Technische Universität Dresden

International Center for Computational Logic

COMPLEXITY THEORY

Lecture 6: Nondeterministic Polynomial Time

Sergei Obiedkov

Knowledge-Based Systems

TU Dresden, 28 Oct 2025

Polynomial-Time Reductions

Polynomial-Time Reductions

As for decidability we can use reductions to show membership in PTime.

Definition 6.1: A language $\mathbf{L_1} \subseteq \Sigma^*$ is polynomially many-one reducible to $\mathbf{L_2} \subseteq \Sigma^*$, denoted $\mathbf{L_1} \leq_p \mathbf{L_2}$, if there is a polynomial-time computable function f such that for all $w \in \Sigma^*$

$$w \in \mathbf{L_1} \quad \text{if and only if} \quad f(w) \in \mathbf{L_2}.$$

Polynomial-Time Reductions

As for decidability we can use reductions to show membership in PTime.

Definition 6.1: A language $\mathbf{L_1} \subseteq \Sigma^*$ is polynomially many-one reducible to $\mathbf{L_2} \subseteq \Sigma^*$, denoted $\mathbf{L_1} \leq_p \mathbf{L_2}$, if there is a polynomial-time computable function f such that for all $w \in \Sigma^*$

$$w \in \mathbf{L_1} \qquad \text{if and only if} \qquad f(w) \in \mathbf{L_2}.$$

Theorem 6.2: If $\mathbf{L_1} \leq_p \mathbf{L_2}$ and $\mathbf{L_2} \in$ PTime then $\mathbf{L_1} \in$ PTime.

Proof: The sum and composition of polynomials is a polynomial. □

Example: Colourability

Definition 6.3 (Vertex Colouring): A vertex colouring of G with k colours is a function

$$c : V(G) \longrightarrow \{1, \ldots, k\}$$

such that adjacent nodes have different colours, that is:

$$\{u, v\} \in E(G) \text{ implies } c(u) \neq c(v)$$

k-COLOURING

Input: Graph G, $k \in \mathbb{N}$

Problem: Does G have a vertex colouring with k colours?

For $k = 2$ this is the same as **BIPARTITE**.

Reducing 2-Colourability to 2-Sat

Theorem 6.4: 2-**Colourability** $\leq_p$ 2-**Sat**, and therefore 2-**Colourability** $\in$ P.

Reducing 2-Colourability to 2-Sat

Theorem 6.4: 2-**Colourability** $\leq_p$ 2-**Sat**, and therefore 2-**Colourability** $\in$ P.

Proof: We define a reduction as follows: Given graph G

- For each vertex $v \in V(G)$ of the graph introduce new variable X_v
- For each $\{u, v\} \in E(G)$ add clauses $(X_u \vee X_v)$ and $(\neg X_u \vee \neg X_v)$

This is obviously computable in polynomial time.

Reducing 2-Colourability to 2-Sat

Theorem 6.4: 2-**Colourability** $\leq_p$ 2-**Sat**, and therefore 2-**Colourability** $\in$ P.

Proof: We define a reduction as follows: Given graph G

- For each vertex $v \in V(G)$ of the graph introduce new variable X_v
- For each $\{u, v\} \in E(G)$ add clauses $(X_u \vee X_v)$ and $(\neg X_u \vee \neg X_v)$

This is obviously computable in polynomial time.

We check that it is a reduction:

- If G is 2-colourable, use colouring to assign truth values.
  (One colour is true, the other false)
- If the formula is satisfiable, the truth assignment defines valid 2-colouring.
  For every edge $\{u, v\} \in E(G)$, one variable X_u, X_v is set to true, the other to false.

□

Reductions in PTime

All non-trivial members of PTime can be reduced to each other:

Theorem 6.5: If **B** is any language in P, **B** $\neq \emptyset$, and **B** $\neq \Sigma^*$, then **A** $\leq_p$ **B** for any **A** $\in$ P.

Reductions in PTime

All non-trivial members of PTime can be reduced to each other:

Theorem 6.5: If **B** is any language in P, **B** $\neq \emptyset$, and **B** $\neq \Sigma^*$, then **A** $\leq_p$ **B** for any **A** $\in$ P.

Proof: Choose $w \in$ **B** and $w' \notin$ **B**.

