Complexity Theory
Circuit Complexity

Daniel Borchmann, Markus Krötzsch

Computational Logic

2016-01-13
Review
Computing with Circuits
Motivation

Some questions:

- What can complexity theory tell us about parallel computation?
- Are there any meaningful complexity classes below $\text{LOGSPACE}$? Do they contain relevant problems?
- Even if it is hard to find a universal algorithm for solving all instances of a problem, couldn’t it still be that there is a simple algorithm for every fixed problem size?
Motivation

Some questions:

- What can complexity theory tell us about parallel computation?
- Are there any meaningful complexity classes below $\text{LOGSPACE}$? Do they contain relevant problems?
- Even if it is hard to find a universal algorithm for solving all instances of a problem, couldn’t it still be that there is a simple algorithm for every fixed problem size?

$\leadsto$ circuit complexity provides some answers

Intuition: use circuits with logical gates to model computation
Boolean Circuits

Definition 17.1

A **Boolean circuit** is a finite, directed, acyclic graph where

- each node that has no predecessor is an input node
- each node that is not an input node is one of the following types of logical gate:
  - AND with two input wires
  - OR with two input wires
  - NOT with one input wire
- one or more nodes are designated output nodes

The outputs of a Boolean circuit are computed in the obvious way from the inputs.

\[ \sim \text{ circuits with } k \text{ inputs and } \ell \text{ outputs represent functions } \{0, 1\}^k \rightarrow \{0, 1\}^\ell \]

We often consider circuits with only one output.
Example 1

$$\text{XOR function:}$$

$$\overline{X}_1 \lor \overline{X}_2$$

2015 Daniel Borchmann, Markus Krötzsch

Complexity Theory

2016-01-13 #6
Example 1

XOR function:
Example 2

Parity function with four inputs:

\( x_1 \lor x_2 \lor x_3 \lor x_4 \)

(true for odd number of 1s)

2015 Daniel Borchmann, Markus Krötzsch
Example 2

Parity function with four inputs:
(true for odd number of 1s)
Alternative Ways of Viewing Circuits (1)

Propositional formulae

- propositional formulae are special circuits: each non-input node has only one outgoing wire
- each variable corresponds to one input node
- each logical operator corresponds to a gate
- each sub-formula corresponds to a wire

\( (((\neg x_1 \land x_2) \lor (x_1 \land \neg x_2)) \lor (x_1 \land \neg x_2)) \)
Alternative Ways of Viewing Circuits (2)

Straight-line programs

- are programs without loops and branching (if, goto, for, while, etc.)
- that only have Boolean variables
- and where each line can only be an assignment with a single Boolean operator

\[ n \text{-line programs correspond to } n \text{-gate circuits} \]

\[
\begin{align*}
01 & \quad z_1 := \neg x_1 \\
02 & \quad z_2 := \neg x_2 \\
03 & \quad z_3 := z_1 \land x_2 \\
04 & \quad z_4 := z_2 \land x_1 \\
05 & \quad \text{return } z_3 \lor z_4
\end{align*}
\]
Example: Generalised AND

The function that tests if all inputs are 1 can be encoded by combining binary AND gates:

- works similarly for OR gates
- number of gates: \( n - 1 \)
- we can use \( n \)-way AND and OR (keeping the real size in mind)
Solving Problems with Circuits

Circuits are not universal: fixed number of inputs!
How can they solve arbitrary problems?

Definition 17.2
A circuit family is an infinite list $C = C_1, C_2, C_3, \ldots$ where each $C_i$ is a Boolean circuit with $i$ inputs and one output. We say that $C$ decides a language $\mathcal{L}$ (over \{0, 1\}) if

$$w \in \mathcal{L} \quad \text{if and only if} \quad C_n(w) = 1 \text{ for } n = |w|.$$ 

Example 17.3
The circuits we gave for generalised AND are a circuit family that decides the language \{1^n \mid n \geq 1\}. 
Circuit Complexity

To measure difficulty of problems solved by circuits, we can count the number of gates needed:

**Definition 17.4**

The *size* of a circuit is its number of gates.

Let $f : \mathbb{N} \rightarrow \mathbb{R}^+$ be a function. A circuit family $C$ is *$f$-size bounded* if each of its circuits $C_n$ is of size at most $f(n)$.

$\text{SIZE}(f(n))$ is the class of all languages that can be decided by an $O(f(n))$-size bounded circuit family.

**Example 17.5**

Our circuits for generalised AND show that $\{1^n \mid n \geq 1\} \in \text{SIZE}(n)$. 
Examples

Many simple operations can be performed by circuits of polynomial size:

- Boolean functions such as parity (=sum modulo 2), sum modulo $n$, or majority
- Arithmetic operations such as addition, subtraction, multiplication, division (taking two fixed-arity binary numbers as inputs)
- Many matrix operations

See exercise for some more examples
Polynomial Circuits
A natural class of problems to consider are those that have polynomial circuit families:

**Definition 17.6**

\[ P_{/\text{poly}} = \bigcup_{d \geq 1} \text{SIZE}(n^d). \]

**Note:** A language is in \( P_{/\text{poly}} \) if it is solved by some polynomial-sized circuit family. There may not be a way to compute (or even finitely represent) this family.

