Advanced Computational Complexity

George Voutsadakis[1]

[1]Mathematics and Computer Science
Lake Superior State University

LSSU Math 600

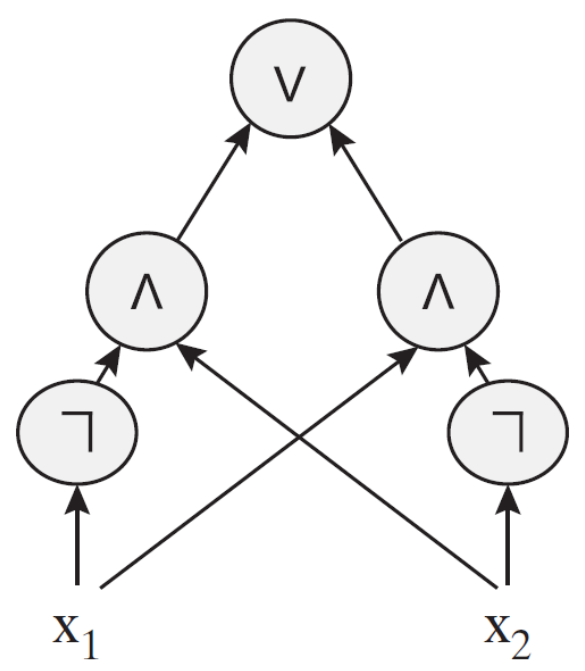
Subsection 1

Boolean Circuits and $P_{/POLY}$

Introducing Boolean Circuits

- A Boolean circuit is a diagram showing how to derive an output from a binary input string by applying a sequence of basic Boolean operations OR ($\vee$), AND ($\wedge$) and NOT ($\neg$) on the input bits.

  Example: The figure shows a Boolean circuit computing the XOR function on two bits.



Boolean Circuits

Definition (Boolean Circuits)

For every $n \in \mathbb{N}$, an ***n*-input, single-output Boolean circuit** is a directed acyclic graph with:

- n **sources** (vertices with no incoming edges);
- One **sink** (vertex with no outgoing edges).

All non source vertices are called **gates** and are labeled with one of $\vee$, $\wedge$ or $\neg$ (i.e., the logical operations OR, AND and NOT).

- The vertices labeled with $\vee$ and $\wedge$ have fan-in (i.e., number of incoming edges) equal to 2;
- The vertices labeled with $\neg$ have fan-in 1.

The **size of** C, denoted by $|C|$, is the number of vertices in it.

Boolean Circuits (Cont'd)

Definition (Boolean Circuits Cont'd)

Let C be a Boolean circuit.
Let $x \in \{0,1\}^n$ be some input.
Then the **output of** C **on** x, denoted $C(x)$, is defined in the natural way.
For every vertex v of C, we give it a value $\text{val}(v)$ as follows:

- If v is the i-th input vertex then $\text{val}(v) = x_i$;
- Otherwise, $\text{val}(v)$ is defined recursively by applying v's logical operation on the values of the vertices connected to v.

The **output** $C(x)$ is the value of the output vertex.

Remarks on Fan-in and Fan-out

- The definition restricts fan-in to 2.
- This, however, does not restrict generality.
- A $\vee$ or $\wedge$ gate with fan-in f can be easily replaced with a subcircuit consisting of $f-1$ gates of fan-in 2.
- Fan-in becomes important when we study circuits with restricted depth.
- The Boolean formulas we looked at earlier are circuits where the fan-out (i.e., number of outgoing edges) of each vertex is 1.
- The advantage of fan-out 2 over fan-out 1 is that it allows an intermediate value inside the circuit to be reused many times.
- This definition has the additional advantage that it models the silicon chips used in modern computers.
- Thus, if we show that a certain task can be solved by a small Boolean circuit, then it can be implemented efficiently on a silicon chip.

Circuit Families and Language Recognition

Definition (Circuit Families and Language Recognition)

Let $T : \mathbb{N} \to \mathbb{N}$ be a function.
A $T(n)$-**size circuit family** is a sequence

$$\{C_n\}_{n \in \mathbb{N}}$$

of Boolean circuits, where:

- C_n has n inputs and a single output;
- Its size $|C_n| \leq T(n)$, for every n.

A language L is in SIZE($T(n)$) if there exists a $T(n)$-size circuit family $\{C_n\}_{n \in \mathbb{N}}$, such that, for every $x \in \{0,1\}^n$,

$$x \in L \quad \text{iff} \quad C_n(x) = 1.$$

Examples

- Consider the language
$$\{1^n : n \in \mathbb{N}\}.$$
It can be decided by a linear-sized circuit family.
The circuit is simply a tree of AND gates that computes the AND of all input bits.
- Consider the language
$$\{\langle m, n, m+n\rangle : m, n \in \mathbb{N}\}.$$
It also has linear-sized circuits.
They implement the grade school algorithm for addition.
This algorithm adds two numbers bit by bit.
Addition of two bits is done by a circuit of $O(1)$ size.
It produces a carry bit that is used as input for the addition of the bits in the next position.

The class $\mathrm{P}_{/\mathrm{poly}}$

- A CNF formula is a special type of a circuit.
- So every function f from $\{0,1\}^n$ to $\{0,1\}$ can be computed by a Boolean circuit of size $n2^n$.
- We can show that size $\mathrm{O}\left(\frac{2^n}{n}\right)$ also suffices.
- Therefore, interesting complexity classes arise when we consider "small" circuits.