Reductions in PTime

All non-trivial members of PTime can be reduced to each other:

Theorem 6.5: If **B** is any language in P, $\mathbf{B} \neq \emptyset$, and $\mathbf{B} \neq \Sigma^*$, then $\mathbf{A} \leq_p \mathbf{B}$ for any $\mathbf{A} \in$ P.

Proof: Choose $w \in \mathbf{B}$ and $w' \notin \mathbf{B}$.

Define the function f by setting

$$f(x) := \begin{cases} w & \text{if } x \in \mathbf{A} \\ w' & \text{if } x \notin \mathbf{A} \end{cases}$$

Since $\mathbf{A} \in$ P, this function f is computable in polynomial time, and it is a reduction from **A** to **B**. □

Reducing 2-Colourability to 2-Sat

Theorem 6.6: 2-**Colourability** $\leq_p$ 2-**Sat**.

Reducing 2-Colourability to 2-Sat

Theorem 6.6: 2-**Colourability** $\leq_p$ 2-**Sat**.

Proof: 2-Colourability is the same as **Bipartite**. Hence, 2-**Colourability** $\in$ P and 2-**Colourability** $\leq_p$ 2-**Sat** by Theorem 6.5.

Reducing 2-Colourability to 2-Sat

Theorem 6.6: 2-**Colourability** $\leq_p$ 2-**Sat**.

Proof: 2-Colourability is the same as **Bipartite**. Hence, 2-**Colourability** $\in$ P and 2-**Colourability** $\leq_p$ 2-**Sat** by Theorem 6.5.

In more detail: Define the function f by setting

$$f(G) := \begin{cases} X \vee Y & \text{if } G \text{ is bipartite} \\ X \wedge \neg X & \text{if } G \text{ is not bipartite} \end{cases}$$

Since **Bipartite** $\in$ P, this function f is computable in polynomial time, and it is a reduction from 2-**Colourability** to 2-**Sat**. □

Trivially Tractable Problems

A large class of languages is generally tractable:

Theorem 6.7: If **L** is a finite language, then it is decided by an $O(1)$-time bounded TM. In other words, all finite languages are decidable in constant time (and hence also in polynomial time).

Trivially Tractable Problems

A large class of languages is generally tractable:

Theorem 6.7: If **L** is a finite language, then it is decided by an $O(1)$-time bounded TM. In other words, all finite languages are decidable in constant time (and hence also in polynomial time).

Proof:

- As **L** is finite, there is a maximum length m of words in **L**.
- Read the input up to the first m letters.
- The state space contains a table containing the correct result for all such inputs.
- All other inputs are rejected. □

Trivially Tractable Problems

A large class of languages is generally tractable:

Theorem 6.7: If **L** is a finite language, then it is decided by an $O(1)$-time bounded TM. In other words, all finite languages are decidable in constant time (and hence also in polynomial time).

Proof:

- As **L** is finite, there is a maximum length m of words in **L**.
- Read the input up to the first m letters.
- The state space contains a table containing the correct result for all such inputs.
- All other inputs are rejected. □

Example 6.8: The following problem is solvable in constant time:
Given a position on a standard 8×8 chessboard, decide if the White has a winning strategy.

A Note on Constructiveness

The next result is an example of a theorem that proves the existence of a P algorithm in cases where we do not know what this algorithm is.

Example 6.9: Let **L** be the language that contains all correct sentences from the following set:

$$\{\text{“P is the same as NP”}, \text{“P is not the same as NP”}\}$$

Then **L** is decidable in constant time.

A Note on Constructiveness

The next result is an example of a theorem that proves the existence of a P algorithm in cases where we do not know what this algorithm is.

Example 6.9: Let **L** be the language that contains all correct sentences from the following set:

$$\{\text{“P is the same as NP”}, \text{“P is not the same as NP”}\}$$

Then **L** is decidable in constant time.
However, we don't know which constant-time algorithm decides it.

A Note on Constructiveness

The next result is an example of a theorem that proves the existence of a P algorithm in cases where we do not know what this algorithm is.

Example 6.9: Let **L** be the language that contains all correct sentences from the following set:

$$\{\text{“P is the same as NP”}, \text{“P is not the same as NP”}\}$$

Then **L** is decidable in constant time.
However, we don't know which constant-time algorithm decides it.

