More about the Polynomial Hierarchy

We discovered a hierarchy of complexity classes between P and PSpace, with NP and coNP on the first level, and infinitely many further levels above:

**Definition by ATM:** Classes $\Sigma^p_i / \Pi^p_i$ are defined by polytime ATMs with bounded types of alternation, starting computation with existential/universal states.

**Definition by Verifier:** Classes $\Sigma^p_i / \Pi^p_i$ are given as projections of certain verifier languages in P, requiring existence/universality of polynomial witnesses.

**Definition by Oracle:** Classes $\Sigma^p_i / \Pi^p_i$ are defined as languages of NP/coNP oracle TMs with $\Sigma^p_{i-1}$ (or, equivalently, $\Pi^p_{i-1}$) oracle.

Using such oracles with deterministic TMs, we can also define classes $\Delta^p_i$.

More Classes in PH

We defined $\Sigma^p_i$ and $\Pi^p_i$ by relativising NP and coNP with oracles. What happens if we start from P instead?

**Definition 18.1:** $\Delta^p_0 := P$ and $\Delta^p_{k+1} := \Sigma^p_k$.

Some immediate observations:
- $\Delta^p_0 = P$
- $\Delta^p_1 = P_{NC}$
- $\Delta^p_2 \subseteq \Sigma^p_1$ (since $P \subseteq NP$) and $\Delta^p_2 \subseteq \Pi^p_1$ (since $P \subseteq coNP$)
- $\Sigma^p_1 \subseteq \Delta^p_{k+1}$ and $\Pi^p_1 \subseteq \Delta^p_{k+1}$
Problems for $\Delta^p_k$?

$\Delta^p_k$ seems to be less common in practice, but there are some known complete problems for $P^{NP} = \Delta^p_1$:

- **Uniquely Optimal TSP [Papadimitriou, JACM 1984]**
  - **Input:** Undirected graph $G$ with edge weights (distances).
  - **Problem:** Is there exactly one shortest travelling salesman tour on $G$?

- **Divisible TSP [Krentel, JCSS 1988]**
  - **Input:** Undirected graph $G$ with edge weights; number $k$.
  - **Problem:** Is the shortest travelling salesman tour on $G$ divisible by $k$?

- **Odd Final SAT [Krentel, JCSS 1988]**
  - **Input:** Propositional formula $\varphi$ with $n$ variables.
  - **Problem:** Is $X_n$ true in the lexicographically last assignment satisfying $\varphi$?

Is the Polynomial Hierarchy Real?

Questions:

- Are all of these classes really distinct? Nobody knows.
- Are any of these classes really distinct? Nobody knows.
- Are any of these classes distinct from $P$? Nobody knows.
- Are any of these classes distinct from $PSpace$? Nobody knows.

What do we know then?

What We Know (Excerpt)

**Theorem 18.2:** If there is any $k$ such that $\Sigma^p_k = \Sigma^p_{k+1}$ then $\Sigma^p_j = \Pi^p_j = \Sigma^p_k$ for all $j > k$, and therefore $PH = \Sigma^p_k$.

In this case, we say that the polynomial hierarchy collapses at level $k$.

**Proof:** Left as exercise (not too hard to get from definitions).

**Corollary 18.3:** If $PH \neq P$ then $NP \neq P$.

Intuitively speaking: “The polynomial hierarchy is built upon the assumption that $NP$ has some additional power over $P$. If this is not the case, the whole hierarchy collapses.”

**Theorem 18.4:** $PH \subseteq PSpace$.

**Proof:** Left as exercise (induction over $PH$ levels, using that $PSPACE = PSPACE$).

**Theorem 18.5:** If $PH = PSpace$ then there is some $k$ with $PH = \Sigma^p_k$.

**Proof:** If $PH = PSpace$ then $\text{True QBF} \in PH$. Hence $\text{True QBF} \in \Sigma^p_k$ for some $k$. Since $\text{True QBF}$ is $PSpace$-hard, this implies $\Sigma^p_k = PSpace$. 

---

Markus Krötzsch, 17th Dec 2018 Complexity Theory slide 5 of 27

Markus Krötzsch, 17th Dec 2018 Complexity Theory slide 6 of 27

Markus Krötzsch, 17th Dec 2018 Complexity Theory slide 7 of 27

Markus Krötzsch, 17th Dec 2018 Complexity Theory slide 8 of 27
What We Believe (Excerpt)

“Most experts” think that:
• The polynomial hierarchy does not collapse completely (same as $P \neq NP$)
• The polynomial hierarchy does not collapse on any level
  (in particular $PH \neq PSPACE$ and there is no $PH$-complete problem)

But there can always be surprises . . .

