Computational Complexity Theory

Marcus Hutter
Canberra, ACT, 0200, Australia
http://www.hutter1.net/
- Turing Machine (TM)
- Deterministic Turing Machine (DTM)
- NonDeterministic Turing Machine (NTM)

**Formal Languages** \((L) \cong\) **Boolean Function** \((f)\):

A TM \(M\) decides language \(L \subseteq \{0, 1\}\) if it computes function \(f : \{0, 1\}^* \to \{0, 1\}\), where \(f(x) := 1 \iff x \in L\).
Polynomial Time Classes

- **The class TIME:** Let $T : \mathbb{N} \rightarrow \mathbb{N}$ be some function and $L \subseteq \{0, 1\}^*$. 
  $L \in \text{TIME}(T(n)) : \iff \exists \text{TM that runs in time } O(T(n))$ on all inputs of length $n$ and decides $L$.

- Write $\text{DTIME}$ for DTM, and $\text{NTIME}$ for NTM.

- **The class P**: $= \bigcup_{k=1}^{\infty} \text{DTIME}(n^k) = \text{poly-time} = \text{“tractable”}$
  $= \text{problems that can be decided in polynomial time.}$
  Examples: is-sorted, is-shortest path, is-prime (AKS 2002), ...

- **The class NP**: $= \bigcup_{k=1}^{\infty} \text{NTIME}(n^k)$
  $= \text{solution can be verified in poly-time:}$
  Examples: SAT, Independent Set, ...

- **FTIME, FP, FNP**: Same as TIME, P, NP but for fcts. $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$

- **Poly-time Reduction** $L \leq_p L' : \iff \exists f \in \text{FP}: \ x \in L \iff f(x) \in L'$. 

NP-Completeness

- \( L' \) is said to be **NP-hard** if \( L \leq_p L' \) for all \( L \in \text{NP} \).

- \( L' \) is said to be **NP-complete** if \( L \) is NP-hard and \( L' \in \text{NP} \).

There are literally **thousands of NP-complete problems**: SAT, 3SAT, IndSet, Knapsack, TSP, Clique, 3Col, MaxCut, ...

**Elementary results:**

- Transitivity: If \( L \leq_p L' \) and \( L' \leq_p L'' \) then \( L \leq_p L'' \)

- If \( L \) is NP-hard and \( L \in \text{P} \), then \( \text{P} = \text{NP} \)

- If \( L \) is NP-complete, then \( L \in \text{P} \) iff \( \text{P} = \text{NP} \)

**Implication:** If we could find a poly-time algorithm for a single NP-complete problem, we could solve them all in poly-time!
The InFamous $P=NP$ Question

Informally: Is it harder to find a solution than to verify a solution?

- The standard opinion is that $P \neq NP$.
- Is the deepest open problem in all of mathematics.
- No one has any idea where to begin.

We have a pretty sophisticated idea of why we have no idea:
(keywords: oracles and natural proofs)

- $P^{\text{EXP}} = \text{NP}^{\text{EXP}}$ with access to EXP-oracle.
- $P^{\text{rand}} \neq \text{NP}^{\text{rand}}$ with access to random oracle.

$\Rightarrow$ No proof that relativizes, i.e. continues to hold with oracles, can decide $P=NP$.

- Most proofs in complexity theory relativize.

$\Rightarrow$ $P=NP$ cannot be decided by such proofs.
The class \( \text{SPACE} \): Let \( S : \mathbb{N} \rightarrow \mathbb{N} \) be some function and \( L \subseteq \{0, 1\} \).

\( L \in \text{SPACE}(S(n)) \iff \exists \text{ TM that uses at most } O(S(n)) \text{ tape cells on all inputs of length } n \text{ and decides } L. \)

The class \( \text{PSPACE} := \bigcup_{k=1}^{\infty} \text{SPACE}(n^k) = \text{poly-space} = \text{problems that can be decided in polynomial space.} \)

Main result: \( \text{PSPACE[for DTM]} = \text{PSPACE[for NTM]} \)

\( L' \) is \( \text{PSPACE-complete} \) :\( \iff \) \( [L' \in \text{NP and } \forall L \in \text{PSPACE}: L \leq_p L'] \)

Many interesting problems are \( \text{PSPACE-complete}: \) Quantified Boolean Formulas (QBF), many zero-sum games, ...
QuickSort: A Randomized Algorithm

- **QuickSort**: Pick pivot element \(a_i\) from list \((a_1, \ldots, a_n)\) at random.

- **Divide list in two** lists, one containing all elements \(< a_i\), the other all elements \(> a_i\).

- **Recursively** sort both lists and join together result.

- **Expected running time**: \(O(n \log n)\)

- **Important**: Holds for **all** inputs. Expectation is over internal algorithm randomization.

- But **worst-case running time** is still \(O(n^2)\).

- QuickSort falls outside complexity classes considered so far ...
Probabilistic Turing Machines

- A Probabilistic TM (PTM) is like a standard deterministic TM, but with extra tape filled with random bits, i.e. independent fair coin tosses (head=1 and tail=0).

