Computational Complexity; slides 7, HT 2019
Logarithmic space

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

HT 2019

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 LOGSPACE

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 LOGSPACE

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

To compare the difficulty of problems in PTIME or NLOGSPACE, polynomial-time reductions no longer make sense...

LOGSPACE Reductions

To compare the difficulty of problems in PTIME or NLOGSPACE, polynomial-time reductions no longer make sense...

Definition. A LOGSPACE-transducer $\mathcal{M}$ is a logarithmic space bounded Turing accepter with a read-only input tape and a write only, write once output tape.

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

f is called a logarithmic space computable function.

LOGSPACE Reductions

To compare the difficulty of problems in PTIME or NLOGSPACE, polynomial-time reductions no longer make sense...

Definition. A LOGSPACE-transducer $\mathcal{M}$ is a logarithmic space bounded Turing accepter with a read-only input tape and a write only, write once output tape.

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

Proof idea.

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

On input w:

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

Proof sketch.

We construct $\langle G, s, t\rangle$ from $\mathcal{M}$ and w using a LOGSPACE-transducer:

1. A configuration $(q, w_2, (p_1, p_2))$ of $\mathcal{M}$ 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 $C_1 \vdash_{\mathcal{M}} C_2$.
4. s is the starting configuration of G.
5. Assume w.l.o.g. that $\mathcal{M}$ has a single accepting configuration t.

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

(see Sipser Thm. 8.25)

As for time, we consider complement classes for space.

Recall

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

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

Complement classes for space:

- $\text{co-NLOGSPACE} := \{\mathcal{L} : \overline{\mathcal{L}} \in \text{NLOGSPACE}\}$
- $\text{co-NPSPACE} := \{\mathcal{L} : \overline{\mathcal{L}} \in \text{NPSPACE}\}$

From Savitch's theorem:

PSPACE = NPSPACE and hence co-NPSPACE = PSPACE

However, from Savitch's theorem we only know

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

Theorem. (Immerman and Szelepcsényi '87-'88)

NLOGSPACE = co-NLOGSPACE

Proof idea.

Show that $\overline{\text{REACHABILITY}}$ is in NL.

NLOGSPACE = co-NLOGSPACE

Proof sketch.

On input $\langle G, s, t \rangle$

First compute c_m, the *number* of nodes reachable from s (in $m = |V(G)|$ steps):
Define c_i to be number of nodes reachable in i steps; compute this for increasing i...

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

NLOGSPACE = co-NLOGSPACE

Proof sketch.

On input $\langle G, s, t \rangle$

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 an at most 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.

(See Sipser Theorem 8.27)

Space and Time Hierarchies

A Hierarchy of Complexity Classes

Recall: Relation between complexity classes covered so far:

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{PTIME} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} = \mathsf{NPSPACE} \subseteq \mathsf{EXPTIME} \subseteq \mathsf{NEXPTIME} \subseteq \mathsf{EXPSPACE} = \mathrm{NEXPSPACE} \subseteq \dots$$

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 EXPTIME.

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 $\mathrm{DSPACE}(s) \subsetneq \mathrm{DSPACE}(S)$.

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).

Recall. $f(n) = o\big(g(n)\big)$ if

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \qquad \text{that is} \qquad \lim_{n\to\infty} \frac{s(n)}{S(n)} = 0.$$

Digression: Space-Constructible Functions

Definition.

A function $f : \mathbb{N} \rightarrow \mathbb{N}$ 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)

Digression: Space-Constructible Functions

Definition.

A function $f : \mathbb{N} \to \mathbb{N}$ 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.

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

Most standard functions are time-constructible:

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

For any time-constructible function f we can build a timer that goes off after $f(n)$ steps on inputs of length n.

Consequence: As polynomials are time-constructible:

We can enforce in an n^k-time bounded NTM $\mathcal{M}$ that all computation paths are of length n^k.

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

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 $\mathrm{DSPACE}(s) \subsetneq \mathrm{DSPACE}(S)$.

