Computational Complexity: Classes

AA-GEI FIB, UPC

Spring 2025-2026

Problem types

- Decision

  Input x *in* I

  Is $P(x)$ *true?*

  Example: Given a graph and two vertices, is there a path joining them?

- Function

  Input x *in* I

  Compute y *such that* $Q(x, y)$

  Example: Given a graph and two vertices, compute the minimum distance between them.

Problem types

- Decision
  Input x
  Property $P(x)$

  By coding inputs on alphabet Σ such a problem identifies a language: $\{\langle x\rangle \in \Sigma^* \mid P(x)\} \in \mathcal{P}(\Sigma^*)$

- Function
  Input x
  Compute y *such that* $Q(x, y)$

  By coding inputs/outputs on alphabet Σ a deterministic algorithm solving a problem determines a partial function $f : \Sigma^* \to \Sigma^*$ s.t., for any x, $Q(x, f(x))$ is true.

- We need to guarantee that $\{\langle x\rangle \mid x \in I\}$ is decidable.

Decision problem classes

- Undecidable
  No algorithm can solve the problem.
- Decidable
  There is an algorithm solving them.

Classifying decidable problems

- Decidable problems, admit algorithmic solutions.
- The algorithms solving such problems can use more or less amount of resources.
- To measure the performance of an algorithm we use two measures time and space.
- As usual we take a worst case analysis on inputs with the same size.
- For a TM, time correspond to the length of the computation and space to the portion of the tape accessed during the computation.
- Observe that sometimes the space used might be smaller than the input.

Complexity classes: definitions

Let $f : \mathbb{N} \to \mathbb{N}$,

- The class $\mathrm{TIME}(f(n))$ is the class of decision problems for which an algorithm exists that solves instances of size n in time $O(f(n))$.
- The class $\mathrm{SPACE}(f(n))$ is the class of decision problems for which an algorithm exists that solves instances of size n using space $O(f(n))$.

Some complexity classes

- P $= \cup_k TIME(n^k) = TIME(poly(n))$
- EXP $= TIME(2^{poly(n)})$
- 2EXP $= TIME(2^{2^{poly(n)}})$
- L = LOGSPACE $= SPACE(\lg n)$
- PSPACE $= SPACE(poly(n))$
- EXPSPACE $= SPACE(2^{poly(n)})$

$$\mathrm{L} \subseteq \mathrm{PSPACE} \subseteq \mathrm{EXPSPACE}$$

$$\mathrm{L} \subseteq \mathrm{P} \subseteq \mathrm{EXP} \qquad \mathrm{PSPACE} \subseteq \mathrm{EXP}$$

Non deterministic machines

- We can use a generalized, although unrealistic, model of computation.
- A program has a built-in operation allowing to branch to different positions in the code.
- When the code is executed, the machinefollows the the “best” continuation.

Complexity classes: definitions

Let $f : \mathbb{N} \to \mathbb{N}$,

- The class NTIME($f(n)$) is the class of decision problems that a non deterministic TM recognizes in time $O(f(n))$.
- The class NSPACE($f(n)$) is the class of decision problems for which a non deterministic TM solves instances of size n using space $O(f(n))$.

Some complexity classes

- NP $=$ NTIME$(poly(n))$
- NEXP $=$ NTIME$(2^{poly(n)})$
- NL $=$ NSPACE$(\lg n)$
- NPSPACE $=$ NSPACE$(poly(n))$

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P}$$

$$\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} = \mathsf{NPSPACE} \subseteq \mathsf{EXP}$$

Logic and NP

- A verifier for a language L is an algorithm $\mathcal{A}$, where $L = \{x \mid \mathcal{A} \textit{ accepts} \langle x, c\rangle \text{ for some word } c\}$.
- A polynomial time verifier runs in polynomial time in the length of x.
- A language L is polynomially verifiable if it has a polynomial time verifier.
- A verifier uses additional information, represented by the word c, to verify that a string x is a member of L. This information is called a certificate, or proof, of membership in L.
- A polynomial verifier can access only polynomial space. Therefore, we can assume that $|c|$ is polynomial in $|x|$.

Decision Problems in NP: Examples

Theorem

$L \in NP$ iff $L = \{x \mid \exists y\, R(x, y)\}$ where $|y| = poly(|x|)$ and R can be decided in polynomial time.

NP is the class of polynomially verifiable languages.

Decision Problems in NP: Examples

- COMPOSITE: Given an integer n, is n composite?
  - $\textsc{Composite} = \{n \mid \exists n_1, n_2\ n = n_1 \times n_2\}$
  - The certificate has polynomial size, and the verifier runs in polynomial time.
  - So, $\textsc{Composite} \in \mathsf{NP}$.
- SUBSET-SUM: Given a set $S = \{x_1, \ldots, x_n\}$ of $\mathbb{Z}^+$ and $t \in \mathbb{Z}$, exists $S' \subseteq S$ with $\sum_{x_j \in S'} x_j = t$?
  - The certificate is a subset of S, it can be represented by a boolean vector, so it has polynomial size. The verifier runs in polynomial time.
  - So, $\textsc{Subset-Sum} \in \mathsf{NP}$.
- HAMILTONIAN CYCLE: Given a graph G, is G Hamiltonian?
  - The certificate is a permutation of $V(G)$, it can be represented by a vector, so it has polynomial size. The verifier also runs in polynomial time.
  - So, $\textsc{Hamiltonian Cycle} \in \mathsf{NP}$.

