do not** have circuits of size $2^n/(22n)$.
- Proof. Let $s = 2^n/(22n)$. A circuit of size s has at most s internal nodes. It can be specified by giving the labels of the internal nodes and the adjacency lists.
- Number of circuits of size s is at most $3^s.2^{9s.\log s}$.

Functions outside P/poly

- Are there Boolean functions (i.e., languages) outside P/poly? Yes! There are many. Let $\exp(m) = 2^m$.
- Theorem. $1 - \exp(-2^{n-1})$ fraction of Boolean functions on n variables **do not** have circuits of size $2^n/(22n)$.
- Proof. Let $s = 2^n/(22n)$. A circuit of size s has at most s internal nodes. It can be specified by giving the labels of the internal nodes and the adjacency lists.
- Number of circuits of size s is at most $2^{11 s.\log s}$.

Functions outside P/poly

- Are there Boolean functions (i.e., languages) outside P/poly? Yes! There are many. Let $\exp(m) = 2^m$.
- Theorem. 1- $\exp(-2^{n-1})$ fraction of Boolean functions on n variables **do not** have circuits of size $2^n/(22n)$.
- Proof. Let $s = 2^n/(22n)$. A circuit of size s has at most s internal nodes. It can be specified by giving the labels of the internal nodes and the adjacency lists.
- Number of circuits of size s is at most $\exp(2^{n-1})$.
- Number of functions in n variables is $\exp(2^n)$.

Functions outside P/poly

- Are there Boolean functions (i.e., languages) outside P/poly? Yes! There are many. Let $\exp(m) = 2^m$.
- Theorem. $1 - \exp(-2^{n-1})$ fraction of Boolean functions on n variables **do not** have circuits of size $2^n/(22n)$.
- Proof. Let $s = 2^n/(22n)$. A circuit of size s has at most s internal nodes. It can be specified by giving the labels of the internal nodes and the adjacency lists.
- So, circuits of size s can compute at most $\exp(-2^{n-1})$ fraction of all Boolean functions on n variables.

Functions outside P/poly

- Are there Boolean functions (i.e., languages) outside P/poly? Yes! There are many.
- Is one out of so many functions outside P/poly in NP?

Functions outside P/poly

- Are there Boolean functions (i.e., languages) outside P/poly? Yes! There are many.
- Is one out of so many functions outside P/poly in NP? We don't know even after ~40 yrs of research!

- Theorem. *(Iwama, Lachish, Morizumi & Raz 2002)* There is a language $L \in NP$ such that any circuit C_n that decides $L \cap \{0,1\}^n$ requires $5n - o(n)$ many $\wedge$ and $\vee$ gates.

Functions outside P/poly

- Are there Boolean functions (i.e., languages) outside P/poly? Yes! There are many.
- Is one out of so many functions outside P/poly in NP? We don't know even after ~40 yrs of research!

- Theorem. *(Iwama, Lachish, Morizumi & Raz 2002)* There is a language $L \in NP$ such that any circuit C_n that decides $L \cap \{0,1\}^n$ requires $5n - o(n)$ many $\wedge$ and $\vee$ gates.

↑ Results of this kind are known as circuit lower bound.

Functions outside P/poly

- Are there Boolean functions (i.e., languages) outside P/poly? Yes! There are many.
- Is one out of so many functions outside P/poly in NP? We don't know even after ~40 yrs of research!

- Open problem. Prove that $\mathsf{NEXP} \not\subset \mathsf{P/poly}$.

Lower bounds for restricted circuits

- Nevertheless, the clean combinatorial structure of a circuit has been used to prove lower bounds for some natural classes of circuits.
- The proofs of these lower bounds introduced and developed some highly interesting techniques.

Lower bounds for restricted circuits

- Nevertheless, the clean combinatorial structure of a circuit has been used to prove lower bounds for some natural classes of circuits.
- The proofs of these lower bounds introduced and developed some highly interesting techniques.

- Fact. PARITY(x_1, x_2, …, x_n) can be computed by a circuit of size $O(n)$ and a formula of size $O(n^2)$.

Homework

Lower bound for Boolean formulas

- Nevertheless, the clean combinatorial structure of a circuit has been used to prove lower bounds for some natural classes of circuits.
- The proofs of these lower bounds introduced and developed some highly interesting techniques.

- Theorem. *(Khrapchenko 1971)* Any formula computing PARITY$(x_1, x_2, \ldots, x_n)$ has size $\Omega(n^2)$.

Lower bound for Boolean formulas

- Nevertheless, the clean combinatorial structure of a circuit has been used to prove lower bounds for some natural classes of circuits.
- The proofs of these lower bounds introduced and developed some highly interesting techniques.

- Theorem. (*Andreev 1987, Hastad 1998*) There's a f that can be computed by a $O(n)$-size circuit such that any formula computing f has size $\Omega(n^{3-o(1)})$.

Technique: Shrinkage of formulas under random restrictions (*Subbotovskaya 1961*).

Lower bound for Boolean formulas

- Nevertheless, the clean combinatorial structure of a circuit has been used to prove lower bounds for some natural classes of circuits.
- The proofs of these lower bounds introduced and developed some highly interesting techniques.

- Conjecture. *(Circuits more powerful than formulas)* There's a f that can be computed by a $O(n)$-size circuit such that any formula computing f has size $n^{\omega(1)}$.

An interesting approach was given by *Karchmer, Raz & Wigderson (1995)* .

LB for AC^0 circuits

- Nevertheless, the clean combinatorial structure of a circuit has been used to prove lower bounds for some natural classes of circuits.
- The proofs of these lower bounds introduced and developed some highly interesting techniques.

- We'll discuss a **super-polynomial** lower bound for constant depth circuits (a.k.a. AC^0 circuits) later.

Non-uniform size hierarchy

- Shanon's result. There's a constant $c \geq 1$ such that every Boolean function in n variables has a circuit of size at most $c.(2^n/n)$.

- Theorem. There's a constant $d \geq 1$ s.t. if T_1: $N \rightarrow N$ & T_2: $N \rightarrow N$ and $T_1(n) \leq d^{-1}.T_2(n) \leq T_2(n) \leq c.(2^n/n)$ then

$$SIZE(T_1(n)) \subsetneq SIZE(T_2(n)).$$

Non-uniform size hierarchy

- Shanon's result. There's a constant $c \geq 1$ such that every Boolean function in n variables has a circuit of size at most $c.(2^n/n)$.

- Theorem. There's a constant $d \geq 1$ s.t. if $T_1: \mathbb{N} \rightarrow \mathbb{N}$ & $T_2: \mathbb{N} \rightarrow \mathbb{N}$ and $T_1(n) \leq d^{-1}.T_2(n) \leq T_2(n) \leq c.(2^n/n)$ then

$$\text{SIZE}(T_1(n)) \subsetneq \text{SIZE}(T_2(n)).$$

- Proof. Uses Shanon's result and a counting argument.

(*Homework*)