Definition (The class $\mathrm{P}_{/\mathrm{poly}}$)

$\mathrm{P}_{/\mathrm{poly}}$ is the class of languages that are decidable by polynomial-sized circuit families. That is,

$$\mathrm{P}_{/\mathrm{poly}} = \bigcup_c \mathrm{SIZE}(n^c).$$

- From the complexity theoretic point of view, the hope is to eventually show that languages such as SAT are not in $\mathrm{P}_{/\mathrm{poly}}$.

Straight-Line Programs

- Instead of modeling Boolean circuits as labeled graphs, we can also model them as **straight-line programs**.
- A program P is **straight-line** if it contains no branching or loop operations (such as `if` or `goto`).
- Hence, P's running time is bounded by the number of instructions that it contains.
- The equivalence between Boolean circuits and straight-line programs is fairly general and holds (up to polynomial factors) for essentially any reasonable programming language.

Boolean Circuits and Boolean Straight-Line Programs

- The equivalence between Boolean circuits and straight-line programs is most obviously demonstrated using straight-line programs with Boolean operations.
- A **Boolean straight-line program of length** T, with input variables

$$x_1, x_2, \ldots, x_n \in \{0, 1\},$$

is a sequence of T statements of the form

$$y_i = z_{i_1} \ \mathrm{OP} \ z_{i_2}, \quad i = 1, 2, \ldots, T,$$

where:

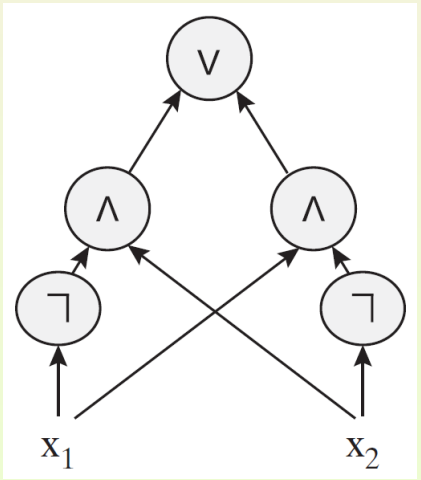
- OP is either $\vee$ or $\wedge$;
- Each z_{i_1}, z_{i_2} is either an input variable, or the negation of an input variable, or y_j for $j < i$.

Boolean Straight-Line Computation

- For every setting of values to the input variables, the **straight-line computation** consists of executing these simple statements in order, thereby finding values for $y_1, y_2, \ldots, y_T$.
- The **output** of the computation is the value of y_T.
- It is can be shown that a function f on n bits can be computed by an S-line straight-line program of this form if and only if it can be computed by an S-sized Boolean circuit.

  Example: A straight-line program in input variables x_1, x_2 that is equivalent to the XOR function circuit is

$$\begin{aligned} y_1 &= \neg x_1; \\ y_2 &= \neg x_2; \\ y_3 &= y_1 \wedge x_2; \\ y_4 &= x_1 \wedge y_2; \\ y_5 &= y_3 \vee y_4. \end{aligned}$$



$\mathsf{P}_{/\mathrm{poly}}$ and P

Theorem

$\mathsf{P} \subseteq \mathsf{P}_{/\mathrm{poly}}$.

- The proof is very similar to the proof of the Cook-Levin Theorem.

  Recall that every TM M that runs in $\mathrm{O}(T(n))$ time can be simulated by an oblivious TM $\widetilde{M}$ (whose head movement is independent of its input) running in time $\mathrm{O}\left(T(n)^2\right)$ (even $\mathrm{O}(T(n)\log T(n))$ if we perform a more careful construction).

  Thus, it suffices to show that, for every oblivious TM M that runs in time $T(n)$, there exists a $\mathrm{O}(T(n))$-sized circuit family $\{C_n\}_{n\in\mathbb{N}}$, such that

$$C_n(x) = M(x), \quad \text{for every } x \in \{0,1\}^n.$$

$P_{/poly}$ and P (Cont'd)

- Let M be such an oblivious TM.

  Let $x \in \{0,1\}^*$ be some input for M.

  Define the **transcript** of M's execution on x to be the sequence

  $$z_1, \ldots, z_{T(n)}$$

  of **snapshots** (the machine's state and symbols read by all heads) of the execution at each step in time.

  We can encode each z_i by a constant-sized binary string.

  Furthermore, we can compute the string z_i based on:

  - The input x;
  - The previous snapshot z_{i-1};
  - The snapshots $z_{i_1}, \ldots, z_{i_k}$, where z_{i_j} denotes the last step that M's j-th head was in the same position as it is in the i-th step.

$P_{/poly}$ and P (Cont'd)

- These are only a constant number of strings of constant length.

  This means that we can compute z_i from these previous snapshots using a constant-sized circuit.

  The composition of all these constant-sized circuits gives rise to a circuit that computes, from the input x, the snapshot $z_{T(n)}$ of the last step of M's execution on x.

  There is a constant-sized circuit that, given $z_{T(n)}$, outputs 1 if and only if $z_{T(n)}$ is an accepting snapshot (M outputs 1 and halts).

