Computational Complexity; slides 14, HT 2021
Space Hierarchy Theorem, Gap Theorem, NP-intermediate problems

Prof. Paul W. Goldberg (Dept. of Computer Science, University of Oxford)

HT 2021

A Hierarchy of Complexity Classes

Recall: Relation between complexity classes covered so far:

$$\begin{aligned} \mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \\ \mathsf{PSPACE} = \mathsf{NPSPACE} \subseteq \mathsf{EXP} \subseteq \mathsf{NEXP} \subseteq \\ \mathsf{EXPSPACE} = \mathsf{NEXPSPACE} \subseteq \ldots \end{aligned}$$

Question. Which of these inclusions are strict?

recall: Time Hierarchy theorem

proper complexity function f: roughly, an increasing function that can be computed by a TM in time $f(n) + n$

For $f(n) \geq n$ a proper complexity function, we have

$$\text{TIME}(f(n)) \text{ is a proper subset of } \text{TIME}((f(2n+1))^3).$$

It follows that P is a proper subset of EXP.

Proof used "time-bounded halting language" H_f and a "diagonalising machine"

$$H_f := \{\langle M, w\rangle \ : \ M \text{ accepts } w \text{ after } \leq f(|w|) \text{ steps}\}$$

Space Hierarchy Theorem

Theorem. (Space Hierarchy Theorem)

Let $S, s : \mathbb{N} \to \mathbb{N}$ be functions such that

1. S is "space constructible", and
2. $S(n) \geq n$,
3. $s = o(S)$.

Then DSPACE(s) $\subsetneq$ DSPACE(S).

Reminder: item 3 means that $\lim_{n\to\infty}(s(n)/S(n)) = 0$.

Space-Constructible Functions

Definition.

$f : \mathbb{N} \to \mathbb{N}$ is space constructible if $f(n) \geq \log n$ and $f(n)$ can be computed from input $1^n := \underbrace{1 \ldots 1}_{n \text{ times}}$ in space $\mathcal{O}(f(n))$.

Most standard functions are space-constructible:

- All polynomial functions (e.g. $3n^3 - 5n^2 + 1$)
- All exponential functions (e.g. 2^n)

Space-Constructible Functions

Definition.

$f : \mathbb{N} \to \mathbb{N}$ is space constructible if $f(n) \geq \log n$ and $f(n)$ can be computed from input $1^n := \underbrace{1 \ldots 1}_{n \text{ times}}$ in space $\mathcal{O}(f(n))$.

Most standard functions are space-constructible:

- All polynomial functions (e.g. $3n^3 - 5n^2 + 1$)
- All exponential functions (e.g. 2^n)

For any space-constructible function f we can build a counter that goes off after $f(n)$ cells have been used on inputs of length n.

Consequence: As polynomials are space constructible:

We can enforce that in an n^k-space bounded NTM $\mathcal{M}$ all computations halt after using $\mathcal{O}(n^k)$ space.

(Let $\mathcal{M}$ and a "counter" run in parallel. Stop if the counter goes off.)

Digression: Time-Constructible Functions

Definition.

$f : \mathbb{N} \to \mathbb{N}$ is time constructible if $f(n) \geq n \log n$ and $f(n)$ can be computed from input $1^n := \underbrace{1 \ldots 1}_{n \text{ times}}$ in time $\mathcal{O}(f(n))$.

Similar points apply for time constructible functions (as for space constructible ones, previous slide).

Proof of Space Hierarchy Theorem — Part I

Construct S-space bounded TM $\mathcal{D}$ as follows.

1. On input $\langle \mathcal{M}, w \rangle$, let $n = |\langle \mathcal{M}, w \rangle|$.
2. If the input is not of the form $\langle \mathcal{M}, w \rangle$, then reject.
3. Compute $S(n)$ and mark off this much tape. If later stages ever exceed this allowance, then reject.
4. Simulate $\mathcal{M}$ on input $\langle \mathcal{M}, w \rangle$ while counting number of steps used in simulation; if count ever exceeds $2^{S(n)}$, then reject.

   The simulation introduces only a constant factor c space overhead.
5. If $\mathcal{M}$ *accepts*, then reject; otherwise accept.

$\mathcal{L}(\mathcal{D}) = \{\langle \mathcal{M}, w \rangle : \mathcal{D} \text{ accepts } \langle \mathcal{M}, w \rangle\}$.

By construction, $\mathcal{L}(\mathcal{D}) \in \mathrm{DSPACE}(S)$

Proof of SH Theorem — Part II

Claim. $\mathcal{L}(\mathcal{D}) \notin \mathrm{DSPACE}(s)$

Towards a contradiction,

let $\mathcal{B}$ be a s space bounded TM with $\mathcal{L}(\mathcal{B}) = \mathcal{L}(\mathcal{D})$.

