Computational Complexity; slides 10, HT 2022
Polynomial hierarchy, LOGSPACE

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

HT 2022

The polynomial-time hierarchy

- NP: given an existentially-quantified QBF, is it true?
- co-NP: given a universally-quantified QBF, is it true?
- PSPACE: given an unrestricted QBF, is it true?

The polynomial-time hierarchy

- NP: given an existentially-quantified QBF, is it true?
- co-NP: given a universally-quantified QBF, is it true?
- PSPACE: given an unrestricted QBF, is it true?

"intermediate" problems:

- Evaluate formula of the form $\exists x_1, \ldots, x_n \forall y_1, \ldots, y_n \ \varphi$
- Evaluate formula of the form $\forall x_1, \ldots, x_n \exists y_1, \ldots, y_n \ \varphi$
- Evaluate formula of the form $\exists x_1, \ldots, x_n \forall y_1, \ldots, y_n \exists z_1, \ldots, z_n \ \varphi$
- etc.

$\rightsquigarrow$ yet more complexity classes! (seemingly)

Sipser, chapter 10.3 (brief mention); Arora/Barak Chapter 5

The polynomial-time hierarchy

There are multiple equivalent definitions of the classes of the polynomial hierarchy. — Wikipedia

Model of computation for (say) $\exists x_1, \ldots, x_n \forall y_1, \ldots, y_n \ \varphi$?

The polynomial-time hierarchy

There are multiple equivalent definitions of the classes of the polynomial hierarchy. — Wikipedia

Model of computation for (say) $\exists x_1, \ldots, x_n \forall y_1, \ldots, y_n\ \varphi$?

—Yes, poly-time alternating TM where $\exists$ states must precede $\forall$ states, in any computation.
Any such formula has ATM of this kind that solves it; any ATM can be converted to equivalent $\exists \ldots \forall$-formula.

—Another answer: in terms of oracle machines...

The polynomial-time hierarchy

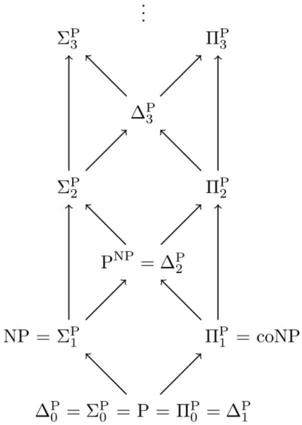

diagram taken from Wikipedia

$\Sigma_{i+1}^{\mathsf{P}} := \mathsf{NP}^{\Sigma_i^{\mathsf{P}}}$
$\Pi_{i+1}^{\mathsf{P}} := \text{co-}\mathsf{NP}^{\Sigma_i^{\mathsf{P}}}$
$\Delta_{i+1}^{\mathsf{P}} := \mathsf{P}^{\Sigma_i^{\mathsf{P}}}$

A^B: problems solved by A-machine with oracle for B-complete problem

Warm-up: consider P^{P}, NP^{P}, P^{NP},...

The polynomial-time hierarchy

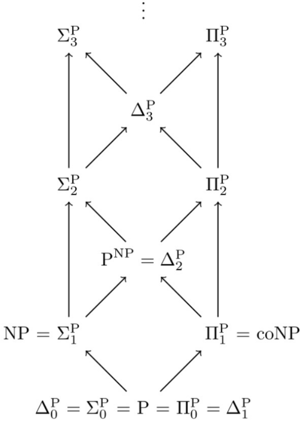

diagram taken from Wikipedia

$$\Sigma_{i+1}^{\mathsf{P}} := \mathsf{NP}^{\Sigma_i^{\mathsf{P}}}$$
$$\Pi_{i+1}^{\mathsf{P}} := \text{co-}\mathsf{NP}^{\Sigma_i^{\mathsf{P}}}$$
$$\Delta_{i+1}^{\mathsf{P}} := \mathsf{P}^{\Sigma_i^{\mathsf{P}}}$$

A^B: problems solved by A-machine with oracle for B-complete problem

Warm-up: consider P^{P}, NP^{P}, P^{NP},...

P^{NP} seems to be more than just NP; indeed there are classes of interest intermediate between NP and P^{NP}!

Examples

Many diverse problems are complete for low levels of PH
`http://ovid.cs.depaul.edu/documents/phcom.pdf`

Example of a Σ_2^P-complete problem: MIN-DNF: consists of a DNF formula φ and integer k.
Question: is there a DNF formula ψ for which $\psi \equiv \varphi$ and ψ has size at most k?

Containment in Σ_2^P: note that the problem is of the form

$\exists$ (bit-string describing ψ) $\forall$ (valuations β of boolean variables)
φ and ψ agree on β

Hardness requires $\exists x \forall y$(formula over variables x, y) to be efficiently encoded as (φ, k), instance of MIN-DNF...

