# P and NP

Evgenij Thorstensen

V18

# Recap

We measure complexity of a DTM as a function of input size $n$. This complexity is a function $t_M(n)$, the maximum number of steps the DTM needs on the input of size $n$.

We compare running times by asymptotic behaviour.

$g \in O(f)$ if there exist $n_0$ and $c$ such that for *every* $n \geqslant n_0$ we have $g(n) \leqslant c \cdot f(n)$.

$\mathsf{TIME}(f)$ is the set of languages decidable by some DTM with running time $O(f)$.

# Polynomial time

The class P, polynomial time, is

$$\mathsf{P} = \bigcup_{k \in \mathbb{N}} \mathsf{TIME}(n^k)$$

To prove that a language is in P, we can exhibit a DTM and prove that it runs in time $O(n^k)$ for some $k$.

This can be done directly or by reduction.

# Abstracting away from DTMs

Let's consider algorithms working on relevant data structures.

Need to make sure that we have a reasonable encoding of input. For now, reasonable = polynomial-time encodable/decodable.

If we have such an encoding, can assume input is already decoded.

For example, a graph $(V, E)$ as input can be reasoned about in terms of $|V|$, since $|E| \leqslant |V|^2$

# Some problems in P

## Problem (PATH)

*Given a directed graph* $(V, E)$ *and* $s, t \in V$*, is there a path from* $s$ *to* $t$*?*

# Some problems in P

## Problem (PATH)

*Given a directed graph* $(V, E)$ *and* $s, t \in V$*, is there a path from* $s$ *to* $t$*?*

Algorithm:

```
1. Mark s
2. While new nodes are marked, repeat:
  2.1. For each marked node v and edge (v, w),
       mark w
3. If t is marked, accept; if not, reject.
```

# Another graph problem

**Problem (Shortest path)**

*Given an edge-weighted undirected graph* $(V, E, w)$*, nodes* $s, t \in V$*, and a bound* $k \in \mathbb{N}$*, is the shortest path from* $s$ *to* $t$ *of weight* $\leqslant k$*?*

Odd formulation. How can we use this to find the size of this shortest path?

Solvable by similar algorithm (BFS).
https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

# Some important properties of P

P is closed under union, intersection, complement, and concatenation.

Additionally, *polynomials* are closed under addition and multiplication.

Closure under multiplication allows for powerful reductions!

# Membership in P by reduction

Given a decision problem L, I can prove that $L \in P$ by reducing L to a problem I already know is in P — if my reduction takes polynomial time.

Recall that a reduction f transforms each $w$ to $f(w)$ such that $w \in L_1 \leftrightarrow f(w) \in L_2$.

Let $L_1 \in P$, and let $M_{L_1}$ be the DTM deciding $L_1$ in polynomial time $p$.

Let f be my reduction, with polynomial time $p_f$. The output of the reduction could be of size at most $p_f$.

# Membership in P by reduction

Given a decision problem L, I can prove that $L \in P$ by reducing L to a problem I already know is in P — if my reduction takes polynomial time.

Recall that a reduction $f$ transforms each $w$ to $f(w)$ such that $w \in L_1 \leftrightarrow f(w) \in L_2$.

Let $L_1 \in P$, and let $M_{L_1}$ be the DTM deciding $L_1$ in polynomial time $p$.

Let $f$ be my reduction, with polynomial time $p_f$. The output of the reduction could be of size at most $p_f$.

The machine $M_{L_1}(f(w))$ therefore runs in time $O(p(p_f(n)))$, which is a polynomial. $(n^k)^l = n^{kl}$.

# The class NP

Recall that $\mathsf{NTIME}(f(n))$ is the class of problems decidable by an NTM in time $O(f(n))$.

NTIME also has a polynomial-based class. $\mathsf{NP} = \bigcup_{k \in \mathbb{N}} \mathsf{NTIME}(n^k)$

Unfortunately, NTMs are annoying to work with. Let's define NP using DTMs.

# Verifiers

**Definition (7.18, reworded)**

A verifier $V$ for a language $L$ is a DTM such that $w \in L$ if and only if there exists a *certificate* $c$ such that $V(w, c)$ accepts.

We measure the running time of verifiers only with respect to $w$.

It follows that if the running time of $V$ is polynomial, $c$ is polynomial in the size of $w$.

# NTMs and verifiers

**Theorem**

NP *is the class of languages that have polynomial-time verifiers.*

Need to prove both directions: NTMs to verifiers and back.

Given a verifier, easy to construct NTM: Just try all certificates of appropriate length.

A given verifier runs in time $O(n^k)$ for some specific $k$.

The resulting NTM has an exponential-size transition table, but runs in polynomial time.

# NTMs to verifiers

Given an NTM, we can build a verifier as follows: Let the certificate be a sequence of choices of transitions.

If there is an accepting branch, there is a sequence of such choices of polynomial size.

Otherwise all branches reject, and so does our verifier.

# Abstracting away TMs

NP is the class of languages with polynomial-size membership proofs.

P $\subseteq$ NP still holds: If I can decide membership in polynomial time, I do not need a certificate.

NP is closed under union, intersection, and concatenation; but is *not known* to be closed under complement.

# Some problems in NP

This class has all the classic useful problems.