Non-constructiveness:

- We can prove that there is a correct polynomial time algorithm.
- We cannot construct such an algorithm.

  Such solutions are called non-constructive.

The Class NP

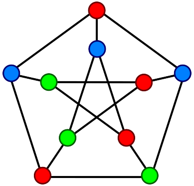
Beyond PTime

- We have seen that the class PTime provides a useful model of “tractable” problems
- This includes 2-Sat and 2-Colourability
- But what about 3-Sat and 3-Colourability?
- No polynomial time algorithms for these problems are known
- On the other hand . . .

Verifying Solutions

For many seemingly difficult problems, it is easy to verify the correctness of a “solution” if given.

p	q	r	$p \to q$
f	f	f	w
f	w	f	w
w	f	f	f
w	w	f	w
f	f	w	w
f	w	w	w
w	f	w	f
w	w	w	w



5		3				7		
			8					6
	7			6			4	
	4		1					
7		8		5		3		9
					9		6	
	5			1			7	
6					4			
		2				5		3

- Satisfiability – a satisfying assignment
- k-Colourability – a k-colouring
- Sudoku – a completed puzzle

Verifiers

Definition 6.10: A Turing machine $\mathcal{M}$ that halts on all inputs is called a verifier for a language **L** if

$$\mathbf{L} = \{w \mid \mathcal{M} \text{ accepts } (w\#c) \text{ for some string } c\}$$

The string c is called a certificate (or witness) for w.

Notation: # is a new separator symbol not used in words or certificates.

Verifiers

Definition 6.10: A Turing machine $\mathcal{M}$ that halts on all inputs is called a verifier for a language **L** if

$$\mathbf{L} = \{w \mid \mathcal{M} \text{ accepts } (w\#c) \text{ for some string } c\}$$

The string c is called a certificate (or witness) for w.

Notation: # is a new separator symbol not used in words or certificates.

Definition 6.11: A Turing machine $\mathcal{M}$ is a polynomial-time verifier for **L** if $\mathcal{M}$ is polynomial-time bounded and

$$\mathbf{L} = \{w \mid \mathcal{M} \text{ accepts } (w\#c) \text{ for some string } c \text{ with } |c| \leq p(|w|)\}$$

for some fixed polynomial p.

The Class NP

NP: “The class of dashed hopes and idle dreams.”[1]

[1] https://complexityzoo.net/Complexity_Zoo:N#np

The Class NP

NP: “The class of dashed hopes and idle dreams.”[1]

More formally:
the class of problems for which a possible solution can be verified in polynomial time

Definition 6.12: The class of languages that have polynomial-time verifiers is called NP.

[1] https://complexityzoo.net/Complexity_Zoo:N#np

The Class NP

NP: “The class of dashed hopes and idle dreams.”[1]

More formally:
the class of problems for which a possible solution can be verified in polynomial time

Definition 6.12: The class of languages that have polynomial-time verifiers is called NP.

In other words: NP is the class of all languages **L** such that:

- for every $w \in \mathbf{L}$, there are one or more certificates $C_w \subseteq \Sigma^*$, where
- the length of each $c \in C_w$ is polynomial in the length of w, and
- the language $\{(w\#c) \mid w \in \mathbf{L}, c \in C_w\}$ is in P

[1] https://complexityzoo.net/Complexity_Zoo:N#np

More Examples of Problems in NP

Hamiltonian Path

Input: An undirected graph G

Problem: Is there a path in G that contains each vertex exactly once?

k-Clique

Input: An undirected graph G and an integer k

Problem: Does G contain a fully connected graph (clique) with k vertices?

More Examples of Problems in NP

SUBSET SUM

Input: A collection of positive integers $S = \{a_1, \ldots, a_k\}$ and a target integer t

Problem: Is there a subset $T \subseteq S$ such that $\sum_{a_i \in T} a_i = t$?

TRAVELLING SALESPERSON

Input: A weighted graph G and a target number t

Problem: Is there a simple path in G with weight $\leq t$ that contains each vertex exactly once?

Complements of NP are often not known to be in NP

No Hamiltonian Path

Input:	An undirected graph G
Problem:	Is there no path in G that contains each vertex exactly once?