The polynomial-time hierarchy

PH denotes the union of class in the hierarchy

Some key facts:

- PH lies below PSPACE; if any problem is complete for PH, it must belong to the k-th level of the hierarchy, and PH would "collapse" to that level
- Classes in PH are characterised by restricted alternating TMs
- If P is equal to NP, then PH would collapse to P (next slide)
- If NP is equal to co-NP, then PH collapses to NP. (hints that NP $\neq$ co-NP.)

If the *graph isomorphism problem* is NP-complete, then the PH collapses to the second level (Schöning 1987)
...evidence that the problem is not in fact NP-complete.

Theorem: If P is equal to NP, then PH would collapse to P

Proof: If P is equal to NP, it's also the same as co-NP

Recall the expressions

$$\Sigma^{\mathrm{P}}_{i+1} := \mathrm{NP}^{\Sigma^{\mathrm{P}}_i}$$
$$\Pi^{\mathrm{P}}_{i+1} := \text{co-NP}^{\Sigma^{\mathrm{P}}_i}$$
$$\Delta^{\mathrm{P}}_{i+1} := \mathrm{P}^{\Sigma^{\mathrm{P}}_i}$$

and proceed by induction on i

i.e. $\Sigma^{\mathrm{P}}_2 = \mathrm{NP}^{\Sigma^{\mathrm{P}}_1} = \mathrm{NP}^{\mathrm{NP}}$ (by def, $\Sigma^{\mathrm{P}}_1 = \mathrm{NP}$)
$= \mathrm{P}^{\mathrm{P}}$ (by assumption of the theorem)
$= \mathrm{P}$
etc.

PH is "structure between NP and PSPACE": a sequence of classes that "seem" to all be different.

Next: Logarithmic space: structure within P

Logarithmic Space

Polynomial space: seems more powerful than NP.

Linear space: we noted is similar to polynomial space

Sub-linear space?

To be meaningful, we consider Turing machines with separate input tape and only count working space.

LOGSPACE (or, L)	Problems solvable by logarithmic space bounded TM
NLOGSPACE (or, NL)	Problems solvable by logarithmic space bounded NTM

Not hard to show that L⊆NL⊆P

(Sipser Chapter 8.4, Arora/Barak, p.80)

Problems in L and NL

What sort of problems are in L and NL?

In logarithmic space we can store

- a fixed number of counters (up to length of input)
- a fixed number of pointers to positions in the input string.

Problems in L and NL

What sort of problems are in L and NL?

In logarithmic space we can store

- a fixed number of counters (up to length of input)
- a fixed number of pointers to positions in the input string.

Hence,

- LOGSPACE contains all problems requiring only a constant number of counters/pointers for solving.
- NLOGSPACE contains all problems requiring only a constant number of counters/pointers for verifying solutions.

Examples: Problems in L

Example. The language $\{0^n 1^n : n \geq 0\}$

Algorithm.

- Check that no 1 is ever followed by a 0

  Requires no working space. (only movements of the read head)

- Count the number of 0's and 1's.
- Compare the two counters.

Examples: Problems in L

Example. The language $\{0^n 1^n : n \geq 0\}$

Algorithm.

- Check that no 1 is ever followed by a 0

  Requires no working space. (only movements of the read head)
- Count the number of 0's and 1's.
- Compare the two counters.

Example. PALINDROMES $\in$ LOGSPACE
(words that read the same forward and backward)

Algorithm.

- Use two pointers, one to the beginning and one to the end of the input.
- At each step, compare the two symbols pointed to.
- Move the pointers one step inwards.

Example: A Problem in NL

Example. The following problem is in NL:

> REACHABILITY **a.k.a.** PATH
> *Input:* Directed graph G, vertices $s, t \in V(G)$
> *Problem:* Does G contain a path from s to t?

Algorithm.

Set counter $c := |V(G)|$
Let pointer p point to s
while $c \neq 0$ **do**
  if $p = t$ **then** halt and accept
  else
    nondeterministically select a successor p' of p
    set $p := p'$
    $c := c - 1$
reject.

LOGSPACE Reductions

Polynomial-time reductions are too "coarse" to compare poly-time vs. log-space computability.

LOGSPACE Reductions

Polynomial-time reductions are too "coarse" to compare poly-time vs. log-space computability.

Definition. A LOGSPACE-transducer M is a TM with

- a read-only input tape
- a write only, write once output tape
- a memory tape of size $O(\log(n))$

M computes a function $f : \Sigma^* \to \Sigma^*$, where $f(w)$ is the content of the output tape of M running on input w when M halts.

f is called a logarithmic space computable function.

LOGSPACE Reductions

Polynomial-time reductions are too "coarse" to compare poly-time vs. log-space computability.