The most famous problem in NP is perhaps SAT: Given a propositional logic formula, is it satisfiable?

Propositional formulas are build up from variables $x_i$ using conjunction ($\wedge$), disjunction ($\vee$), and negation $\neg$.

$x_1 \vee (x_2 \wedge \neg x_3)$ is an example. An assignment assigns true or false to each variable, and it is satisfying if the formula evaluates to true.

# SAT is in NP

If I write down an assignment, we can check that it is satifying in linear time.

However, consider the problem UNSAT: The complement of SAT, i.e. those formulas that have *no satisfying assignment*.

Seems similar, but what certificate can we use?

# SAT is in NP

If I write down an assignment, we can check that it is satifying in linear time.

However, consider the problem UNSAT: The complement of SAT, i.e. those formulas that have *no satisfying assignment*.

Seems similar, but what certificate can we use?

This is why NP is not known to be closed under complement.

# 3-colourability

**Problem (3COL)**

*Given an undirected graph* $(V, E)$, *is it possible to colour* $V$ *using 3 colours such that no edge* $(v, w)$ *connects the same colour?*

Again, easy certificate (a colouring).

We can also, like for P, use reductions rather than explicit algorithms.

Let's reduce 3COL to SAT.

# Propositional logic programming

We will need this for the Cook-Levin theorem.

How do we go from graphs to true and false?

Well, we need to know what colour a vertex is. And we need constraints to ensure that

- Each vertex is exactly one colour
- No two neighbours are the same colour

# Gadgetry

A piece of a problem to simulate another piece of another problem is called a gadget.

We need a gadget for each vertex, and one for each edge.

Variables: $x(v_i, R), x(v_i, G), x(v_i, B)$ (true means that vertex $v_i$ is coloured Red, Green, Blue respectively)

# Gadgets

Each vertex must have a colour: $\bigwedge_{v_i \in V} x(v_i, R) \vee x(v_i, G) \vee x(v_i, B)$

Each vertex must not have two colours:

$$\bigwedge_{v_i \in V} \neg[x(v_i, R) \wedge x(v_i, G)] \wedge \neg[x(v_i, R) \wedge x(v_i, B)] \wedge \neg[x(v_i, G) \wedge x(v_i, B)]$$

# Gadgets

Each vertex must have a colour: $\bigwedge_{v_i \in V} x(v_i, R) \vee x(v_i, G) \vee x(v_i, B)$

Each vertex must not have two colours:

$$\bigwedge_{v_i \in V} \neg[x(v_i, R) \wedge x(v_i, G)] \wedge \neg[x(v_i, R) \wedge x(v_i, B)] \wedge \neg[x(v_i, G) \wedge x(v_i, B)]$$

No edge monochromatic: $\bigwedge_{(v_i, v_j) \in E, C \in \{R, G, B\}} \neg[x(v_i, C) \wedge x(v_j, C)]$

# Analyzing the reduction

Correctness and polynomial time bound.

A satisfying assignment satisfies all my conjunctions of constraints.

If it exists, then exactly one of each $x(v_i, C)$ will be true. Also, no edge $(v, w)$ will have $x(v, C)$ and $x(w, C)$.

The true variables give me a colouring.

# 3COL to SAT, time

How much time did we spend?

The "exactly one colour" formulas used $3|V| + 3|V|$ variables.

The edge formulas used $3|E|$ variables. Total $6|V| + 3|E|$.

# Many-one reductions

**Definition (Polynomial-time many-one reducibility)**

A language $A$ is polynomial-time many-one reducible to another language $B$ ($A \leqslant_P B$) if there exists a polynomial-time computable function $f$ such that for all $w$, $w \in A \leftrightarrow f(w) \in B$.

If $A \leqslant_P B$, then $B$ is "more expressive" — it can simulate all problems in $A$ with a polynomial overhead.

We would expect that $B$ has the more difficult decision problem.

# Hardness

So far, we have talked about memebership in a class.

But even though some NP problems seem harder, P $\subseteq$ NP.

Absent a proof that P $\neq$ NP, we can still talk about the hardest problems in a class using reductions (if $A \leqslant_{\mathrm{P}} \mathrm{B}$, then B is at least as hard as $A$).

# Completeness

Given a type of reduction $\leqslant_X$, consider the following definition.

**Definition ($\leqslant_X$-completeness)**

A language $A$ is $\leqslant_X$-hard for a class $\mathrm{C}$ if and only if $\mathrm{B} \leqslant_X A$ for every $\mathrm{B} \in \mathrm{C}$. If $A$ is also in $\mathrm{C}$, it is $\leqslant_X$-complete.

Choice of $\leqslant_X$ vital. Idea for NP: Polynomial-time reductions, since P is a lower class.

If I choose too powerful a reduction, everything becomes complete.

Take exponential time reductions: I can then reduce SAT to PATH!

# NP-completeness

A language $A \in \mathsf{NP}$ is NP-complete iff $A$ is $\leqslant_{\mathrm{P}}$-hard for NP.

Not obvious that any such problem exists.

Not at all obvious how to prove this property, that *all* other languages reduce to one.

# Completeness, properties

The problem is getting the first NP-complete language. If $A$ is NP-complete and $A \leqslant_P B \in \mathsf{NP}$, then $B$ is also NP-complete.

Reductions can be composed, after all.