  Thus, there is an $O(T(n))$-sized circuit C_n, such that

$$C_n(x) = M(x), \quad \text{for all } x \in \{0,1\}^n.$$

Remark

- The circuit produced in the proof of the theorem:
  - Is of polynomial size;
  - Can also be computed in polynomial time;
  - Can even be computed in logarithmic space.
- This is based on the observation that it is possible to simulate every TM M by an oblivious TM $\widetilde{M}$, such that the function that maps n, i to the $\widetilde{M}$'s position on n-length inputs in the i-th tape can be computed in logarithmic space.

P is a Proper Subset of $P_{/poly}$

- The inclusion $P \subseteq P_{/poly}$ is proper.
- There are unary languages that are undecidable and, hence, not in P.
- In contrast, every unary language is in $P_{/poly}$.

Claim

If $L \in \{0,1\}^*$ is a unary language i.e., $L \in \{1^n : n \in \mathbb{N}\}$ then,

$$L \in P_{/poly}.$$

- We describe a circuit family of linear size.
  If $1^n \in L$, then the circuit for inputs of size n has already been given.
  Otherwise, it is the circuit that always outputs 0.
  A unary language that is undecidable is

$$\text{UHALT} = \{1^n : n\text{'s binary expansion encodes a pair } \langle M, x\rangle, \text{ such that } M \text{ halts on input } x\}.$$

Circuit Satisfiability

- We aim to provide an alternative proof of the Cook-Levin Theorem, based on circuits.
- To this end, we define the Circuit Satisfiability problem.

Definition (Circuit Satisfiability or CktSat)

The language CktSat consists of all (strings representing) circuits that produce a single bit of output and that have a satisfying assignment. That is, a string representing an n-input circuit C is in CktSat iff, there exists $u \in \{0,1\}^n$, such that

$$C(u) = 1.$$

Circuit Satisfiability and The Cook-Levin Theorem

- CktSat $\in$ NP, since a satisfying assignment is a certificate.
- The Cook-Levin Theorem follows from the next two lemmas.

Lemma

CktSat is NP-hard.

- Suppose $L \in$ NP.

  Then there is a polynomial-time TM M and a polynomial p, such that

$$x \in L \quad \text{iff} \quad M(x,u) = 1, \text{ for some } u \in \{0,1\}^{p(|x|)}.$$

  The proof of the theorem yields a polynomial-time transformation from M, x to a circuit C, such that

$$M(x,u) = C(u), \quad \text{for every } u \in \{0,1\}^{\text{poly}(|x|)}.$$

  Thus, x is in L iff $C \in$ CktSat.

The Second Lemma

Lemma

$\textsc{CktSat} \leq_p \textsc{3Sat}$.

- Let C be a circuit.
  We map it to a 3CNF formula φ as follows.
  For every node v_i of C, we have a corresponding variable z_i in φ.
  Suppose the node v_i is an AND of the nodes v_j and v_k.
  Then we add to φ clauses that are equivalent to the condition

$$\text{“}z_i = (z_j \wedge z_k)\text{”}.$$

  That is, we add

$$(\overline{z}_i \vee (z_j \wedge z_k)) \wedge (z_i \vee \neg(z_j \wedge z_k))$$
$$\equiv (\overline{z_i} \vee \overline{z_j} \vee z_k) \wedge (\overline{z_i} \vee z_j \vee \overline{z_k}) \wedge (\overline{z_i} \vee z_j \vee z_k) \wedge (z_i \vee \overline{z_j} \vee \overline{z_k}).$$

The Second Lemma (Cont'd)

- Suppose v_i is an OR of v_j and v_k.

  Then we add clauses equivalent to

  $$\text{“}z_i = (z_j \vee z_k)\text{”}.$$

  Suppose v_i is the NOT of v_j.

  Then we add the clauses

  $$(z_i \vee z_j) \wedge (\overline{z_i} \vee \overline{z_j}).$$

  Finally, suppose v_i is the output node of C.

  Then we add the clause (z_i) to φ.

  I.e., we add the clause that is true iff z_i is true.

  Then formula φ is satisfiable if and only if the circuit C is.

  Clearly, the reduction runs in time polynomial in the input size.

Subsection 2

Uniformly Generated Circuits

P-Uniform Circuit Families

- The class $\mathrm{P}_{/\mathrm{poly}}$ fits rather awkwardly in the complexity world since it contains even undecidable languages such as UHALT.
- The root of the problem is that for a language L to be in $\mathrm{P}_{/\mathrm{poly}}$, it suffices that a circuit family for L exists, even if we have no way of actually constructing the circuits.
- This motivates trying to restrict attention to circuits that can actually be built using a fairly efficient Turing machine.

Definition (P-Uniform Circuit Families)

A circuit family $\{C_n\}$ is P-**uniform** if there is a polynomial-time TM that, on input 1^n, outputs the description of the circuit C_n.

- Restricting circuits to be P-uniform "collapses" $\mathrm{P}_{/\mathrm{poly}}$ to the class P.

P-uniformity Collapses $\mathrm{P}_{/\mathrm{poly}}$ to P

Theorem

A language L is computable by a P-uniform circuit family iff $L \in \mathrm{P}$.

- Suppose L is computable by a circuit family $\{C_n\}$ that is generated by a polynomial-time TM M.