Definition. A LOGSPACE-transducer M is a TM with

- a read-only input tape
- a write only, write once output tape
- a memory tape of size $O(\log(n))$

M computes a function $f : \Sigma^* \to \Sigma^*$, where $f(w)$ is the content of the output tape of M running on input w when M halts.

f is called a logarithmic space computable function.

Definition.

A LOGSPACE reduction from $\mathcal{L} \subseteq \Sigma^*$ to $\mathcal{L}' \subseteq \Sigma^*$ is a log space computable function $f : \Sigma^* \to \Sigma^*$ such that for all $w \in \Sigma^*$:

$$w \in \mathcal{L} \iff f(w) \in \mathcal{L}'$$

We write $\mathcal{L} \leq_L \mathcal{L}'$

NLOGSPACE (or, NL)-Completeness

NL-completeness.

A problem $\mathcal{L} \in$ NL is complete for NL, if every other language in NL is log space reducible to $\mathcal{L}$.

Theorem. REACHABILITY (or, PATH) is NL-complete.

Proof idea. (details to follow)

Let M be a non-deterministic LOGSPACE TM deciding $\mathcal{L}$.

On input w:

1. construct a graph whose nodes are configurations of M and edges represent possible computational steps of M on w
2. Find a path from the start configuration to an accepting configuration.

NL-Completeness of PATH

some more details.

Construct $\langle G, s, t \rangle$ from M and w using a LOGSPACE-transducer:

1. A configuration $(q, w_2, (p_1, p_2))$ of M can be described in $c \log n$ space for some constant c and $n = |w|$.
2. List the nodes of G by going through all strings of length $c \log n$ and outputting those that correspond to legal configurations.
3. List the edges of G by going through all pairs of strings (C_1, C_2) of length $c \log n$ and outputting those pairs where $C_1 \vdash_M C_2$.
4. s is the starting configuration of G.
5. Assume w.l.o.g. that M has a single accepting configuration t.

$w \in \mathcal{L}$ iff $\langle G, s, t \rangle \in \textsc{Reachability}$

(see Sipser Thm. 8.25)

co-NLOGSPACE

As for time, we consider complement classes for space.

Recall

If $\mathcal{C}$ is a complexity class, we define

$$\text{co-}\mathcal{C} := \{\mathcal{L} : \overline{\mathcal{L}} \in \mathcal{C}\}.$$

From Savitch's theorem:

PSPACE $=$ NPSPACE and hence co-NPSPACE $=$ PSPACE

NLOGSPACE = co-NLOGSPACE

However, from Savitch's theorem we only know

$$\text{NLOGSPACE} \subseteq \text{DSPACE}(\log^2 n).$$

Theorem. (Immerman and Szelepcsényi '87-8)

NLOGSPACE = co-NLOGSPACE

Proof idea.

Show that $\overline{\textsc{Reachability}}$ is in NL.

NLOGSPACE = co-NLOGSPACE

Proof sketch. On input $\langle G, s, t \rangle$, let $m = |V(G)|$.

Define c_i to be number of nodes reachable from s in $\leq i$ steps; compute c_i for increasing $i = 1, 2, \ldots, m$

1. Only node s is reachable in 0 steps, so $c_0 = 1$
2. For each $i = 1, \ldots, m$, set $c_i = 1$, remember c_{i-1}, and for each $v \neq s$ in G
    1. $d := 0$
    2. For each node u in G
        1. guess if reachable from s in $\leq i - 1$ steps, if so do (2,3):
        2. Verify each "yes" guess by guessing an at most $i - 1$ step path from s to u; if so, $d := d + 1$; reject if no such path found
        3. If we guessed that u is reachable, and $(u, v) \in E(G)$, then increment c_i and continue with next v
    3. If total number d of u guessed is not equal to c_{i-1}, then reject

Continued...

NLOGSPACE = co-NLOGSPACE

Proof sketch (continued). On input $\langle G, s, t \rangle$

(at this stage we have c_m)

Then try to guess c_m nodes reachable from s and not equal to t:

1. For each node u in G, guess if reachable from s in m steps
2. Verify each "yes" guess by guessing a $\leq m$ step path from s to u; reject if no such path found
3. If we guessed that u is reachable, and $u = t$, then reject
4. If total number d of u guessed not equal to c_m, then reject
5. Otherwise accept

Algorithm stores (at one time) only 6 counters (u, v, c_{i-1}, c_i, d and i) and a pointer to the head of a path; hence runs in logspace.

(more details in Sipser Theorem 8.27)

To summarise

It's unknown where L is equal to NL, or if NL is equal to P.

$$\mathsf{L} \subseteq \mathsf{NL} = \text{co-}\mathsf{NL} \subseteq \mathsf{P}$$

Still, we have that NL is closed under complement — contrast with NP

By space hierarchy theorem, L $\subsetneq$ PSPACE
Indeed (from s.h.t. and Savitch's theorem) NL $\subsetneq$ PSPACE

Next: more structure within P