- As $s = o(S)$ there is $n_0 \in \mathbb{N}$ such that $S(n) \geq c \cdot s(n)$ for all $n \geq n_0$.
- Hence, for almost all inputs $\langle \mathcal{B}, w \rangle$ (all $\langle \mathcal{B}, w \rangle \geq n_0$)

  $\mathcal{D}$ completely simulates the run of $\mathcal{B}$ on $\langle \mathcal{B}, w \rangle$
- Hence, for almost all $w \in \{0,1\}^*$

  $\langle \mathcal{B}, w \rangle \in \mathcal{L}(\mathcal{D}) \iff \mathcal{B}$ does not accept $\langle \mathcal{B}, w \rangle$ (Def of $\mathcal{D}$)

  $\langle \mathcal{B}, w \rangle \in \mathcal{L}(\mathcal{B}) \iff \mathcal{B}$ accepts $\langle \mathcal{B}, w \rangle$. (Def of "$\mathcal{L}(\mathcal{B})$")

A Hierarchy of Complexity Classes

Consequence of hierarchy theorems:

- LOGSPACE $\subsetneq$ PSPACE $\subsetneq$ EXPSPACE
- P $\subsetneq$ EXP

Relation between complexity classes covered so far:

$$
\begin{array}{ccccccccc}
\mathsf{L} & & \subseteq & & \mathsf{NL} & & \subseteq & \mathsf{P} & \subseteq & \mathsf{NP} & \subseteq \\
\neq & & & & \neq & & & \neq & & \neq & \\
\mathsf{PSPACE} & = & & = & \mathsf{NPSPACE} & & \subseteq & \mathsf{EXP} & \subseteq & \mathsf{NEXP} & \subseteq \\
\neq & & & & \neq & & & & & & \\
\mathsf{EXPSPACE} & = & & = & \mathsf{NEXPSPACE} & & \subseteq & \ldots & & &
\end{array}
$$

The Gap Theorem

Question. Given more resources, can we always solve more problems?

How much more resources do we need to be able to solve more problems? (Can we solve strictly more problems in time $2^{2^{g(n)}}$ than in $g(n)$?)

The Gap Theorem

Question. Given more resources, can we always solve more problems?

How much more resources do we need to be able to solve more problems? (Can we solve strictly more problems in time $2^{2^{g(n)}}$ than in $g(n)$?)

Theorem. (Gap theorem for time complexity)

For every total computable function $g : \mathbb{N} \to \mathbb{N}$ with $g(n) \geq n$ there is a total computable function $f : \mathbb{N} \to \mathbb{N}$ such that

$$\mathrm{DTIME}\big(f(n)\big) = \mathrm{DTIME}\big(g(f(n))\big)$$

Analogously for space complexity.
contrast with Time hierarchy theorem

For $f(n) \geq n$ a proper complexity function, we have $\mathrm{TIME}(f(n))$ is a proper subset of $\mathrm{TIME}((f(2n+1))^3)$.

The Gap Theorem

Special case (Papadimitriou's book, theorem 7.3): There is a recursive function $f : \mathbb{N} \to \mathbb{N}$ such that $\mathrm{TIME}(f(n))=\mathrm{TIME}(2^{f(n)})$. Proof works by constructing f such that no TM, on input of length n, halts between $f(n)$ and $2^{f(n)}$ steps.