How does \( P_{/\text{poly}} \) relate to other classes?
Quadratic Circuits for Deterministic Time

Theorem 17.7

For $f(n) \geq n$, we have $\text{DTIME}(f) \subseteq \text{SIZE}(f^2)$.

Proof sketch (see also Sipser, Theorem 9.30).

- We can represent the $\text{DTIME}$ computation as in the proof of Theorem 15.5: as a list of configurations encoded as words

  $\ast \sigma_1 \cdots \sigma_{i-1} \langle q, \sigma_i \rangle \sigma_{i+1} \cdots \sigma_m \ast$

  of symbols from the set $\Omega = \{\ast\} \cup \Gamma \cup (Q \times \Gamma)$. $\leadsto$ tableau with $O(f^2)$ cells.

- We can describe each cell with a list of bits (wires in a circuit).

- We can compute one configuration from its predecessor by $O(f)$ circuits (idea: compute the value of each cell from its three upper neighbours as in Theorem 15.5)

- Acceptance can be checked by assuming that the TM returns to a unique configuration position/state when accepting
From $\mathsf{DTIME}(f) \subseteq \mathsf{SIZE}(f^2)$ we get:

**Corollary 17.8**

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

This suggests another way of approaching the $\mathsf{P}$ vs. $\mathsf{NP}$ question:

If any language in $\mathsf{NP}$ is not in $\mathsf{P/poly}$, then $\mathsf{P} \neq \mathsf{NP}$.

(but nobody has found any such language yet)
**Circuit-Sat**

*Input:* A Boolean Circuit \( C \) with one output.

*Problem:* Is there any input for which \( C \) returns 1?

Theorem 17.9

Circuit-Sat is \( \text{NP-complete} \).
**Circuit-Sat**

*Input:* A Boolean Circuit $C$ with one output.

*Problem:* Is there any input for which $C$ returns 1?

**Theorem 17.9**

Circuit-Sat is $\text{NP}$-complete.

**Proof.**

Inclusion in $\text{NP}$ is easy (just guess the input).

For $\text{NP}$-hardness, we use that $\text{NP}$ problems are those with a $\text{P}$-verifier:

- The DTM simulation of Theorem 17.7 can be used to implement a verifier (input: $(w\#c)$ in binary)
- We can hard-wire the $w$-inputs to use a fixed word instead (remaining inputs: $c$)
- The circuit is satisfiable iff there is a certificate for which the verifier accepts $w$
Theorem 17.10

$3\text{Sat}$ is NP-complete.
A New Proof for Cook-Levin

Theorem 17.10

3SAT is \textit{NP}-complete.

Proof.

Membership in \textit{NP} is again easy (as before).

For \textit{NP}-hardness, we express the circuit that was used to implement the verifier in Theorem 17.9 as propositional logic formula in 3-CNF:

- Create a propositional variable \( X \) for every wire in the circuit
- Add clauses to relate input wires to output wires, e.g., for AND gate with inputs \( X_1 \) and \( X_2 \) and output \( X_3 \), we encode \((X_1 \land X_2) \leftrightarrow X_3\) as:

\[
(\neg X_1 \lor \neg X_2 \lor X_3) \land (X_1 \lor \neg X_3) \land (X_2 \lor \neg X_3)
\]

- Fixed number of clauses per gate = linear size increase
- Add a clause \((X)\) for the output wire \( X \).
Is $\text{P} = \text{P/poly}$?

We showed $\text{P} \subseteq \text{P/poly}$. Does the converse also hold?
Is $P = P/poly$?

We showed $P \subseteq P/poly$. Does the converse also hold?

No!

**Theorem 17.11**

$P/poly$ contains undecidable problems.

**Proof.**

We define the unary Halting problem as the (undecidable) language:

$$U_{\text{HALT}} := \{1^n \mid \text{the binary encoding of } n \text{ encodes a pair } \langle M, w \rangle \text{ where } M \text{ is a TM that halts on word } w\}$$

For a number $1^n \in U_{\text{HALT}}$, let $C_n$ be the circuit that computes a generalised AND of all inputs. For all other numbers, let $C_n$ be a circuit that always returns 0. The circuit family $C_1, C_2, C_3, \ldots$ accepts $U_{\text{HALT}}$.  

Uniform Circuit Families

$\mathsf{P/poly}$ too powerful, since we do not require the circuits to be computable. We can add this:

**Definition 17.12**

A circuit family $C_1, C_2, C_3, \ldots$ is **log-space-uniform** if there is a log-space computable function that maps words $1^n$ to (an encoding of) $C_n$. (We could also define similar notions of uniformity for other complexity classes.)
Uniform Circuit Families

$P_{\text{poly}}$ too powerful, since we do not require the circuits to be computable. We can add this:

**Definition 17.12**

A circuit family $C_1, C_2, C_3, \ldots$ is **log-space-uniform** if there is a log-space computable function that maps words $1^n$ to (an encoding of) $C_n$. (We could also define similar notions of uniformity for other complexity classes.)