  Then we can come up with a polynomial-time TM $\widetilde{M}$ for L.

  Suppose $\widetilde{M}$ receives input x.
  $\widetilde{M}$ runs $M(1^{|x|})$ to obtain the circuit $C_{|x|}$.
  It then evaluates $C_{|x|}$ on the input x.

  For the other direction, assume $L \in \mathrm{P}$.

  Follow closely the proof of the inclusion $\mathrm{P} \subseteq \mathrm{P}_{/\mathrm{poly}}$.

  Note that it actually yields a P-uniform circuit family for any $L \in \mathrm{P}$.

Logspace-Uniform Families

- We now impose an even stricter notion of uniformity.
- We stipulate generation by logspace machines.
- A function $f : \{0,1\}^* \to \{0,1\}^*$ is **implicitly logspace computable** if the mapping

$$x, i \mapsto f(x)_i$$

can be computed in logarithmic space.

Definition (Logspace-Uniform Circuit Families)

A circuit family $\{C_n\}$ is **logspace uniform** if there is an implicitly logspace computable function mapping 1^n to the description of the circuit C_n.

- Logspace computations run in polynomial time.
- Hence, logspace-uniform circuits are also P-uniform.

A Concrete String Representation

- The definition of logspace uniform circuits is robust to various representations as strings.
- A concrete way is to represent a circuit of size S by:
  - The $S \times S$ adjacency matrix of its underlying directed graph;
  - An array of size S that provides the labels (gate type) of each vertex.
- Identifying the vertices with numbers in $[S]$, we let:
  - The first n vertices be the input vertices;
  - The last vertex be the output vertex.

Representation and Conditions for Logspace Uniformity

- Assume the concrete matrix representation of a family $\{C_n\}$.
- Then $\{C_n\}$ is logspace-uniform if and only if the following functions are computable in $O(\log n)$ space:
  - SIZE(n) returns the size S (in binary representation) of the circuit C_n;
  - TYPE(n, i), where $i \in [m]$, returns the label of the i-th vertex of C_n. I.e., it returns one of $\{\vee, \wedge, \neg, \mathrm{NONE}\}$;
  - EDGE(n, i, j) returns 1 if there is a directed edge in C_n from the i-th vertex to the j-th vertex.
- Both the inputs and the outputs of these functions can be encoded using a logarithmic (in $|C_n|$) number of bits.

Logspace-Uniform Circuits of Polynomial Size

Theorem

A language has logspace-uniform circuits of polynomial size iff it is in P.

- The result follows by a careful analysis of the proof of $\mathsf{P} \subseteq \mathsf{P}_{/\mathsf{poly}}$.

Subsection 3

Turing Machines that Take Advice

Turing Machines that Take Advice

- We characterize $\mathsf{P}_{/\mathsf{poly}}$ using Turing machines that "take advice".
- Such a machine has, for each n, an **advice string** α_n, which it is allowed to use in its computation whenever the input has size n.

Definition (Turing Machines with Advice)

Let $T, a : \mathbb{N} \to \mathbb{N}$ be functions. The class of **languages decidable by time-$T(n)$ TMs with $a(n)$ bits of advice**, denoted $\mathrm{DTIME}(T(n))/a(n)$, contains every L, such that, there exists a sequence $\{\alpha_n\}_{n\in\mathbb{N}}$ of strings, with $\alpha_n \in \{0,1\}^{a(n)}$, and a TM M, such that:

- For every $x \in \{0,1\}^n$,

$$M(x, \alpha_n) = 1 \quad \text{iff} \quad x \in L;$$

- On input (x, α_n), the machine M runs for at most $\mathrm{O}(T(n))$ steps.

Example

- Consider an arbitrary unary language.
- It can be be decided by a polynomial time Turing machine with 1 bit of advice.
- The advice string for inputs of length n is the bit indicating whether or not 1^n is in the language.
- In particular, this is true of the language UHALT.

Polynomial-Time TM's with Advice Decide $\mathrm{P}_{/\mathrm{poly}}$

Theorem (Polynomial-Time TM's with Advice Decide $\mathrm{P}_{/\mathrm{poly}}$)

$$\mathrm{P}_{/\mathrm{poly}} = \bigcup_{c,d} \mathrm{DTIME}(n^c)/n^d.$$

- Suppose $L \in \mathrm{P}_{/\mathrm{poly}}$.

  Then L is computable by a polynomial-sized circuit family $\{C_n\}$.

  Use the description of C_n as an advice string on inputs of size n.

  The TM, taking this advice, is the polynomial-time TM M that, on input a string x and a string representing an n-input circuit C, outputs $C(x)$.

  Conversely, suppose L is decidable by a polynomial-time Turing machine M, with access to advice $\{\alpha_n\}_{n\in\mathbb{N}}$ of polynomial size $a(n)$.

  Then we use the idea in the $\mathrm{P} \subseteq \mathrm{P}_{/\mathrm{poly}}$-theorem.

Polynomial-Time TM's with Advice Decide $P_{/poly}$ (Cont'd)

- We construct, for every n, a polynomial-sized circuit D_n, such that on every $x \in \{0,1\}^n$, $\alpha \in \{0,1\}^{a(n)}$,

$$D_n(x, \alpha) = M(x, \alpha).$$

We let the circuit C_n be the polynomial circuit that, given x computes the value $D_n(x, \alpha_n)$.