Whereas it is easy to certify that a graph has a Hamiltonian path, there does not seem to be a polynomial certificate that it has not.

But we may just not be clever enough to find one.

More Examples

Composite (non-prime) Number

Input: A positive integer $n > 1$

Problem: Are there integers $u, v > 1$ such that $u \cdot v = n$?

Prime Number

Input: A positive integer $n > 1$

Problem: Is n a prime number?

More Examples

Composite (non-prime) Number

Input: A positive integer $n > 1$

Problem: Are there integers $u, v > 1$ such that $u \cdot v = n$?

Prime Number

Input: A positive integer $n > 1$

Problem: Is n a prime number?

Surprisingly: both are in NP (see Wikipedia “Primality certificate”)

More Examples

COMPOSITE (NON-PRIME) NUMBER

Input:	A positive integer $n > 1$
Problem:	Are there integers $u, v > 1$ such that $u \cdot v = n$?

PRIME NUMBER

Input:	A positive integer $n > 1$
Problem:	Is n a prime number?

Surprisingly: both are in NP (see Wikipedia "Primality certificate")

In fact: Composite Number (and thus Prime Number) was shown to be in P

N is for Nondeterministic

Reprise: Nondeterministic Turing Machines

A nondeterministic Turing Machine (NTM) $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}})$ consists of

- a finite set Q of **states**,
- an **input alphabet** Σ not containing $\llcorner\!\lrcorner$,
- a **tape alphabet** Γ such that $\Gamma \supseteq \Sigma \cup \{\llcorner\!\lrcorner\}$.
- a **transition function** $\delta \colon Q \times \Gamma \to 2^{Q \times \Gamma \times \{\mathsf{L},\mathsf{R}\}}$
- an **initial state** $q_0 \in Q$,
- an **accepting state** $q_{\text{accept}} \in Q$.

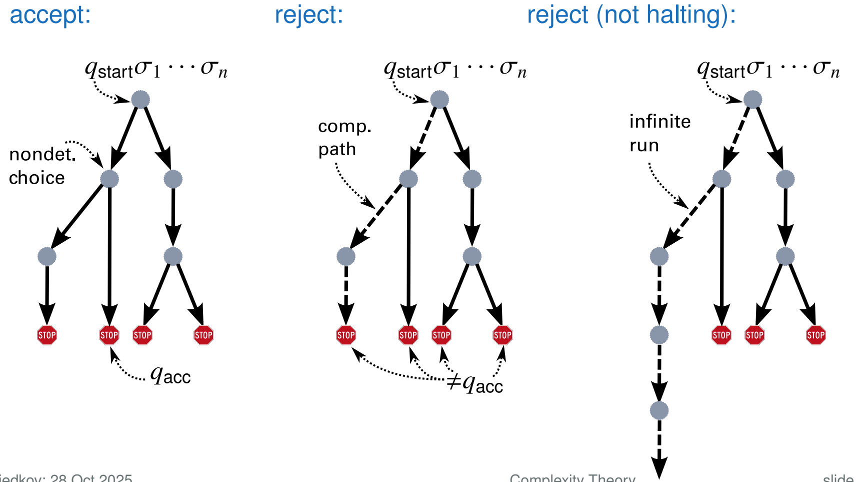
Note

An NTM can halt in any state if there are no options to continue
$\leadsto$ no need for a special rejecting state

Reprise: Runs of NTMs

An (N)TM configuration can be written as a word uqv where $q \in Q$ is a state and $uv \in \Gamma^*$ is the current tape contents.

NTMs produce configuration trees that contain all possible runs:

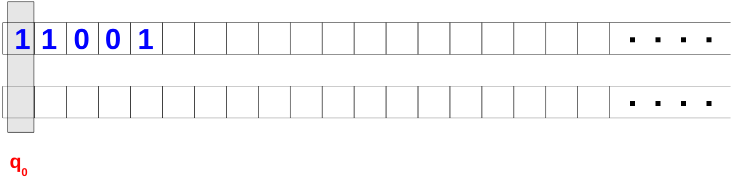


Example: Multi-Tape NTM

Consider the NTM $\mathcal{M} = (Q, \{0,1\}, \{0,1,\sqcup\}, \delta, q_0, q_{\text{accept}})$ where

$$\delta = \left\{ \begin{array}{l} (q_0, \binom{-}{-}, q_0, \binom{-}{0}, \binom{N}{R}) \\ (q_0, \binom{-}{-}, q_0, \binom{-}{1}, \binom{N}{R}) \\ (q_0, \binom{-}{-}, q_{\text{check}}, \binom{-}{-}, \binom{N}{N}) \\ \ldots \\ \text{transition rules for } \mathcal{M}_{\text{check}} \end{array} \right\}$$

and where $\mathcal{M}_{\text{check}}$ is a deterministic TM deciding whether the number on second tape is > 1 and divides the number on the first.

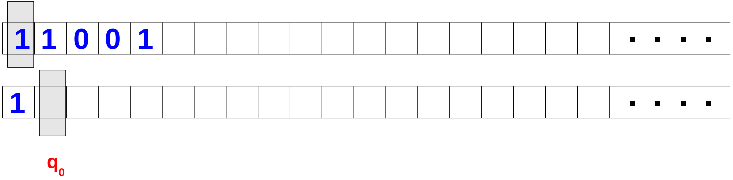


Example: Multi-Tape NTM

Consider the NTM $\mathcal{M} = (Q, \{0,1\}, \{0,1,\sqcup\}, \delta, q_0, q_{\text{accept}})$ where

$$\delta = \left\{ \begin{array}{l} (q_0,\ \binom{-}{-}, q_0, \binom{-}{0}, \binom{N}{R}) \\ (q_0,\ \binom{-}{-}, q_0, \binom{-}{1}, \binom{N}{R}) \\ (q_0,\ \binom{-}{-}, q_{\text{check}}, \binom{-}{-}, \binom{N}{N}) \\ \ldots \\ \text{transition rules for } \mathcal{M}_{\text{check}} \end{array} \right\}$$

and where $\mathcal{M}_{\text{check}}$ is a deterministic TM deciding whether the number on second tape is > 1 and divides the number on the first.

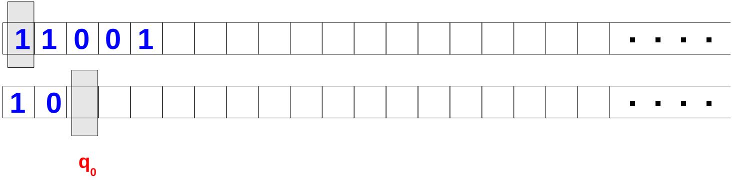


Example: Multi-Tape NTM

Consider the NTM $\mathcal{M} = (Q, \{0,1\}, \{0,1,\sqcup\}, \delta, q_0, q_{\text{accept}})$ where

$$\delta = \left\{ \begin{array}{l} \left(q_0,\ \binom{-}{-}, q_0, \binom{-}{0}, \binom{N}{R}\right) \\ \left(q_0,\ \binom{-}{-}, q_0, \binom{-}{1}, \binom{N}{R}\right) \\ \left(q_0,\ \binom{-}{-}, q_{\text{check}}, \binom{-}{-}, \binom{N}{N}\right) \\ \ldots \\ \text{transition rules for } \mathcal{M}_{\text{check}} \end{array} \right\}$$

and where $\mathcal{M}_{\text{check}}$ is a deterministic TM deciding whether the number on second tape is > 1 and divides the number on the first.

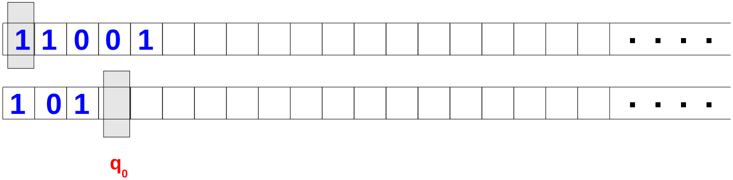


Example: Multi-Tape NTM

Consider the NTM $\mathcal{M} = (Q, \{0,1\}, \{0,1,\sqcup\}, \delta, q_0, q_{\text{accept}})$ where

$$\delta = \left\{ \begin{array}{l} \left(q_0,\ \binom{\sqcup}{\sqcup}, q_0, \binom{\sqcup}{0}, \binom{N}{R}\right) \\ \left(q_0,\ \binom{\sqcup}{\sqcup}, q_0, \binom{\sqcup}{1}, \binom{N}{R}\right) \\ \left(q_0,\ \binom{\sqcup}{\sqcup}, q_{\text{check}}, \binom{\sqcup}{\sqcup}, \binom{N}{N}\right) \\ \dots \\ \text{transition rules for } \mathcal{M}_{\text{check}} \end{array} \right\}$$

and where $\mathcal{M}_{\text{check}}$ is a deterministic TM deciding whether the number on second tape is > 1 and divides the number on the first.



