Complexity theory, summarized

Evgenij Thorstensen

V18

Classes and theorems

We can divide the complexity theory part of the course into two categories.

After learning about worst-case complexity as a function of input size, we have looked at *classes* and at *theorems*.

Under classes, I put definitions, inclusions, and completeness.

Theorems are big and usually concern many classes.

Worst-case complexity

Abstraction from M uses x time/space on input w to functions of input size.

$t_M(|w|)$ is the maximum time M uses on strings of length $|w|$. Same for space.

Allows comparisons between arbitrary languages and machines.

We compare these functions using asymptotic analysis (loosely, growth rate).

Asymptotic behaviour

$f(n) = O(g(n))$ when there exists n_0 and c such that

$$f(n) \leqslant c \cdot g(n)$$

for all $n \geqslant n_0$.

f then grows no faster than g, up to a constant factor, and g is an upper bound for f.

Can augment to $f(n) = o(g(n))$ by requiring that for *all* $c \in \mathbb{R}^+$,

$$f(n) \leqslant c \cdot g(n)$$

or equivalently, that

$$c \cdot f(n) \leqslant g(n)$$

for all $n \geqslant n_0$.

Functions of interest

$\log n$, polynomials n^k, and exponentials 2^n. All constants equal 2, it simplifies arithmetic.

Polynomials of $\log n$ have also made an appearance.

Using these, we define the deterministic classes $\mathsf{TIME}(f(n))$ and $\mathsf{SPACE}(f(n))$, as well as the nondeterministic classes $\mathsf{NTIME}(f(n))$ and $\mathsf{NSPACE}(f(n))$.

Other classes defined using these.

Classes

Space $\log n$:

- $L = SPACE(\log n)$,
- $NL = NSPACE(\log n)$,
- $polyL = \bigcup_{k \in \mathbb{N}} SPACE((\log n)^k)$

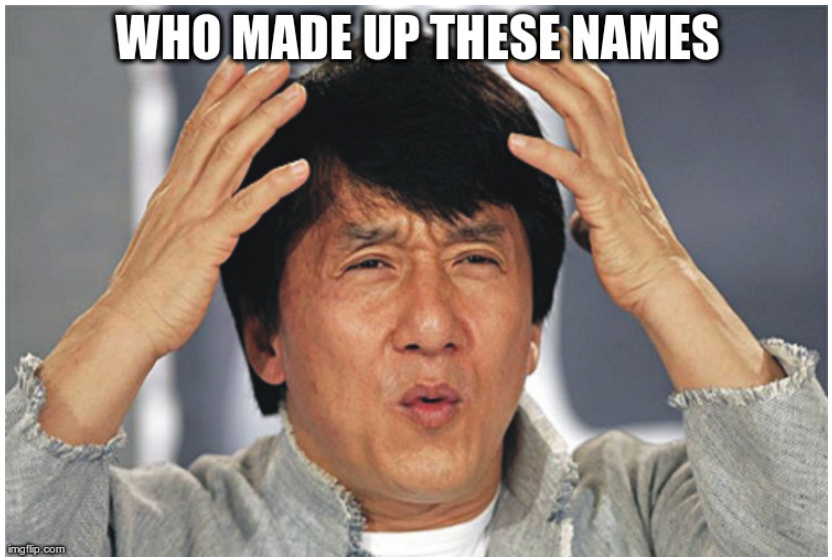
Polynomial time and space:

- $P = \bigcup_{k \in \mathbb{N}} TIME(n^k)$
- $NP = \bigcup_{k \in \mathbb{N}} NTIME(n^k)$
- $PSPACE = \bigcup_{k \in \mathbb{N}} SPACE(n^k)$

And exponential time $EXP = \bigcup_{k \in \mathbb{N}} TIME(2^{(n^k)})$

co-classes

These classes all have corresponding "complement classes", that are not their set complements.



$\mathsf{coNP} = \{\bar{L} \mid L \in \mathsf{NP}\}$, etc.

Only of interest for nondeterministic classes. Completeness is inherited.

Theorems about time and space

Space bounds time (configurations argument):
$\mathsf{SPACE}(f(n)) \subseteq \mathsf{TIME}(2^{cf(n)})$ for some c.

Hierarchy theorems: For time or space constructable functions,

$$\mathsf{SPACE}(o(f(n))) \subset \mathsf{SPACE}(f(n))$$

and

$$\mathsf{TIME}(o(\frac{f(n)}{\log f(n)})) \subset \mathsf{TIME}(f(n))$$

Proof by simulation/diagonalization over deciders.

Theorems about space

Savitch: For $f(n) \geqslant \log n$, $\mathsf{NSPACE}(f(n)) \subseteq \mathsf{SPACE}(f(n)^2)$

Proof by algorithm; we divide search problem into two halves, use the same space for both.