Classes of complements

- Let $\mathcal{C}$ be a class of decision problems. co-$\mathcal{C}$ is formed by the complements of languages in $\mathcal{C}$, i.e., $L \in$ co-$\mathcal{C}$ iff $\overline{L} \in \mathcal{C}$.
- Classes defined by TMs are closed under complement.

$$\mathrm{P} = \text{co-P} \qquad \mathrm{EXP} = \text{co-EXP} \qquad \mathrm{PSPACE} = \text{co-PSPACE}$$

- Not so clear for classed defined by NTMs.
  $\text{co-NP} = \{x \mid \forall y R(x, y)\}$ where $|y| = poly(|x|)$ and R can be decided in polynomial time.
- It seems different to asses that a property holds for a leaf in a tree than on all their leaves.

Decision Problems in co-NP: Examples

- $\textsc{Prime}$: Given an integer n, is n prime?
  - $\textsc{Prime} = \{n \mid \forall a \leq n \; n \mod a \neq 0\}$.
  - $\textsc{Prime}$ is the complement of $\textsc{Composite}$, we have $\textsc{Prime} \in$ co-NP
  - $\textsc{Prime} \in NP$ as it can be verified in poly time (Pratt 1975). So in NP $\cap$ co-NP
  - A polynomial time algorithm $O(\lg n^6)$ was devised by Agrawal, Kayal and Saxena in 2002.

Decision Problems in NP: Examples

- FACTORIZATION: Given an integer n, are there primes $p_1, \ldots, p_k$ s.t. $x = p_1 \times \ldots \times p_k$.
  - $p_1, \ldots, p_k$ is the certificate.
  - k and all the possible factors p are $\leq n$.
  - The verifier, on input n and $p_1, \ldots, p_k$, computes $p_1 \times \ldots \times p_k$, checks if the result is n. If so, checks that all the values are prime numbers.
  - This takes time $O(k \lg n^2 + k \lg n^6)$.
  - So, FACTORIZATION $\in$ NP.

Open questions

- Is $\mathrm{P} = \mathrm{NP}$?
- $\mathrm{P} \neq \mathrm{EXP}$, but is $\mathrm{NP} = \mathrm{EXP}$?
- $\mathrm{P} \subseteq \mathrm{NP} \cap \text{co-NP}$, but is $\mathrm{P} = \mathrm{NP} \cap \text{co-NP}$?
- Is $\mathrm{L} = \mathrm{NL}$?

Efficient algorithms?

Feasible problem there is an algorithm to find a solution efficiently.

The Determinant Problem: Given an $n \times n$ matrix $M = (m_{ij})$, compute

$$\det(M) = \sum_{\pi \in S_n} (-1)^{\pi} \prod_{i=1}^{n} m_{i,\pi(i)},$$

where $(-1)^{\pi} = \begin{cases} -1 & \text{if } \pi \text{has odd parity,} \\ +1 & \text{if } \pi \text{ has even parity.} \end{cases}$

Can be solved in $O(n^3)$, using the LU decomposition.

Efficient algorithms?

The Permanent Problem: Given an $n \times n$ matrix $M = (m_{ij})$, compute

$$\mathrm{perm}(M) = \sum_{\pi \in S_n} \prod_{i=1}^{n} m_{i,\pi(i)}.$$

$$\mathrm{Perm} \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} = aei + bfg + cdh + ceg + bdi + afh$$

Probably exponential Valiant 1979, best complexity known, $O(n2^n)$. Glynn 2010

Main difference $\det(M_1 \times M_2) = \det(M_1) \times \det(M_2)$ but $\mathrm{perm}(M_1 \times M_2) \neq \mathrm{perm}(M_1) \times \mathrm{perm}(M_2)$

Efficient algorithms?

- In computational complexity efficient algorithm is synonym of polynomial time.
- In practice, it is synonym of a low degree polynomial time.
- Nevertheless, we some times use efficient as synonym of best known cost.

- How to differentiate strict exponential time from polynomial time?
- We cannot do that in most of the cases, we relate such answers to some of the open complexity questions.

Complexity classes for other types of problems

In the same way we can define complexity classes for other types of problems

- Function problems
  Given x, compute y such that $R(x, y)$ holds.
  Classes: FP, FNP, FEXP, ...
- Optimization problems
  Given x, compute y such that $R(x, y)$ holds and $m(x, y)$ is maximum/minimum.
  Classes: PO, NPO, EXPO, ...
- Counting problems
  Given x, how many y are there such that $R(x, y)$?
  Classes: #P

Function Problems: Examples

- Reachability: Given a direct graph G and two vertices u and v, find a path a path from u to v, if one exists.
- Eulerian cycle: Given a graph G, find an Eulerian cycle that traverses all the edges exactly once, if such a cycle exists.
- Factoring: Given an integer n, find its prime factors.
- Subset Sum: Given a sequence of positive integers $S = \{a_1, \ldots, a_n\}$ and $t \in \mathbb{Z}$, obtain $I \subseteq \{1, \ldots, n\}$ s.t. $\sum_{i \in I} a_i = t$.

Optimization Problems: Examples

- Max Cut: Given a graph G, find a partition a partition V into V_1, V_2, such that it maximizes the number of crossing edges between V_1 and V_2.
- Min Cut: Given a digraph G, find a partition V into V_1, V_2, such that it minimizes the number of edges going from V_1 to V_2.
- Shortest path: Given a connected and edge weighted digraph G and $s, t \in V(G)$, find a path between s and t minimizing the sum of the weights in the path.

Counting problems

- How many perfect matchings are there for a given bipartite graph?
- How many graph colorings using k colors are there for a particular graph G?
- What is the value of the permanent of a given matrix whose entries are 0 or 1?
- How many different variable assignments will satisfy a given general boolean formula?

More complexity classes?

Sure, plenty!

Look into the Complexity zoo.

547 classes the last time I've had a look!