Recall. $f(n) = o\big(g(n)\big)$ if

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \qquad \text{that is} \qquad \lim_{n\to\infty} \frac{s(n)}{S(n)} = 0.$$

Proof of SH 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. $\mathcal{L}(\mathcal{D}) \notin \mathsf{DSPACE}(s)$

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:

- LOGSPACE $\subsetneq$ PSPACE $\subsetneq$ EXPSPACE
- PTIME $\subsetneq$ EXPTIME

Recall: Relation between complexity classes covered so far:

$$\begin{array}{ccccccccc}
\mathsf{L} & \subseteq & \mathsf{NL} & \subseteq & \mathsf{PTIME} & \subseteq & \mathsf{NP} & \subseteq \\
\neq & & \neq & & \neq & & \neq & \\
\mathsf{PSPACE} & = & \mathsf{NPSPACE} & \subseteq & \mathsf{EXPTIME} & \subseteq & \mathsf{NEXPTIME} & \subseteq \\
\neq & & \neq & & & & & \\
\mathsf{EXPSPACE} & = & \mathsf{NEXPSPACE} & \subseteq & \ldots & & &
\end{array}$$

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

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

Analogously for space complexity.

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 $2^{2^{f(n)}}$ than in $f(n)$?)

Corollary. There are computable functions g such that

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

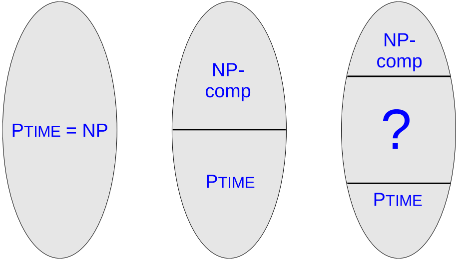
However, the functions g are not time (space) constructible.

NP-Intermediate Problems

NP-Intermediate Problems

Question.

- Do complexity classes only contain easy and hard problems?
- 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

Theorem. (Ladner 1975)

If P $\neq$ NP then NP contains infinitely many (polynomial-time) inequivalent problems.

Proof. Non-constructive argument (using diagonalisation). For details see Papadimitriou Chapter 14.

Ladner's theorem

Theorem. (Ladner 1975)

If P $\neq$ NP then NP contains infinitely many (polynomial-time) inequivalent problems.

Proof. Non-constructive argument (using diagonalisation). For details see Papadimitriou Chapter 14.

Consequence. Unless P $=$ NP, the class NP contains infinitely many problems that are neither in P nor NP-complete.

Ladner's theorem. Unless P=NP, the class NP contains infinitely many problems that are neither in P nor NP-complete.

Which problems are (possible candidates for) NP-intermediate?

Obviously, a proof that a problem is NP-intermediate separates P and NP and hence will not be easy to obtain.

NP-Intermediate Problems

Ladner's theorem. Unless P=NP, the class NP contains infinitely many problems that are neither in P nor NP-complete.

Which problems are (possible candidates for) NP-intermediate?

Obviously, a proof that a problem is NP-intermediate separates P and NP and hence will not be easy to obtain.

Garey and Johnson 1979. In their text book they highlight three problems whose complexity was undecided:

- Linear Programming
- Primes/Composite
- Graph Isomorphism

Linear Programming (LP)

Input: Integer valued $(n \times m)$-matrix M,
$D \in \mathbb{Z}^m \quad C \in \mathbb{Z}^n \quad b \in \mathbb{Z}$

Problem: Is there a vector $X \in \mathbb{Q}^n$ such that $M \cdot X \leq D$ and $C \cdot X \geq b$?

This is the problem to maximise a linear function subject to linear constraints.

Linear Programming (LP)

Input: Integer valued $(n \times m)$-matrix M,
$D \in \mathbb{Z}^m \quad C \in \mathbb{Z}^n \quad b \in \mathbb{Z}$

Problem: Is there a vector $X \in \mathbb{Q}^n$ such that $M \cdot X \leq D$ and $C \cdot X \geq b$?

This is the problem to maximise a linear function subject to linear constraints.