Example: Multi-Tape NTM

Consider the NTM $\mathcal{M} = (Q, \{0, 1\}, \{0, 1, \sqcup\}, q_0, \Delta, q_{\text{accept}})$ where

$$\Delta = \left\{ \begin{array}{l} (q_0, \binom{-}{-}, q_0, \binom{-}{0}, \binom{N}{R}) \\ (q_0, \binom{-}{-}, q_0, \binom{-}{1}, \binom{N}{R}) \\ (q_0, \binom{-}{-}, q_{\text{check}}, \binom{-}{-}, \binom{N}{N}) \\ \ldots \\ \text{transition rules for } \mathcal{M}_{\text{check}} \end{array} \right\}$$

and where $\mathcal{M}_{\text{check}}$ is a deterministic TM deciding whether number on second tape is > 1 and divides the number on the first.

The machine $\mathcal{M}$ recognizes if the input is a composite number:

- guess a number on the second tape
- check if it divides the number on the first tape
- accept if a suitable number exists

Time- and Space-Bounded NTMs

Q: Which of the nondeterministic runs do time/space bounds apply to?

Time- and Space-Bounded NTMs

Q: Which of the nondeterministic runs do time/space bounds apply to?
A: To all of them!

Definition 6.13: Let $\mathcal{M}$ be a nondeterministic Turing machine and let $f : \mathbb{N} \to \mathbb{R}^+$ be a function.

(1) $\mathcal{M}$ is f-time bounded if it halts on every input $w \in \Sigma^*$ and on every computation path after $\leq f(|w|)$ steps.

(2) $\mathcal{M}$ is f-space bounded if it halts on every input $w \in \Sigma^*$ and on every computation path using $\leq f(|w|)$ cells on its tapes.

(Here we typically assume that Turing machines have a separate input tape that we do not count in measuring space complexity.)

Nondeterministic Complexity Classes

Definition 6.14: Let $f : \mathbb{N} \rightarrow \mathbb{R}^+$ be a function.

(1) $\mathsf{NTime}(f(n))$ is the class of all languages **L** for which there is an $O(f(n))$-time bounded nondeterministic Turing machine deciding **L**.

(2) $\mathsf{NSpace}(f(n))$ is the class of all languages **L** for which there is an $O(f(n))$-space bounded nondeterministic Turing machine deciding **L**.

All Complexity Classes Have a Nondeterministic Variant

$$\text{NPTime} = \bigcup_{d \geq 1} \text{NTime}(n^d)$$ nondet. polynomial time

$$\text{NExp} = \text{NExpTime} = \bigcup_{d \geq 1} \text{NTime}(2^{n^d})$$ nondet. exponential time

$$\text{N2Exp} = \text{N2ExpTime} = \bigcup_{d \geq 1} \text{NTime}(2^{2^{n^d}})$$ nond. double-exponential time

$$\text{NL} = \text{NLogSpace} = \text{NSpace}(\log n)$$ nondet. logarithmic space

$$\text{NPSpace} = \bigcup_{d \geq 1} \text{NSpace}(n^d)$$ nondet. polynomial space

$$\text{NExpSpace} = \bigcup_{d \geq 1} \text{NSpace}(2^{n^d})$$ nondet. exponential space

Equivalence of NP and NPTime

Theorem 6.15: NP = NPTime.

Equivalence of NP and NPTime

Theorem 6.15: NP = NPTime.

Proof: We first show NP $\supseteq$ NPTime:

Equivalence of NP and NPTime

Theorem 6.15: NP = NPTime.

