Agenda

- Polynomial-time verifiers
- Examples of polynomial-time verifiers
- The language class NP
- Nondeterministic Turing Machines
- NP-completeness
Travelling Salesman Problem

Travelling Salesman Problem (Decision Version)

- **Instance:** Graph $G = (V, E)$ with positive edge weights, integer $W \geq 0$, a vertex $v \in V$.
- **Question:** Is there a tour starting from vertex $v$ that visits all other vertices exactly once and then returns to $v$ with weight at most $W$?
Travelling Salesman Problem

- We don’t know how to solve TSP in polynomial time
- We can **verify** the correctness of a solution:
  - **Solution**: a tour $T = (v_1, v_2, \ldots, v_n)$
  - **Verification**: check that $T$ is a valid tour, $T$ visits all vertices once, and has weight at most $W$
  - Verification takes polynomial time
k-colouring

Definition

Let $k$ be a fixed positive integer, and let $G = (V, E)$ be an undirected graph. A $k$-colouring of $G$ is a function

$$c : V \rightarrow \{1, 2, \ldots, k\}$$

such that for adjacent vertices $v$ and $u$, we have $c(v) \neq c(u)$.

$k$-colouring problem ($k$-COL)

- **Instance:** Graph $G = (V, E)$.
- **Question:** Is there a $k$-colouring of $G$?
**k-colouring**

- We don’t know how to solve *k*-colouring in polynomial time.
- We can **verify** the correctness of a solution:
  - **Solution**: a *k*-colouring \( c : V : \{1, 2, \ldots, k\} \)
  - **Verification**: check that for all edges \( \{u, v\} \in E \), we have \( c(u) \neq c(v) \)
  - Verification takes polynomial time
Polynomial-time Verifiers

Definition (Polynomial-time Verifier)
Let $L \subseteq \{0, 1\}^*$. A polynomial-time verifier for $L$ is a polynomial-time Turing machine $M$ such that for some polynomial function $p : \mathbb{N} \rightarrow \mathbb{N}$ the following holds:

- if $x \in L$, there is a string $u \in \{0, 1\}^*$ with $|u| \leq p(|x|)$ so that $M((x, u)) = 1$, and
- if $x \notin L$, we have $M((x, u)) = 0$ for all $u \in \{0, 1\}^*$.

If $M((x, u)) = 1$, we call $u$ the certificate or witness for $x$. 

Maximum Independent Set

Maximum independent set (MaxIS)

- **Instance**: Graph $G = (V, E)$ and an integer $k \geq 1$.
- **Question**: Is there a set of vertices $I$ such that $|I| \geq k$ and for all $u, v \in I$, we have that $\{u, v\} \notin E$?
Maximum Independent Set

- **Certificate:** A vertex set $I \subseteq V$ of size $k$ ($O(k \log n)$ bits)
- **Verifier:**
  - Check that $I$ has correct size
  - Check that for each edge $\{u, v\} \in V$, either $u \notin I$ or $v \notin I
Subset Sum

Subset sum

- **Instance:** A list of integers \(a_1, a_2, \ldots, a_n\) and an integer \(T\).
- **Question:** Is there a subset of the input list that sums up to \(T\)?

- **Certificate:** A subset \(S\) of the input list
- **Verifier:**
  - Check that \(S\) is a valid subset of input
  - Compute the sum of \(S\) and check that it is \(T\)
Composite Numbers

Composite number

- **Instance**: An integer $N$.
- **Question**: Are there numbers $p$ and $q$ with $p, q \notin \{1, N\}$ such that $pq = N$? (That is, $N$ is not a prime number.)

- **Certificate**: Numbers $p$ and $q$ ($O(\log |N|)$ bits)
- **Verifier**: Check that $pq = N$
Connectivity

Instance: Graph $G = (V, E)$, two vertices $s$ and $t$.

Question: Is there a path from $s$ to $t$ in $G$?

Certificate: A path $P = (v_1, v_2, \ldots, v_k)$ in graph $G$

Verifier: Check that $P$ is a valid path from $s$ to $t$
Connectivity

Instance: Graph $G = (V, E)$, two vertices $s$ and $t$.

Question: Is there a path from $s$ to $t$ in $G$?

Certificate: A path $P = (v_1, v_2, \ldots, v_k)$ in graph $G$

Verifier: Check that $P$ is a valid path from $s$ to $t$
Definition (NP)

The class NP is the class of all languages $L \subseteq \{0, 1\}^*$ that have a polynomial-time verifier.
NP and Other Classes

Consider the following time complexity classes:

- **Polynomial time**: $P = \bigcup_{d=1}^{\infty} \text{DTIME}(n^d)$
- **Exponential time**: $\text{EXP} = \bigcup_{d=1}^{\infty} \text{DTIME}(2^{n^d})$

**Theorem**

$P \subseteq \text{NP} \subseteq \text{EXP}$

**Proof ($P \subseteq \text{NP}$):**
- Use length-0 string $\epsilon$ as certificate

**Proof ($\text{NP} \subseteq \text{EXP}$):**
- Try all possible certificates of length $p(|x|)$
- $O(2^{p(n)})$ possibilities + checking
P vs NP

- Do all polynomial-time verifiable problems also have polynomial-time algorithms?
  - Formally: does it hold that $P = NP$?
  - This is the famous *P vs. NP question*
  - This seems to be a really difficult problem

- In practice, we tend to assume $P \neq NP$
  - We will use this assumption to prove that certain problems are difficult
  - Gives *conditional lower bounds*
Nondeterministic Turing Machines