**Theorem 17.13**

The class of all languages that are accepted by a log-space-uniform circuit family of polynomial size is exactly $P$.

**Proof sketch.**

A detailed analysis shows that out that our earlier reduction of $P$ DTMs to circuits is log-space-uniform. Conversely, a polynomial-time procedure can be obtained by first computing a suitable circuit (in log-space) and then evaluating it (in polynomial time). □
One can also describe $P_{/\text{poly}}$ using TMs that take “advice”:

**Definition 17.14**

Consider a function $a : \mathbb{N} \rightarrow \mathbb{N}$. A language $L$ is accepted by a Turing Machine $M$ with $a$ bits of advice if there is a sequence of advice strings $\alpha_0, \alpha_1, \alpha_2, \ldots$ of length $|\alpha_i| = a(i)$ and $M$ accepts inputs of the form $(w\#a_{|w|})$ if and only if $w \in L$.

$P_{/\text{poly}}$ is equivalent to the class of problems that can be solved by a $P_{\text{TIME}}$ TM that takes a polynomial amount of “advice”.

(This is where the notation $P_{/\text{poly}}$ comes from.)
We showed $P \subseteq P_{/\text{poly}}$. Does $NP \subseteq P_{/\text{poly}}$ also hold?
We showed $P \subseteq P_{/\text{poly}}$. Does $NP \subseteq P_{/\text{poly}}$ also hold?

Nobody knows

**Theorem 17.15 (Karp-Lipton Theorem)**

> If $NP \subseteq P_{/\text{poly}}$ then $PH = \Sigma^P_2$.

**Proof sketch (see Arora/Barak Theorem 6.19).**

- if $NP \subseteq P_{/\text{poly}}$ then there is a polysize circuit family solving $\text{Sat}$
- Using this, one can argue that there is also a polysize circuit family that computes the lexicographically first" satisfying assignment ($k$ output bits for $k$ variables)
- A $\Pi^P_2$-QBF formula $\forall X.\exists Y.\varphi$ is true if, for all values of $X$, $\varphi[X]$ is satisfiable.
- In $\Sigma^P_2$, we can: (1) guess the polysize circuit for SAT, (2) check for all values of $X$ if its output is really a satisfying assignment (to verify the guess)
- This solves $\Pi^P_2$-hard problems in $\Sigma^P_2$
- But then the Polynomial Hierarchy collapses at $\Sigma^P_2$, as claimed.
**P/poly and EXPTIME**

We showed $P \subseteq P/poly$. Does $\text{ExpTime} \subseteq P/poly$ also hold?
We showed $P \subseteq P_{/\text{poly}}$. Does $\text{ExpTime} \subseteq P_{/\text{poly}}$ also hold? Nobody knows.

**Theorem 17.16 (Meyer’s Theorem)**

*If* $\text{ExpTime} \subseteq P_{/\text{poly}}$ *then* $\text{ExpTime} = PH = \Sigma^p_2$.

See [Arora/Barak, Theorem 6.20] for a proof sketch.

**Corollary 17.17**

*If* $\text{ExpTime} \subseteq P_{/\text{poly}}$ *then* $P \neq NP$.

**Proof.**

*If* $\text{ExpTime} \subseteq P_{/\text{poly}}$ *then* $\text{ExpTime} = \Sigma^p_2$ (Meyer’s Theorem).

By the Time Hierarchy Theorem, $P \neq \text{ExpTime}$, so $P \neq \Sigma^p_2$.

So the Polynomial Hierarchy doesn’t collapse completely, and $P \neq NP$. □
How Big a Circuit Could We Need?

We should not be surprised that $\mathbb{P}/\text{poly}$ is so powerful: exponential circuit families are already enough to accept any language

Exercise: show that every Boolean function over $n$ variables can be expressed by a circuit of size $\leq n2^n$.

It turns out that these exponential circuits are really needed:

**Theorem 17.18 (Shannon 1949 (!))**

*For every $n$, there is a function $\{0, 1\}^n \rightarrow \{0, 1\}$ that cannot be computed by any circuit of size $2^n/(10n)$.***

In fact, one can even show: *almost every Boolean function requires circuits of size $> 2^n/(10n)$ — and is therefore not in $\mathbb{P}/\text{poly}$*

Is any of these functions in $\mathbb{NP}$? Or at least in $\mathbb{EXP}$? Or at least in $\mathbb{NEXP}$?
How Big a Circuit Could We Need?

We should not be surprised that $P_{\text{poly}}$ is so powerful: exponential circuit families are already enough to accept any language.

Exercise: show that every Boolean function over $n$ variables can be expressed by a circuit of size $\leq n2^n$.

It turns out that these exponential circuits are really needed:

**Theorem 17.18 (Shannon 1949 (!))**

For every $n$, there is a function $\{0, 1\}^n \rightarrow \{0, 1\}$ that cannot be computed by any circuit of size $2^n/(10n)$.

In fact, one can even show: almost every Boolean function requires circuits of size $> 2^n/(10n)$ – and is therefore not in $P_{\text{poly}}$

Is any of these functions in $\text{NP}$? Or at least in $\text{Exp}$? Or at least in $\text{NExp}$? Nobody knows.