I.e., C_n is equal to the circuit D_n with the string α_n "hard-wired" as its second input.

"Hard-wiring" an input into a circuit means:

- Taking a circuit C, with two inputs $x \in \{0,1\}^n$, $y \in \{0,1\}^m$;
- Fixing the inputs corresponding to y.

This gives the circuit C_y that, for every x, returns $C(x, y)$.

It is easy to do so while ensuring that the size of C_y is not greater than the size of C.

Subsection 4

$P_{/POLY}$ and NP

The Karp-Lipton Theorem

- Whether or not SAT has small circuits is formalized as "Is $\text{SAT} \in P_{/poly}$?".
- The answer is "NO", if the polynomial hierarchy does not collapse.

Theorem (Karp-Lipton Theorem)

If $NP \subseteq P_{/poly}$, then $PH = \Sigma_2^p$.

- We know that to show $PH = \Sigma_2^p$, it suffices to show $\Pi_2^p \subseteq \Sigma_2^p$.
  In particular, it suffices to show that Σ_2^p contains the Π_2^p-complete language $\Pi_2\text{SAT}$ consisting of all true formulas of the form

$$\forall u \in \{0,1\}^n \exists v \in \{0,1\}^n \varphi(u,v) = 1,$$

  where φ is an unquantified Boolean formula.

The Karp-Lipton Theorem (Cont'd)

- Suppose $\mathrm{NP} \subseteq \mathrm{P}_{/\mathrm{poly}}$.

  Then, there exists a polynomial p and a $p(n)$-sized circuit family $\{C_n\}_{n\in\mathbb{N}}$, such that, for every Boolean formula φ and $u \in \{0,1\}^n$,

  $$C_n(\varphi, u) = 1 \quad \text{iff} \quad \text{there exists } v \in \{0,1\}^n, \text{ such that } \varphi(u, v) = 1.$$

  Thus, the circuit solves the decision problem for $\textsc{Sat}$.

  However, we have an algorithm that converts any decision algorithm for $\textsc{Sat}$ into an algorithm that actually outputs a satisfying assignment whenever one exists.

  Think of this algorithm as a circuit.

  We obtain from the family $\{C_n\}$ a $q(n)$-sized circuit family $\{C'_n\}_{n\in\mathbb{N}}$, where q is a polynomial, such that, for every such formula φ and $u \in \{0,1\}^n$, if there is a string $v \in \{0,1\}^n$, such that $\varphi(u, v) = 1$, then $C'_n(\varphi, u)$ outputs such a string v.

The Karp-Lipton Theorem (Cont'd)

- The assumption $NP \subseteq P_{/poly}$ only implies the existence of C and C'.

  The main idea of the proof is that C' can be "guessed" using $\exists$ quantification.

  Since the circuit outputs a satisfying assignment if one exists, this answer can be checked directly.

  C'_n can be described using $10q(n)^2$ bits.

  So if $\forall u \in \{0,1\}^n \exists v \in \{0,1\}^n \varphi(u,v) = 1$ holds, then the following quantified formula is true:

$$\exists w \in \{0,1\}^{10q(n)^2} \forall u \in \{0,1\}^n$$
$$(w \text{ describes a circuit } C' \text{ and } \varphi(u, C'(\varphi, u)) = 1).$$

The Karp-Lipton Theorem (Cont'd)

- If $\forall u \in \{0,1\}^n \exists v \in \{0,1\}^n \varphi(u,v) = 1$ holds, then the following quantified formula is true:

$$\exists w \in \{0,1\}^{10q(n)^2} \forall u \in \{0,1\}^n$$
$$(w \text{ describes a circuit } C' \text{ and } \varphi(u, C'(\varphi, u)) = 1).$$

If $\forall u \in \{0,1\}^n \exists v \in \{0,1\}^n \varphi(u,v) = 1$ is false, then, for some u, no v exists such that $\varphi(u,v) = 1$.

Hence the preceding formula is false as well.

Evaluating a circuit on an input can be done deterministically in polynomial time.

So the truth of the displayed formula can be verified in Σ_2^p.

Meyer's Theorem

- Similarly, $P_{/poly}$ is unlikely to contain EXP.

Theorem (Meyer's Theorem)

If $EXP \subseteq P_{/poly}$, then $EXP = \Sigma_2^p$.

- Let $L \in EXP$.

  Then L is computable by a $2^{p(n)}$-time oblivious TM M, where p is some polynomial.

  Let $x \in \{0,1\}^n$ be some input string.

  For every $i \in [2^{p(n)}]$, let z_i be the encoding of the i-th snapshot of M's execution on input x.

Meyer's Theorem (Cont'd)

- Suppose M has k tapes.

  Then $x \in L$ if and only if, for every $k+1$ indices $i, i_1, \ldots, i_k$, the snapshots $z_i, z_{i_1}, \ldots, z_{i_k}$ satisfy some easily checkable criteria:
  - If z_i is the last snapshot, then it should encode M outputting 1;
  - if $i_1, \ldots, i_k$ are the last indices where M's heads were in the same locations as in i, then the values read in z_i should be consistent with the input and the values written in $z_{i_1}, \ldots, z_{i_k}$.

  These indices can be represented in polynomial time.

