On P vs NP

The map of NP
- Isomorphism
- Density
- Oracle Turing machines
- Monotonic circuits

(C. Papadimitriou: Computational Complexity, Chapter 14)

The Map of NP

NP-completeness provides a powerful tool for classifying challenging computational problems.

However, there are problems not known to be in P or NP-complete, such as GRAPH ISOMORPHISM.

Are there problems in NP that are neither in P nor NP-complete, i.e., which is the case:

<table>
<thead>
<tr>
<th>NP</th>
<th>NP</th>
<th>NP = P</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP-complete</td>
<td>NP-complete</td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>P</td>
<td></td>
</tr>
</tbody>
</table>

The middle alternative is not possible.

Theorem. If P ≠ NP, then there is a language in NP which is neither in P nor NP-complete.

Isomorphism

- All known NP-complete languages are polynomially isomorphic.
- Languages K, L are polynomially isomorphic if there is a function h : Σ* → Σ* such that
  1. h is bijection;
  2. for each x ∈ Σ*, x ∈ K iff h(x) ∈ L;
  3. h and h⁻¹ are polynomial-time computable.
- There is a polynomial-time mapping from any NP-complete problem to any other NP-complete problem (a logarithmic space reduction).
- However, a reduction is not necessarily a polynomial isomorphism (not a bijection) but with a padding function an isomorphism is obtained.

Isomorphism—cont’d

- A function pad : (Σ*)² → Σ* is a padding function for L if
  1. pad is computable in logarithmic space;
  2. for each x, y ∈ Σ*, pad(x, y) ∈ L iff x ∈ L;
  3. for each x, y ∈ Σ*, |pad(x, y)| > |x| + |y|;
  4. there is a logarithmic-space algorithm which given pad(x, y) recovers y.

Example. A padding function for SAT:

Given x (a conjunction of m clauses with n variables) and a binary string y, pad(x, y) is all clauses of x together with m + |y| new clauses and |y| + 1 more variables where the m new clauses are copies of the clause \( u_{n+1} \) and where the \( m + i \)th new clause is either \( \neg u_{n+i+1} \) or \( u_{n+i+1} \) depending on whether the \( i \)th symbol in \( y \) is 0 or 1.
Isomorphism—cont’d

➤ If \( R \) is a reduction from \( K \) to \( L \) and \( \text{pad} \) is a padding function for \( L \), \( \text{pad}(R(x), x) \) is a length-increasing one-to-one reduction. Furthermore, there is a logarithmic-space algorithm for the inverse of \( \text{pad}(R(x), x) \).
➤ If there is a reduction from \( K \) to \( L \) and a reduction from \( L \) to \( K \) and the reductions are length-increasing, one-to-one and logarithmic-space invertible, then \( K, L \) are polynomially isomorphic.
➤ Corollary: The following \( \text{NP} \)-languages are polynomially isomorphic: SAT, HAMILTON PATH, CLIQUE, MAX CUT, TRIPARTITE MATCHING, KNAPSACK, . . .

Density—cont’d

➤ Examples:
- Any unary language (\( \subseteq \{0\}^* \)) is sparse (\( \text{dens}_L(n) \leq n \)).
- All \( \text{NP} \)-complete language seen this far are dense.

**Theorem.** If a unary language \( U \subseteq \{0\}^* \) is \( \text{NP} \)-complete, then \( \text{P} = \text{NP} \).

**Theorem.** If a sparse language is \( \text{NP} \)-complete, then \( \text{P} = \text{NP} \).

Oracle Turing Machines

➤ The idea: study complexity in a setting where a part of the computation comes “for free”.
➤ Can be used for exploring conjectures (like \( \text{P} \neq \text{NP} \)) in an alternative setting.
➤ Can isolate orthogonal (independent) sources of complexity.

Density

Density of a language \( L \):

\[
\text{dens}_L(n) = |\{x \in L : |x| \leq n\}|
\]

**Proposition.** If \( K, L \subseteq \Sigma^* \) are polynomially isomorphic, then \( \text{dens}_K \) and \( \text{dens}_L \) are polynomially related (\( \text{dens}_L(n) \leq \text{dens}_K(p(n)) \)).
➤ A sparse language: polynomially bounded density function
➤ A dense language: superpolynomial density function
An oracle Turing machine $M^A$:

- New elements: query string, query state $q^?$, answer states $q_{\text{YES}}$, $q_{\text{NO}}$
- From the query state $q^?$ the machine moves to $q_{\text{YES}}$ or to $q_{\text{NO}}$ depending on whether $y \in A$ holds or not where $y$ is the content of the query string and $A$ the oracle set.
- Note that a query is performed in one step!
- Computation of $M^A$ with oracle $A$ on input $x$: $M^A(x)$.

Theorem. There is an oracle $A$ for which $P^A = NP^A$.

Proof: Let $A$ be $PSPACE$-complete. Then $PSPACE \subseteq P^A \subseteq NP^A \subseteq NPSPACE \subseteq PSPACE$

Monotone Circuits

Conjecture B. NP-complete problems have no polynomial circuits, uniform or not.

This implies $P \neq NP$ by Theorem 11.5 (A language $L$ has uniformly polynomial circuits iff $L \in P$).

- Lower bounds on the size of circuits for families of functions are hard to establish.
- Consider a weaker circuit model: monotone circuits (ones without NOT gates).
- Monotone circuits can only compute monotone functions: output cannot changed from true to false by changing input from false to true.

Theorem. (Razborov’s Theorem): There is a constant $c$ such that for large enough $n$ all monotone circuits for $\text{CLIQUE}_{n,k}$ with $k = n^{1/4}$ have size at least $2^{cn^{1/8}}$.

To show $P \neq NP$ it would be sufficient to establish that all monotone languages in $P$ have polynomial monotone circuits.

However, this does not hold (e.g., for MATCHING).
Learning Objectives

➤ The concepts of polynomial isomorphism and density of a language
➤ The concept of an oracle Turing machine
➤ The role of monotonic circuits in studying the $P \overset{?}{=} \text{NP}$ question.