Proof: We first show NP $\supseteq$ NPTime:

- Suppose $\mathbf{L} \in$ NPTime.
- Then there is an NTM $\mathcal{M}$ such that

$$w \in \mathbf{L} \iff \text{there is an accepting run of } \mathcal{M} \text{ of length } O(n^d)$$

for some d.

Equivalence of NP and NPTime

Theorem 6.15: NP = NPTime.

Proof: We first show NP $\supseteq$ NPTime:

- Suppose **L** $\in$ NPTime.
- Then there is an NTM $\mathcal{M}$ such that

$$w \in \mathbf{L} \iff \text{there is an accepting run of } \mathcal{M} \text{ of length } O(n^d)$$

  for some d.
- This path can be used as a certificate for w.

Equivalence of NP and NPTime

Theorem 6.15: NP = NPTime.

Proof: We first show NP $\supseteq$ NPTime:

- Suppose **L** $\in$ NPTime.
- Then there is an NTM $\mathcal{M}$ such that

$$w \in \mathbf{L} \iff \text{there is an accepting run of } \mathcal{M} \text{ of length } O(n^d)$$

  for some d.
- This path can be used as a certificate for w.
- A DTM can check in polynomial time that a candidate certificate is a valid accepting run.

Therefore NP $\supseteq$ NPTime.

Equivalence of NP and NPTime

Theorem 6.15: NP = NPTime.

Proof: We now show NP $\subseteq$ NPTime:

Equivalence of NP and NPTime

Theorem 6.15: NP = NPTime.

Proof: We now show NP $\subseteq$ NPTime:

- Assume **L** has a polynomial-time verifier $\mathcal{M}$ with certificates of length at most $p(n)$ for a polynomial p.

Equivalence of NP and NPTime

Theorem 6.15: NP = NPTime.

Proof: We now show NP $\subseteq$ NPTime:

- Assume **L** has a polynomial-time verifier $\mathcal{M}$ with certificates of length at most $p(n)$ for a polynomial p.
- Then we can construct an NTM $\mathcal{M}^*$ deciding **L** as follows:
  (1) $\mathcal{M}^*$ guesses a string of length $p(n)$
  (2) $\mathcal{M}^*$ checks in deterministic polynomial time if this is a certificate.

Therefore NP $\subseteq$ NPTime. □

NP and coNP

Note: the definition of NP is not symmetric

- there does not seem to be any polynomial certificate for Sudoku **un**solvability or propositional logic **un**satisfiability
- the converse of an NP problem is in coNP
- similar for NExpTime and N2ExpTime

Some other complexity classes are symmetric:

- Deterministic classes (e.g., coP = P)
- Space classes mentioned above (e.g., coNL = NL)

Deterministic vs. Nondeterminsitic Time

Theorem 6.16: P $\subseteq$ NP, and also P $\subseteq$ coNP.

(Clear since DTMs are a special case of NTMs)

Deterministic vs. Nondeterminsitic Time

Theorem 6.16: P $\subseteq$ NP, and also P $\subseteq$ coNP.

(Clear since DTMs are a special case of NTMs)

It is not known to date if the converse is true or not.

- Put differently: “If it is easy to check a candidate solution to a problem, is it also easy to find one?”

Deterministic vs. Nondeterminsitic Time

Theorem 6.16: P $\subseteq$ NP, and also P $\subseteq$ coNP.

(Clear since DTMs are a special case of NTMs)

It is not known to date if the converse is true or not.

- Put differently: “If it is easy to check a candidate solution to a problem, is it also easy to find one?”
- Exaggerated: “Can creativity be automated?” (Wigderson, 2006)

Deterministic vs. Nondeterminsitic Time

Theorem 6.16: P $\subseteq$ NP, and also P $\subseteq$ coNP.

(Clear since DTMs are a special case of NTMs)

It is not known to date if the converse is true or not.