  If $\mathrm{EXP} \subseteq \mathrm{P}_{/\mathrm{poly}}$, then there is a $q(n)$-sized circuit C, for some polynomial q, that computes z_i from i.

Meyer's Theorem (Cont'd)

- Now the main point is that the correctness of the transcript implicitly computed by this circuit can be expressed as a coNP predicate.

  The predicate checks that the transcript satisfies all local criteria.

  Hence, $x \in L$ iff the following condition is true

$$\exists C \in \{0,1\}^{q(n)} \forall i, i_1, \ldots, i_k \in \{0,1\}^{p(n)} \\ T(x, C(i), C(i_1), \ldots, C(i_k)) = 1,$$

  where T is some polynomial-time TM checking these conditions.

  This implies that $L \in \Sigma_2^p$.

A Consequence

Corollary

If $P = NP$, then $EXP \not\subseteq P_{/poly}$.

- We know that, if $P = NP$, then $P = \Sigma_2^p$.

  Suppose $EXP \subseteq P_{/poly}$.

  By Meyer's Theorem, $EXP = \Sigma_2^p$.

  Then, we get $P = EXP$.

  This contradicts the Time Hierarchy Theorem.
- Thus, upper bounds (e.g., $NP \subseteq P$) can potentially be used to prove circuit lower bounds.

Subsection 5

Circuit Lower Bounds

The Circuit Approach to P vs. NP

- Since $\mathsf{P} \subseteq \mathsf{P}_{/\mathsf{poly}}$, if we ever prove $\mathsf{NP} \not\subseteq \mathsf{P}_{/\mathsf{poly}}$, then we will have shown $\mathsf{P} \neq \mathsf{NP}$.
- The Karp-Lipton Theorem gives evidence that $\mathsf{NP} \not\subseteq \mathsf{P}_{/\mathsf{poly}}$.
- There is reason to invest hope in resolving the P versus NP by proving $\mathsf{NP} \not\subseteq \mathsf{P}_{/\mathsf{poly}}$.
  - By representing computation using circuits, we seem to actually peer into the guts of it rather than treating it as a black box.
  - Thus, the approach may allow bypassing the limitations of relativizing methods.
- In fact, it is easy to show that some functions do require very large circuits to compute.

Existence of Hard Functions

Theorem (Existence of Hard Functions)

For every $n > 1$, there exists a function $f : \{0,1\}^n \to \{0,1\}$ that cannot be computed by a circuit C of size $\frac{2^n}{10n}$.

- The proof uses a simple counting argument:
  - The number of functions from $\{0,1\}^n$ to $\{0,1\}$ is 2^{2^n}.
  - Every circuit of size at most S can be represented as a string of $9S \log S$ bits, e.g., using the adjacency list representation.
    So the number of such circuits is at most $2^{9S \log S}$.

Let $S = \frac{2^n}{10n}$.

Then the number of circuits of size S is

$$\leq 2^{9S \log S} = 2^{9 \frac{2^n}{10n} \log \frac{2^n}{10n}} \leq 2^{\frac{2^n 9n}{10n}} < 2^{2^n}.$$

Existence of Hard Functions (Cont'd)

- It follows that the number of functions computed by such circuits is smaller than 2^{2^n}.

  This implies that there exists a function that is not computed by circuits of that size.
- With a more careful calculation, one can obtain a bound of

$$(1-\epsilon)\frac{2^n}{n}, \quad \text{for every } \epsilon > 0.$$

- We can even get a bound of

$$2^n\left(1+\frac{\log n}{n}-O\left(\frac{1}{n}\right)\right).$$

Another Proof Involving Probability

- Suppose that we pick a function $f : \{0,1\}^n \to \{0,1\}$ at random.

  We do this by picking, for every one of the 2^n possible inputs $x \in \{0,1\}^n$, the value $f(x)$ in $\{0,1\}$ uniformly and independently.

  Then, for every fixed circuit C and input x, the probability that $C(x) = f(x)$ is $\frac{1}{2}$.

  The choices made are independent.

  So the probability that C computes f, i.e., that

$$C(x) = f(x), \quad \text{for every } x \in \{0,1\}^n,$$

  is 2^{-2^n}.

  There are at most $2^{0.9 \cdot 2^n}$ circuits of size at most $\frac{2^n}{10n}$.

Another Proof Involving Probability (Cont'd)

- There are at most $2^{0.9 \cdot 2^n}$ circuits of size at most $\frac{2^n}{10n}$.

  By the probabilistic union bound, the probability that there exists such a circuit C computing f is at most

$$\frac{2^{0.9 \cdot 2^n}}{2^{2^n}} = 2^{-0.1 \cdot 2^n}.$$

  This is a number that tends very fast to zero as n grows.

  In particular, this number is smaller than one.

  This implies that there exists a function f that is not computed by any circuit of size at most $\frac{2^n}{10n}$.

Discussion on P vs. NP

- The proof technique of showing an object with a particular property exists by showing a random object satisfies this property with nonzero probability, is called the **probabilistic method**.
- The probabilistic proof yields a stronger statement than that in the theorem.
- Not only does there exist a hard function, but in fact the vast majority of the functions from $\{0,1\}^n$ to $\{0,1\}$ are hard.
- This gives hope that we should be able to find one such function that also happens to lie in NP, thus proving $\mathrm{NP} \not\subseteq \mathrm{P}_{/\mathrm{poly}}$.
- Unfortunately, after two decades, the best circuit size lower bound for an NP language is only $(5 - o(1))n$.
- On the positive side, we have had notable success in proving lower bounds for more restricted circuit models.