E.g., (Maximise:) $C_1 \cdot X_1 + \ldots + C_n \cdot X_n \quad (\geq b)$

Subject to: $0 \leq M_{1,1} \cdot X_1 + \ldots + M_{1,n} \cdot X_n \leq D_1$

$\vdots$

Subject to: $0 \leq M_{m,1} \cdot X_1 + \ldots + M_{m,n} \cdot X_n \leq D_m$

Linear Programming

Linear Programming (LP)

Input: Integer valued $(n \times m)$-matrix M, $D \in \mathbb{Z}^m \quad C \in \mathbb{Z}^n \quad b \in \mathbb{Z}$

Problem: Is there a vector $X \in \mathbb{Q}^n$ such that $M \cdot X \leq D$ and $C \cdot X \geq b$?

This is the problem to maximise a linear function subject to linear constraints.

In 1979, Leonid Khachiyan proved that this problem is in P (Fulkerson Prize).

LP-based algorithms (e.g. based on the (exponential) simplex method) are among the popular approaches to solve algorithmic problems.

Linear Programming (LP)

Input: Integer valued $(n \times m)$-matrix M,
$D \in \mathbb{Z}^m \quad C \in \mathbb{Z}^n \quad b \in \mathbb{Z}$

Problem: Is there a vector $X \in \mathbb{Q}^n$ such that $M \cdot X \leq D$ and $C \cdot X \geq b$?

This is the problem to maximise a linear function subject to linear constraints.
In 1979, Leonid Khachiyan proved that this problem is in P (Fulkerson Prize).
LP-based algorithms (e.g. based on the (exponential) simplex method) are among the popular approaches to solve algorithmic problems.

Integer programming. As we saw, the problem is NP-complete if X is required to be integer valued.

Prime and Composite Numbers

Primes

Input:	Positive integer $n \in \mathbb{N}$
Problem:	Is n prime?

For a long time, it was only known that this problem is in NP∩co-NP.

In 2002, PRIMES was shown to be in P by Agrawal and two undergraduate students, Kayal, Saxena with their AKS primality test (Gödel Prize, Fulkerson Prize)

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 (GI)

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 (GI)	
Input:	Undirected graphs G and H
Problem:	Is there an isomorphism between H and G?

- every problem in NLOGSPACE 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 (GI)	
Input:	Undirected graphs G and H
Problem:	Is there an isomorphism between H and G?

- every problem in NLOGSPACE 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

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-completeness: "*There is no algorithm that on all inputs computes the correct answer asymptotically in polynomial time.*"

Hierarchy:

$$\begin{array}{ccccccccc} \mathsf{L} & \subseteq & \mathsf{NL} & \subseteq & \mathsf{PTIME} & \subseteq & \mathsf{NP} & \subseteq \\ \neq & & \neq & & \neq & & \neq & \\ \mathsf{PSPACE} & = & \mathsf{NPSPACE} & \subseteq & \mathsf{EXPTIME} & \subseteq & \mathsf{NEXPTIME} & \subseteq \\ \neq & & \neq & & & & & \\ \mathsf{EXPSPACE} & = & \mathsf{NEXPSPACE} & \subseteq & \dots & & & \end{array}$$

Worst-Case Analysis

NP-completeness: *"There is no algorithm that on all inputs computes the correct answer asymptotically in polynomial time."*

Possible relaxations:

- Relax time constraint:
  - Average case complexity
  - Randomised algorithms
- Relax correctness constraint:
  - Randomised algorithms with bounded error probability.
  - Heuristics (for optimisation problems)
  - Approximation (for optimisation problems)

Practical Implications

In practice: From a practical point of view, classifying problems into complexity classes is much less about "solvable" or not ...

Example: Satisfiability

- SATISFIABILITY is one of the most important NP-complete problems.
- However, current SAT-solvers can solve instances coming from bounded model-checking with thousands and sometimes millions of variables.

In practice: ... but about the type of algorithms that will probably work.

- P: explicit construction of solutions
- NP: search for solutions, backtracking etc.
- PSPACE: Algorithms from artificial intelligence for solving games