- A Polynomial PTM halts after a polynomial (in the length of the input) number of steps.

- A PTM can make mistakes: Randomized algorithms may fail to produce the desired output with some specified failure probability.
Random Polynomial Time Classes

A Polytime PTM can recognize languages in different ways:

**RP** (Randomized Polynomial time) 1-sided error:
- \( x \in L \Rightarrow Pr[M(x) = 1] \geq 1/2, \quad x \notin L \Rightarrow M(x) = 0 \)

**coRP** (complement of RP) 1-sided error:
- \( x \in L \Rightarrow M(x) = 1, \quad x \notin L \Rightarrow Pr[M(x) = 0] \geq 1/2 \)

**BPP** (Bounded-error Probabilistic Polynomial time) 2-sided error:
- \( x \in L \Rightarrow Pr[M(x) = 1] \geq 2/3, \quad x \notin L \Rightarrow Pr[M(x) = 1] \leq 1/3 \)

**ZPP** (Zero-error Probabilistic Polynomial time) 0 error:
- \( x \in L \Rightarrow M(x) = 1 \) or ‘don’t know’, \( x \notin L \Rightarrow M(x) = 0 \) or ‘don’t know’
  and \( \forall x : Pr[M(x) = ‘don’t know’] \leq 1/2 \)

**Note:** PP := similar BPP but with \( 1/3 \& 2/3 \sim 1/2. \) PP is very different!
Relation Between the Random Classes

- One can boost probabilities from 1/2 or 2/3 to get exponentially close to 1 by poly-many repeated runs of algorithm.
- Combine results by “reject if one exception” -or- by “majority vote”
- By repeating ZPP, we always get the correct answer in expected polytime.

\[
\text{BPP} \subseteq \text{PSPACE} \\
\cup \\
\text{P} \subseteq \text{ZPP} \subseteq \text{RP} \subseteq \text{NP} \\
\mid \\
\text{RP} \cap \text{coRP}
\]

- None of the inclusions is known to be strict (or equality).
- Indeed, a proof of one of the ‘hor.’ \(\subseteq\) to be \(\subsetneq\), would prove \(P \neq NP\).
- Many algorithms have been derandomized (with ingenuity). Conjecture: All can be derandomized, i.e. \(\text{BPP} = \text{RP} = \text{ZPP} = P\).
Complexity of Primality Testing

- **Prime factorization** \((504 = 2^3 \times 3^2 \times 7)\) obviously in NP.

- Several computer security techniques rely on the assumption that it is hard to factor numbers.

- **Public-Key Cryptography:**
  RSA uses public 128bit integer (used for encryption) that is a product of two private 64bit primes (used for decryption).

- **Public-Key Signatures:** I encode contract with two private 64bit primes. You decode it with public public 128bit integer. Only I have the factors, so only I could have sent the contract, so you know/can prove the letter (promise/signature/offer) is from me.

- **Coding** uses modular arithmetic and exploits Fermat’s little theorem.

- Amazingly **Primes** \(\in \mathbb{P}\) (AKS 2002, Goedel Prize)

- But finding the factors seems to be hard! RSA is safe – for now.
An oracle Turing machine $M^O$ is a TM with a special “oracle” tape and three special states $q_{\text{query}}$, $q_{\text{yes}}$, $q_{\text{no}}$.

The oracle is a language $O \subseteq \{0, 1\}^*$.

If $M$ enters the state $q_{\text{query}}$, the machine moves into the state $q_{\text{yes}}$ if $y \in O$ and $q_{\text{no}}$ if $q \notin O$, where $y$ denotes the contents of the special oracle tape.

A membership query to $O$ counts only as a single computational step.

Nondeterministic oracle TMs are defined similarly.

Example application: $P^{\text{EXP}} = NP^{\text{EXP}}$ but $P^{\text{rand}} \neq NP^{\text{rand}}$

Halting oracle: $O = \emptyset \equiv L_U = \{\langle M, w \rangle : M(w) \text{ halts}\}$
Hierarchies

There are many different hierarchies in different disciplines:

- **Second-order arithmetic**: Analytical hierarchy $\Sigma^1_n, \Pi^1_n, \Delta^1_n$

- **First-Order Logic**: Arithmetical hierarchy $\Sigma^0_n, \Pi^0_n, \Delta^0_n$ ($\#\text{alt.}\forall\exists\ldots$)

  Computability Theory: Turing jumps = hierarchy of halting oracles $\emptyset^{(n)}$

- **Complexity Theory**: Polynomial hierarchy (PH): $\Sigma^P_n, \Pi^P_n, \Delta^P_n$
  = Hierarchy of NP-oracles. $\text{PH} \subseteq \text{PSPACE}$

The hierarchies themselves form a hierarchy: The smallest = base class in each item includes the entire hierarchy of the next item.
Time Hierarchy

Time Hierarchy Theorem: If $f$ and $g$ are time-constructible functions satisfying $f(n) \log f(n) = o(g(n))$, then $\text{TIME}(f(n)) \subsetneq \text{TIME}(g(n))$.