Subsection 6

Nonuniform Hierarchy Theorem

Nonuniform Hierarchy Theorem

- Like TMs and NDTM's, Boolean circuits have a hierarchy theorem.

Theorem (Nonuniform Hierarchy Theorem)

For every functions $T, T' : \mathbb{N} \to \mathbb{N}$, with $\frac{2^n}{n} > T'(n) > 10\,T(n) > n$,

$$\mathrm{SIZE}(T(n)) \subsetneq \mathrm{SIZE}(T'(n)).$$

- The diagonalization methods do not seem to apply in this setting.
- On the other hand, a counting argument works.
- Here, we prove that $\mathrm{SIZE}(n) \subsetneq \mathrm{SIZE}(n^2)$.

  We know that, for every ℓ, there is a function $f : \{0,1\}^\ell \to \{0,1\}$ that is not computable by $\frac{2^\ell}{10\ell}$-sized circuits.

Nonuniform Hierarchy Theorem (Cont'd)

- On the other hand, every function from $\{0,1\}^\ell$ to $\{0,1\}$ is computable by a $2^\ell 10\ell$-sized circuit.

  Set $\ell = 1.1 \log n$.

  Let

  $$g : \{0,1\}^n \to \{0,1\}$$

  be the function that applies f on the first ℓ bits of its input.

  Then

  $$g \in \mathsf{SIZE}(2^\ell 10\ell) = \mathsf{SIZE}(11 n^{1.1} \log n) \subseteq \mathsf{SIZE}(n^2);$$

  $$g \notin \mathsf{SIZE}\left(\frac{2^\ell}{10\ell}\right) = \mathsf{SIZE}\left(\frac{n^{1.1}}{11 \log n}\right) \supseteq \mathsf{SIZE}(n).$$

Subsection 7

Finer Gradations Among Circuit Classes

Subclasses of $P_{/poly}$

- Some subclasses of $P_{/poly}$, are interesting for two reasons.
- Separating NP from these subclasses may give insight into how to separate NP from $P_{/poly}$.
- These subclasses correspond to interesting computational models in their own right.

Parallel Computation

- An interesting connection is to **massively parallel computers**.
- Simple microprocessors are linked using an **interconnection network** that allows them to send messages to each other.
- Usual networks, e.g., the **hypercube**, allow linking n processors, such that interprocessor communication is possible, assuming some upper bounds on the total load, in $O(\log n)$ steps.
- The processors compute in lock-step, e.g., to the ticks of a global clock.
- Each processor is assumed to do a small amount of computation in each step, e.g., an operation on $O(\log n)$ bits.
- Each processor has enough memory to:
  - Remember its own address in the interconnection network;
  - Write down the address of any other processor, and thus send messages to it.

Efficient Parallel Computation

- We say that a computational problem has an **efficient parallel algorithm** if it can be solved for inputs of size n using a parallel computer:
  - With $n^{O(1)}$ processors;
  - In time $\log^{O(1)} n$.

Example

- Suppose the input consists of two n bit numbers x, y.

  We wish to compute $x + y$ fast in parallel.
- The grade-school algorithm proceeds from the least significant bit and maintains a **carry bit**.

  The most significant bit is computed only after n steps.
- A better algorithm, called **carry lookahead**, assigns each bit position to a separate processor.

  It then uses interprocessor communication to propagate carry bits.

  It takes $\mathrm{O}(n)$ processors and $\mathrm{O}(\log n)$ time.

Parallel Computation and Specific Problems

- There are also efficient parallel algorithms for integer multiplication and division.
- Many matrix computations can be done efficiently in parallel.

  E.g. computing the product, rank, determinant and inverse.
- Some graph theoretic algorithms, such as shortest path and minimum spanning tree, also have fast parallel implementations.
- However well-known polynomial-time problems, such as maximum flows and linear programming are not known to have any good parallel implementations and are conjectured not to have any.

The class NC

- We relate parallel computation to circuits.
- The **depth** of a circuit is the length of the longest directed path from an input node to the output node.

Definition (The Class NC)

For every d, a language L is in NC^d if L can be decided by a family of circuits $\{C_n\}$, where C_n has:

- Size $\mathrm{poly}(n)$;
- Depth $O\left(\log^d n\right)$.

Finally, we define the class

$$\mathrm{NC} = \bigcup_{i \geq 1} \mathrm{NC}^i.$$

- Uniform NC could require the circuits to be logspace-uniform.

The class AC

Definition (The Class AC)

The class AC^i is defined similarly to NC^i except gates are allowed to have unbounded fan-in, i.e., the OR and AND gates can be applied to more than two bits. Finally, define the class

$$\mathrm{AC} = \bigcup_{i \geq 0} \mathrm{AC}^i.$$

- Note that unbounded, but poly(n) fan-in can be simulated using a tree of ORs/ANDs of depth $O(\log n)$.
- So we get
$$\mathrm{NC}^i \subseteq \mathrm{AC}^i \subseteq \mathrm{NC}^{i+1}.$$
- The inclusion is known to be strict for $i = 0$.
- NC^0 is very limited since the circuit's output depends upon a constant number of input bits, but AC^0 does not suffer from this limitation.