Immermann-Szelepcsényi: For $f(n) \geqslant \log n$, $\mathsf{SPACE}(f(n))$ is closed under complement.

We proved this for NL and coNL, by exhibiting a clever algorithm for the NOPATH problem using counting.

Logarithmic space classes

Computation with very limited memory, but as much time as you like.

Allows a fixed number of pointers into the input and counters up to n.

Have to solve problem by looking at fixed-size pieces of the input at a time.

For NL, prototypical problem is PATH.

NL-completeness

For the class NL, we use logspace reductions, and write $A \leqslant_L B$ if A reduces to B.

Formally, can output a correct yes/no instance of B given an instance of A while using only log space *working memory*.

Allowed to output more than $\log n$ bits.

A problem is NL-complete if it is in NL and all problems in NL reduce to it under $\leqslant_L$.

PATH is such a problem — reduction from TM configuration graph reachability (universal reduction).

Properties of L and NL

We have $\mathsf{L} \subseteq \mathsf{NL} = \mathsf{coNL} \subseteq \mathsf{P}$.

Logspace reductions are polynomial-time reductions, and PATH can be solved in polynomial time.

L is closed under composition and subroutines — by re-computing relevant bits of other problems, can keep space bound.

Polynomial time classes

Classic problems that can be efficiently computed vs. efficiently verified.

Polynomials are closed under composition and so forth.

Verifier definition of NP: DTM that accepts yes-instances in polynomial time for some certificate/proof of membership. Usually this is a candidate solution.

E.g. for SAT, an assignment, for 3COL a colouring, etc.

Allows us to dispence with nondeterminism.

NP-completeness

We use polynomial time reducibility $\leqslant_{\mathrm{P}}$. A problem is NP-complete if it is in NP and all other problems reduce to it under $\leqslant_{\mathrm{P}}$.

Such problems exist, but less obvious than for NL. SAT is the basic such problem.

Again, universal reduction from polynomial-time NTMs to SAT formulas encoding acceptance.

We have seen some gadgets for other reductions. More here https://en.wikipedia.org/wiki/Karp%27s_21_NP-complete_problems

coNP

The most interesting co-class. Corresponds to problems with universal, rather than existential, statements.

Theorem (holds for NL, too): L is coNP-complete if and only if $\bar{L}$ is NP-complete.

A language is in coNP when it has a certificate for the NO-instances.

Example: Tautologies and UNSAT. If NOT a tautology, exists a falsifying assignment. If NOT unsatisfiable, exists a satisfying one.

P, NP, and coNP

If all three are distinct, we get a rich ecosystem around them.

In particular, we get a whole polynomial hierarchy PH consisting of problems that can be answered by querying polynomial-time NTMs.

However, it's hard to seperate classes except by diagonalization, and that won't work here.

Relativization

Oracles machines: TMs with a special "solve this problem in one step" instruction.

Can be used to define the polynomial hierarchy: $\Sigma_i = \Sigma_{i-1}^{\mathsf{NP}}$, and $\Sigma_0 = \mathsf{P}$.

Can also be used to prove that certain techniques (diagonalization/simulation) can't distinguish classes.

Baker-Gill-Solovay

There exist oracles A and B such that $\mathsf{P}^A = \mathsf{NP}^A$ and $\mathsf{P}^B \neq \mathsf{NP}^B$.

Oracle A is PSPACE, oracle B is constructed by diagonalising DTMs that run in polynomial time.

The oracle "outruns" all these DTMs: They can check only polynomially many strings, but an NTM can test all strings of a given length (exponentially many).

Polynomial space classes

PSPACE and NPSPACE. Thanks to Savitch, can ignore NPSPACE when *reasoning*, but use it for proving membership in PSPACE.

Completeness once again wrt. polynomial *time* reducibility, since polynomial space reducibility makes everything trivially complete.

The overpowered reduction: Solve the given instance, and output a tiny yes/no instance depending on answer.

$\leqslant_{\mathrm{P}}$ avoids this. PSPACE has complete problems, prototypical one is TQBF.

PSPACE-completeness

Universal reduction again. We encode reachability between configurations.

Use Savitch trick and universal quantifier to avoid writing two formulas.

Bottoms out into SAT formulas with quantifiers on the outside.

NPSPACE computations correspond to and/or trees.

Odds and ends

Ladner's theorem: If $P \neq NP$, there exist problems in $NP \setminus P$ that are not NP-complete.

Counting class #P: Number of solutions an NP problem has.

Blum's speedup theorem: There are problems without an asymptotically optimal algorithm (can speed up arbitrarily much).