Corollaries of Gap theorem. There are computable functions f such that

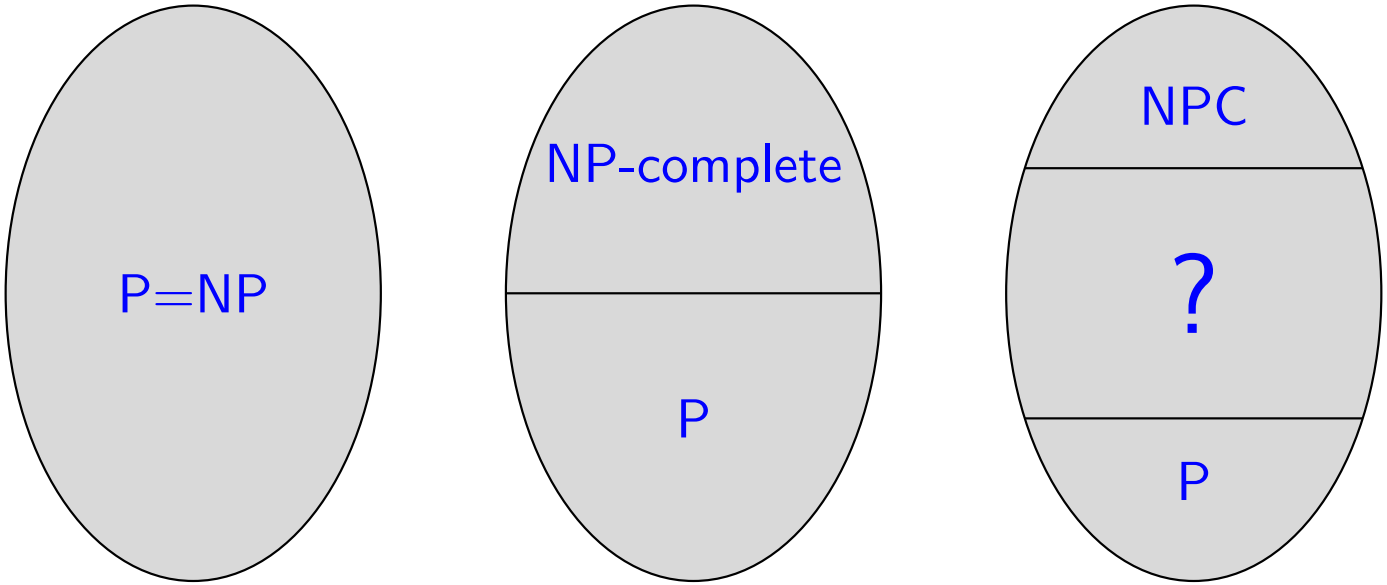
- $\mathrm{DTIME}(f) = \mathrm{DTIME}(2^f)$
- $\mathrm{DTIME}(f) = \mathrm{DTIME}(2^{2^f})$
- $\mathrm{DTIME}(f) = \mathrm{DTIME}\left(\left\{ 2^{2^{\cdot^{\cdot^{2}}}} \right\} f(n) \text{ times} \right)$

However, the functions f are not time (space) constructible.

NP-Intermediate Problems

Question.

- Can we classify any problem in NP as polynomial or NP-complete?
- Which of the following diagrams corresponds to a true picture of NP?



Ladner's theorem

background

Cook/Levin (1971): SAT is NP-complete
Karp (1972): many other diverse NP problems of interest also NP-complete

Ladner's Theorem (1975)

If P $\neq$ NP then there is a language in NP that is neither in P not NP-complete.

Proof. Non-constructive argument (using diagonalisation). (details in Papadimitriou Chapter 14; Arora/Barak Ch.3).

Proof idea

Diagonalisation; let M_i be i-th Turing machine...

For $f : \mathbb{N} \longrightarrow \mathbb{N}$ let $\mathrm{SAT}_f = \{\varphi 1^{n^{f(n)}} \ : \ \varphi \in \mathrm{SAT} \text{ and } n = |\varphi|\}$

Q: How hard is SAT_f for f constant? $f(n) = n$?

Let $f(n)$ be smallest $i < \log\log n$ such that for every bit-string x with $|x| < \log n$, M_i on input x outputs $\mathrm{SAT}_f(x)$ within $i|x|^i$ steps; if no such i, set $f(n) = \log\log n$.

$f(n)$ can be computed from n in $O(n^3)$ time

Proof idea

Diagonalisation; let M_i be i-th Turing machine...

For $f : \mathbb{N} \longrightarrow \mathbb{N}$ let $\text{SAT}_f = \{\varphi 1^{n^{f(n)}} \ : \ \varphi \in \text{SAT} \text{ and } n = |\varphi|\}$

Q: How hard is SAT_f for f constant? $f(n) = n$?

Let $f(n)$ be smallest $i < \log\log n$ such that for every bit-string x with $|x| < \log n$, M_i on input x outputs $\text{SAT}_f(x)$ within $i|x|^i$ steps; if no such i, set $f(n) = \log\log n$.

$f(n)$ can be computed from n in $O(n^3)$ time

Claim. $\text{SAT}_f \in \text{P}$ iff $f = O(1)$.

Then if $\text{SAT}_f \in \text{P}$, solved by some TM M_i — for $n > 2^{2^i}$, $f(n) \leq i$ — f never gets larger than a constant.

If SAT_f is NP-complete, consider reduction from SAT to SAT_f. Reduction must map instances of SAT to instances of SAT_f only polynomially larger...

NP-Intermediate Problems

Ladner's theorem gives an *artificial* problem between P and NP.
Other candidates exist, however. Keep in mind, *unconditional* NP-intermediateness is too much to hope for...
We can base this property on stronger assumptions than P≠NP.