Example

- The language

$$\textsc{Parity} = \{x : x \text{ has an odd number of 1s}\}$$

  is in NC^1.

  The circuit computing it has the form of a binary tree.
  - The answer appears at the root;
  - The left subtree computes the parity of the first $\frac{|x|}{2}$ bits;
  - The right subtree computes the parity of the remaining bits.
  - The gate at the top computes the parity of these two bits.

  The recursion resulting from this description gives a circuit of depth $O(\log n)$.

  The circuit is also logspace-uniform.
- It can be shown that $\textsc{Parity}$ is not in AC^0.

NC and Parallel Algorithms

- NC characterizes the languages with efficient parallel algorithms.

Theorem (NC and Parallel Algorithms)

A language has efficient parallel algorithms iff it is in NC.

- Suppose a language $L \in \mathrm{NC}$.

  Then, it is decidable by a circuit family $\{C_n\}$ of:
  - Size $N = \mathrm{O}(n^c)$;
  - Depth $D = \mathrm{O}\left(\log^d n\right)$.

  Take a parallel computer with N nodes.

  Configure it to decide L as follows:
  - Compute a description of C_n.
  - Allocate the role of each circuit node to a distinct processor.
  - Each processor computes the output at its assigned node.
  - It then sends the resulting bit to every other circuit node that needs it.

NC and Parallel Algorithms (Cont'd)

- Assuming the interconnection network delivers all messages in $O(\log N)$ time, the total running time is $O\left(\log^{d+1} N\right)$.

  Incidentally, note that:
  - If the circuit is nonuniform, then so is this parallel algorithm.
  - If the circuit is logspace-uniform, then so is the parallel algorithm.

  Conversely, suppose L has an efficient parallel algorithm using, for inputs of size n, a parallel computer with:
  - $N = n^{O(1)}$ processors;
  - Time $D = \log^{O(1)} n$.

  We construct a circuit with $N \cdot D$ nodes arranged in D layers.

  The i-th node in the t-th layer performs the computation of processor i at time t.

  The network is replaced by the circuit wires.

Efficient Parallelization of P

- A major open question is whether every polynomial-time algorithm has an efficient parallel implementation, i.e., whether P = NC.
- We believe that the answer is NO.
- However, currently we are unable to even separate PH from NC^1.
- The theory of P-completeness can be used to study which problems are likely to be efficiently parallelizable (are in NC) and which are not.

P-Completeness

Definition (P-Complete Language)

A language is P-**complete** if:

- It is in P;
- Every language in P is logspace reducible to it.

- Recall that L denotes the class of languages decidable in logarithmic space.

Theorem

If language L is P-complete, then:

1. $L \in \mathsf{NC}$ iff $\mathsf{P} = \mathsf{NC}$.
2. $L \in \mathsf{L}$ iff $\mathsf{P} = \mathsf{L}$.

A P-Complete Language

- Define the language

$$\textsc{CircuitEval} = \{\langle C, x\rangle : C \text{ is an } n\text{-input single-output circuit and } x \in \{0,1\}^n \text{ is such that } C(x) = 1\}.$$

Theorem

$\textsc{CircuitEval}$ is P-complete.

- The language is clearly in P.

  For completeness, one also needs to showcase a logspace-reduction from any other language in P to this language.

  Such a reduction is implicit in the proof of the equivalence between having logspace uniform networks and being in P.

Subsection 8

Circuits of Exponential Size

On Uniform Circuit Families

- We saw that every language has circuits of size $O\left(\frac{2^n}{n}\right)$.
- Finding these circuits may be difficult or even undecidable.
- If we place a uniformity condition on the circuits, that is, require them to be efficiently computable, then the circuit complexity of some languages could potentially exceed 2^n.

DC Uniform Circuit Families

Definition (DC Uniform Circuit Family)

Let $\{C_n\}_{n\geq 1}$ be a circuit family. We say that it is a **Direct Connect uniform** (**DC uniform**) **family** if, there is a polynomial time algorithm that, given $\langle n, i\rangle$, can compute the i-th bit of (the adjacency matrix representation of) the circuit C_n.
More precisely, a family $\{C_n\}_{n\in\mathbb{N}}$ is DC uniform iff the functions SIZE, TYPE and EDGE are computable in polynomial time.

- Note that the circuits may have exponential size.
- However, they have a succinct representation in terms of a TM.
- The TM can systematically generate any required vertex of the circuit in polynomial time.

Characterization of PH in Terms of DC Uniform Circuits

- Now we give another characterization of the class PH in terms of computability by uniform circuit families of bounded depth.

Theorem (Characterization of PH)

$L \in \text{PH}$ iff L can be computed by a DC uniform circuit family $\{C_n\}$, that satisfies the following conditions:

1. It uses AND, OR, NOT gates.
2. It has size $2^{n^{O(1)}}$ and constant depth.
3. Its gates can have unbounded (exponential) fan-in.
4. Its NOT gates appear only at the input level (i.e., they are only applied directly to the input and not to the result of any other gate).

- We omit the proof.
- If we drop the restriction that the circuits have constant depth, then we obtain exactly EXP.