- Put differently: "If it is easy to check a candidate solution to a problem, is it also easy to find one?"
- Exaggerated: "Can creativity be automated?" (Wigderson, 2006)
- Unresolved after more than 50 years of effort
- One of the major problems in computer science and math of our time
- 1,000,000 USD prize for resolving it ("Millenium Problem")
  (might not be much money at the time it is actually solved)

Status of P vs. NP

Many people believe that P $\neq$ NP

- Main argument: "If NP = P, someone ought to have found some polynomial algorithm for an NP-complete problem by now."
- "This is, in my opinion, a very weak argument. The space of algorithms is very large and we are only at the beginning of its exploration." (Moshe Vardi, 2002)
- Another source of intuition: Humans find it hard to solve NP-complete problems and hard to imagine how to make them simpler—possibly "human chauvinistic bravado" (Zeilenberger, 2006)
- There are better arguments, but none go beyond intuition

Status of P vs. NP

Many outcomes conceivable:

Status of P vs. NP

Many outcomes conceivable:

- P $=$ NP could be shown with a non-constructive proof

Status of P vs. NP

Many outcomes conceivable:

- P = NP could be shown with a non-constructive proof
- The question might be independent of standard mathematics (ZFC)

Status of P vs. NP

Many outcomes conceivable:

- P $=$ NP could be shown with a non-constructive proof
- The question might be independent of standard mathematics (ZFC)
- Even if P $\neq$ NP, it is unclear if NP problems require exponential time in a strict sense: many super-polynomial functions exist

Status of P vs. NP

Many outcomes conceivable:

- $P = NP$ could be shown with a non-constructive proof
- The question might be independent of standard mathematics (ZFC)
- Even if $P \neq NP$, it is unclear if NP problems require exponential time in a strict sense: many super-polynomial functions exist
- The problem might never be solved

Status of P vs. NP

Results of a 2019 poll among 124 experts, together with results of previous surveys [Gasarch 2019]:

	$P \neq NP$	$P = NP$	Ind	DC	DK	DK and DC	other
2002	61 (61%)	9 (9%)	4 (4%)	1 (1%)	22 (22%)	0	3 (3%)
2012	126 (83%)	12 (9%)	5 (3%)	5 (3%)	1 (0.66%)	1 (0.66%)	1 (0.66%)
2019	109 (88%)	15 (12%)	0	0	0	0	0

Ind: independent (of ZFC), DC: don't care, DK: don't know

- Lance Fortnow: "People that think P=NP are like people who think Elvis is still alive."
- Experts have guessed wrongly in other major questions before
- Over 100 "proofs" show $P = NP$ to be true/false/both/neither: `https://www.win.tue.nl/~gwoegi/P-versus-NP.htm`

A Simple Proof for P = NP

Clearly	$\mathbf{L} \in \mathrm{P}$	implies	$\mathbf{L} \in \mathrm{NP}$
therefore	$\mathbf{L} \notin \mathrm{NP}$	implies	$\mathbf{L} \notin \mathrm{P}$
hence	$\mathbf{L} \in \mathrm{coNP}$	implies	$\mathbf{L} \in \mathrm{coP}$
that is	$\mathrm{coNP} \subseteq \mathrm{coP}$		
using coP = P	$\mathrm{coNP} \subseteq \mathrm{P}$		
and hence	$\mathrm{NP} \subseteq \mathrm{P}$		
so by $\mathrm{P} \subseteq \mathrm{NP}$	$\mathrm{NP} = \mathrm{P}$		

q.e.d.

A Simple Proof for P = NP

Clearly	$\mathbf{L} \in \mathrm{P}$	implies	$\mathbf{L} \in \mathrm{NP}$
therefore	$\mathbf{L} \notin \mathrm{NP}$	implies	$\mathbf{L} \notin \mathrm{P}$
hence	$\mathbf{L} \in \mathrm{coNP}$	implies	$\mathbf{L} \in \mathrm{coP}$
that is	$\mathrm{coNP} \subseteq \mathrm{coP}$		
using $\mathrm{coP} = \mathrm{P}$	$\mathrm{coNP} \subseteq \mathrm{P}$		
and hence	$\mathrm{NP} \subseteq \mathrm{P}$		
so by $\mathrm{P} \subseteq \mathrm{NP}$	$\mathrm{NP} = \mathrm{P}$		

q.e.d.?

Summary and Outlook

NP can be defined using polynomial-time verifiers or polynomial-time nondeterministic Turing machines

Many problems are easily seen to be in NP

NTM acceptance is not symmetric: coNP as complement class, which is assumed to be unequal to NP

What's next?

- NP hardness and completeness
- More examples of problems
- Space complexities