Garey and Johnson 1979.

In their text book they highlight three problems whose complexity was undecided:

- Linear Programming
- Primes/Composite
- Graph Isomorphism

The first 2 of these now known to belong to P.
Total search problems (FACTORING, Nash equilibrium computation, and others) are NP-intermediate assuming they're not in P, and NP≠co-NP.

Graph Isomorphism

Definition. An isomorphism between two graphs H and G is a function $f : V(H) \to V(G)$ such that

1. f is a bijection between $V(H)$ and $V(G)$ and
2. for all $u, v \in V(H)$: $\{u, v\} \in E(H) \iff \{f(v), f(u)\} \in E(G)$.

Graph Isomorphism	
Input:	Undirected graphs G and H
Problem:	Is there an isomorphism between H and G?

Graph Isomorphism

Definition. An isomorphism between two graphs H and G is a function $f : V(H) \to V(G)$ such that

1. f is a bijection between $V(H)$ and $V(G)$ and
2. for all $u, v \in V(H)$: $\{u, v\} \in E(H) \iff \{f(v), f(u)\} \in E(G)$.

GRAPH ISOMORPHISM	
Input:	Undirected graphs G and H
Problem:	Is there an isomorphism between H and G?

- GRAPH ISOMORPHISM seems hard; but if it's NPC then PH collapses to second level!
- every problem in NL is logspace reducible to GI.
- GI also denotes class of problems poly-time reducible to it

Graph Isomorphism

Definition. An isomorphism between two graphs H and G is a function $f : V(H) \to V(G)$ such that

1. f is a bijection between $V(H)$ and $V(G)$ and
2. for all $u, v \in V(H)$: $\{u, v\} \in E(H) \iff \{f(v), f(u)\} \in E(G)$.

GRAPH ISOMORPHISM	
Input:	Undirected graphs G and H
Problem:	Is there an isomorphism between H and G?

- GRAPH ISOMORPHISM seems hard; but if it's NPC then PH collapses to second level!
- every problem in NL is logspace reducible to GI.
- GI also denotes class of problems poly-time reducible to it

Subgraph isomorphism. If we only require f to be injective, then the problem becomes NP-complete.

NP *total search* problems (more later)

Decision problem: one bit output (yes/no)
Search (or, function computation) problem: $poly(n)$ bits of output

NP search problem: binary relation $R(\cdot,\cdot)$ checkable in polynomial time; given x find y such that $R(x,y)$. Finding yes/no answer to an NP decision problem is polynomial-time equivalent to finding y (certificate of input x)
NP total search problem: as above but we have:
$\forall x \exists y \quad |y| = poly(|x|), R(x,y)$

Important example: FACTORING.
Key fact: Problems like FACTORING cannot be NP-hard unless NP=co-NP.

Hence, FACTORING is NP-intermediate in a strong sense (but not in quite such a strong sense as problems from Ladner's theorem).

Conclusion: Complexity of Decision Problems

Decision problems:

- Established a hierarchy of complexity classes for decision problems.
- Tools to classify problems into the correct complexity class.
- Examples for typical problems in various complexity classes.

Theoretical considerations:

- Analysis based on the asymptotic worst-case behaviour.
- NP-hardness: "*There is no algorithm that on all inputs computes the correct answer asymptotically in polynomial time.*"

Hierarchy:

$$\begin{array}{ccccccccc}
\mathsf{L} & \subseteq & \mathsf{NL} & \subseteq & \mathsf{P} & \subseteq & \mathsf{NP} & \subseteq \\
 & \neq & & \neq & & \neq & & \\
\mathsf{PSPACE} & = & \mathsf{NPSPACE} & \subseteq & \mathsf{EXP} & \subseteq & \mathsf{NEXP} & \subseteq \\
 & \neq & & \neq & & & & \\
\mathsf{EXPSPACE} & = & \mathsf{NEXPSPACE} & \subseteq & \ldots & & &
\end{array}$$

Worst-Case Analysis

NP-hardness: *"There is no algorithm that on all inputs computes the correct answer asymptotically in polynomial time."*
For practical purposes, gives guidance on what kind of algorithm (and performance guarantee) you can reasonably expect.

Possible relaxations:

- Relax time constraint:
  - Average case complexity
  - Randomised algorithms
- Relax correctness constraint:
  - Randomised algorithms with bounded error probability.
  - SAT: instances arising from model-checking may be "solvable in practice"
  - Heuristics (for optimisation problems)
  - Approximation (for optimisation problems)