Motivation

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

Some questions:
• 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?
• What can complexity theory tell us about parallel computation?
• Are there any meaningful complexity classes below LogSpace? Do they contain relevant problems?

$\rightarrow$ circuit complexity provides some answers

Intuition: use circuits with logical gates to model computation

Boolean Circuits

Definition 18.6: 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. $\rightarrow$ circuits with $k$ inputs and $\ell$ outputs represent functions $\{0, 1\}^k \rightarrow \{0, 1\}^\ell$

We often consider circuits with only one output.
Example 1

**XOR function:**

```
  A
 /\  \\
V  V
  A
  x1  x2
```

Example 2

**Parity function with four inputs:**
(true for odd number of 1s)

```
  A
 /\  \\
V  V
  A
  x1  x2  x3  x4
```

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))
```

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

```
1 z_1 := \neg x_1
2 z_2 := \neg x_2
3 z_3 := z_1 \land x_2
4 z_4 := z_2 \land x_1
5 \text{return } z_3 \lor z_4
```
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)

... (\( n/4 \) gates)

... (\( n/2 \) gates)

\[ x_1, x_2, x_3, x_4, x_5, \ldots, x_n \]

Solving Problems with Circuits

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

**Definition 18.7:** 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 \( L \) (over \( \{0, 1\} \)) if \( w \in L \) if and only if \( C_n(w) = 1 \) for \( n = |w| \).

**Example 18.8:** The circuits we gave for generalised AND are a circuit family that decides the language \( \{1^n | n \geq 1\} \).

Circuit Complexity

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

**Definition 18.9:** The size of a circuit is its number of gates.

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

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

**Example 18.10:** Our circuits for generalised AND show that \( \{1^n | n \geq 1\} \in \text{Size}(n) \).

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.
A natural class of problems to consider are those that have polynomial circuit families:

**Definition 18.11:** \( P/poly = \bigcup_{d \geq 1} \text{Size}(n^d) \).

**Note:** A language is in \( P/poly \) 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 \( P/poly \) relate to other classes?

**Theorem 18.12:** 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 DTime computation as in the proof of Theorem 16.10: as a list of configurations encoded as words

  \( \ast \, \sigma_1 \, \ldots \, \sigma_{r-1} \, (q, \sigma_r) \, \sigma_{r+1} \, \ldots \, \sigma_m \ast \)

  of symbols from the set \( \Omega = \{ \ast \} \cup \Gamma \cup (Q \times \Gamma) \).

- Tableau (i.e., grid) 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 16.10)

- Acceptance can be checked by assuming that the TM returns to a unique configuration position/state when accepting

**From Polynomial Time to Polynomial Size**

From \( \text{DTime}(f) \subseteq \text{Size}(f^2) \) we get:

**Corollary 18.13:** \( P \subseteq P/poly \).

This suggests another way of approaching the P vs. NP question:

If any language in NP is not in \( P/poly \), then \( P \neq NP \).

(but nobody has found any such language yet)
**Theorem 18.14:** \textsc{Circuit-Sat} is NP-complete.

**Proof:** Inclusion in NP is easy (just guess the input).

For NP-hardness, we use that NP problems are those with a P-verifier:

- The DTM simulation of Theorem 18.12 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 \(w\)

**Note:** It would also be easy to reduce \textsc{Sat} to \textsc{Circuit-Sat}, but the above yields a proof from first principles.

---

**A New Proof for Cook-Levin**

**Theorem 18.15:** \textsc{3Sat} is NP-complete.

**Proof:** Membership in NP is again easy (as before).

For NP-hardness, we express the circuit that was used to implement the verifier in Theorem 18.14 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 = constant factor size increase
- Add a clause \((X)\) for the output wire \(X\)

---

**Summary and Outlook**

We do not know if the Polynomial Hierarchy is real or collapses

Circuits provide an alternative model of computation

\[ P \subseteq P^{\text{poly}} \]

\textbf{What's next?}

- Circuits for parallelism
- Complexity classes (strictly!) below P
- Randomness