In particular: $\text{TIME}(\log n) \subsetneq \text{TIME}(n) \subsetneq \text{TIME}(n^2) \subsetneq ... \subsetneq \text{P} \subsetneq \text{EXP}$

Space Hierarchy Theorem: If $f$ and $g$ are space-constructible functions satisfying $f(n) = o(g(n))$, then $\text{SPACE}(f(n)) \subsetneq \text{SPACE}(g(n))$.

In particular: $L \subsetneq \text{SPACE}(n) \subsetneq \text{SPACE}(n^2) \subsetneq ... \subsetneq \text{PSPACE}$

Finally we have some separation results!

Existence of NP-Intermediate Problems: Suppose $P \neq \text{NP}$.
Then there exists a language $L \in \text{NP} \setminus P$ that is not NP-complete.
Function Classes

- Classical Classes (P, NP, PSPACE, RP, ...) are language classes.

- Every language class has a corresponding function class (FP, FNP, FPSPACE, FRP, ...):

  - Time/space-bounded TM that computes function \( f : \{0, 1\}^* \to \mathbb{N} \cong \{0, 1\}^* \).

- Languages are special case of 0-1-valued functions:
  \[ L = \{ f(x) = 1 \} \implies P \subset FP, \text{ etc.} \]

- FP can be reduced to polynomially many problems in P, each predicting one bit of the FP problem. Similarly for other classes. In this sense, P and FP are equally hard.
**Finding Witnesses**

- **Witness**: Sequence of non-deterministic choices in NTM computation.

- For many decision problems in NP, the witness is the solution of the underlying function problem.

- **Example**: NP-algorithm for SAT guesses a satisfying assignment \( w \in \{0, 1\}^{\#\text{vars}} \).

- If \( P=NP \), then \( w \) can also be found in polytime:

- **Algorithm**: Incrementally increase partial solution for \( w = w_1...w_{p(n)} \):
  Assume \( w_1...w_{k-1} \) is known. Let \( M_k \) be NTM \( M \) but with first \( k \) choices fixed to \( w_1...w_{k-1} \). If \( M' \) accepts \( x \), choose \( w_k = 1 \) else \( w_k = 0 \).

- **Cf.** Primes \( \in P \), but finding factors seems hard.
Universal Levin Search

- **Levin Search (LS)** is fastest algorithm within a (large) constant factor to invert a function $g : Y \rightarrow X$, if $g$ can be evaluated quickly.

- **LS Algorithm** runs all programs $p_1, p_2, \ldots, p_k$ on input $x$ each for $k$ time-steps, for $k = 1, 2, 3, \ldots$. Whenever a program $p_i$ halts with some output, say $y$, LS cmp. $g(y)$ and if $g(y) = x$, LS halts with output $y$.

- **Application**: Assume someone found a non-constructive proof of P=NP, then LS finds witnesses of any NP (complete) problem in polynomial time.

- **Proof**: Let $g(\varphi, z) = (\varphi, \varphi(z))$. $\varphi(z)$ is evaluation of Boolean Formula $\varphi$ for variable assignment $z$. Let $p$ be any alleged polytime alg. $p(\varphi) = T \iff \varphi \in \text{SAT}$. Convert to polytime alg. $p'(\varphi, T) = (\varphi, z)$, which is inverse of $g$. Run LS for $x = (\varphi, T)$. Since LS executes $p$ with constant slow-down, if $\varphi \in \text{SAT}$, LS finds $(\varphi, z)$ in poly-time, where $z$ is satisfying assignment of $\varphi$.

- **Note**: If $\varphi \notin \text{SAT}$, LS does not terminate.
Why is PSPACE “harder” than NP

- True NP instances can (at least) be easily verified:
  Provide witness = accepting-ID-path of NTM.
  Has polynomial length and can be verified in polynomial time.

- **Example**: Powerful Prover (in PSPACE) provides satisfying assignment for $\varphi$. Verifier can check correctness in polytime.
  Not possible for unsatisfiable $\varphi$.

- **Example**: Composite numbers:
  Prover provides factors. Verifier checks by multiplication.
  Remarkably also possible for Primes ($\text{Primes} \in \text{NP}$).

- **No short proofs=certificates for PSPACE complete problems**:
  Prover even of unlimited power cannot convince poly-time verifier
  that some Language is in some class. [if PSPACE $\neq$ NP]

- **Examples**: Prover cannot convince Verifier that white (or black) has winning strategy in many zero-sum games such as $n \times n$ Chess or Go, or that QBF formula $\varphi$ is true.
Interactive Proofs (IP): Sequential interaction between Prover ∈ PSPACE and Verifier ∈ BPP.

Problem ∈ IP ⇔ Verifier ∈ BPP can ask Prover ∈ PSPACE sequence of nasty questions and get convinced.

PSPACE = IP! → PSPACE can be efficiently verified by interactive proof.

Simple example of protocol: Colorblind verifies that prover can distinguish shoe color. Verifier repeatedly chooses shoes (identical except for color) at random (but remembering choice) and asks Prover for color. If Prover provides consistent answers, then shoes must indeed have different color.

Real examples: Graph NonIsomorphism is in IP and the Permanent is even IP-complete.