- We give an alternative definition of NP in terms of polynomial-time nondeterministic Turing machines
  - NP stands for nondeterministic polynomial time

- NDTM is an abstract model of computation
  - Does not correspond to any physical method of computation
  - Purely a conceptual tool
  - Can be viewed as an abstraction of computation that tries all possible solutions
Nondeterministic Turing Machines

- A *nondeterministic Turing machine* $M$ is a Turing machine with following special features:
  - $M$ has a special *accept state* $q_{\text{accept}}$
  - $M$ has two *transition functions* $\delta_1$ and $\delta_2$
  - $M$ does not have an output tape

- An *execution* of nondeterministic Turing machine $M$:
  - Start from the starting state as usual
  - Apply *either* $\delta_1$ or $\delta_2$ at each step
  - Halt when reaching $q_{\text{accept}}$ or $q_h$

- For each input, a NDTM has *multiple possible executions*
Nondeterministic Turing Machines

Definition

A NDTM $M$ decides language $L$ in time $T(n)$ if:

- For any $x \in L$, there is at least one execution on input $x$ that reaches state $q_{\text{accept}}$
- For any $x \notin L$, all executions halt without entering $q_{\text{accept}}$
- All executions on input $x \in \{0, 1\}^*$ run for at most $T(|x|)$ steps
Nondeterministic Time Complexity

**Definition (Class NTIME)**

Let $T : \mathbb{N} \rightarrow \mathbb{N}$ be a function. The class $\text{NTIME}(T(n))$ is the set of languages $L$ for which there exists a nondeterministic Turing machine $M$ and a constant $c > 0$ such that $M$ decides $L$ and runs in time $c \cdot T(n)$. 
NP: Alternative Definition

**Theorem**

\[ \text{NP} = \bigcup_{d=1}^{\infty} \text{NTIME}(n^d). \]

**Proof** \((\bigcup_{d=1}^{\infty} \text{NTIME}(n^d) \subseteq \text{NP}):\)

- Let \( L \in \bigcup_{d=1}^{\infty} \text{NTIME}(n^d) \)
- **We have:** \( p(n) \)-time NDTM \( M \) for \( L \), where \( p \) is polynomial
- **We want:** Polynomial-time verifier \( M' \) for \( L \)
- For any \( x \in L \), \( M \) has an accepting execution
- **Certificate:** a string \( u \in \{0, 1\}^{p(|x|)} \)
- **Verifier:** Simulate \( M \), use \( u \) to choose which transition function to use \((0 \rightarrow \delta_1, 1 \rightarrow \delta_2)\), check that the execution ends in \( q_{\text{accept}} \)
NP: Alternative Definition

Theorem

$$\text{NP} = \bigcup_{d=1}^{\infty} \text{NTIME}(n^d).$$

Proof ($\text{NP} \subseteq \bigcup_{d=1}^{\infty} \text{NTIME}(n^d)$):

- Let $L \in \text{NP}$
- \textbf{We have:} $p(n)$-time verifier $M$ using certificates of length at most $q(n)$ for $L$, where $p, q$ are polynomial
- \textbf{We want:} Polynomial-time NDTM $M'$ for $L$

- Use nondeterminism to generate a certificate $u$ of length at most $q(|x|)$ for input $x$
- Concretely: $\delta_1$ writes 0, $\delta_2$ writes 1
- Deterministically simulate verifier $M$ with $(x, u)$, move to $q_{\text{accept}}$ if $M$ accepts
NP-hardness and NP-completeness

Definition
We say that a language $L$ is **NP-hard** if for any language $L' \in \text{NP}$, there is a polynomial-time reduction from $L'$ to $L$.

Definition
We say that $L$ is **NP-complete** if $L$ is NP-hard and $L \in \text{NP}$.
NP-hardness and NP-completeness

Theorem

- If $L$ is NP-hard and $L \in P$, then $P = NP$.
- If $L$ is NP-complete, then $L \in P$ if and only if $P = NP$.

Proof (first statement):

- Recall: $L' \leq_p L$ and $L \in P$ implies $L' \in P$
- If $L$ is NP-hard and $L \in P$, then for any language $L' \in NP$ we have $L' \leq_p L$ and thus $L' \in P$
- Thus it follows from the assumption that $NP \subseteq P$
NP-complete Languages

- **NP-complete problems are the hardest problems in NP**
  - If we believe $P \neq NP$, then NP-complete languages are not in $P$
  - Important technique for proving *conditional* lower bounds, as many interesting problems are in NP
  - On the other hand, if one NP-complete problem has a polynomial-time algorithm, then $P = NP$

- **Typical application:**
  - We have a computational problem $L$ we are interested in
  - Prove that $L$ is NP-complete and conclude there is probably no polynomial-time algorithm
An NP-complete Language

Definition (TMSAT)

- **Instance:** A tuple \((\alpha, x, 1^n, 1^t)\), where \(\alpha, x \in \{0, 1\}^*\)
- **Question:** Is there a string \(u \in \{0, 1\}^*\) with \(|u| \leq n\) such that the Turing machine \(M_\alpha\) outputs 1 on input \((x, u)\) within \(t\) steps? (\(*\))

- \(TMSAT = \{(\alpha, x, 1^n, 1^t) : \text{Condition (\(*\)) holds for } (\alpha, x, 1^n, 1^t) \}\)
An NP-complete Language

**Theorem**

TMSAT is NP-complete.

**Proof:**

(i) TMSAT ∈ NP, i.e. TMSAT has a polynomial-time verifier.

Note that

\[ |\downarrow(\alpha, x, 1^n, 1^t)\downarrow| \geq |1^n| = n \]

\[ |\downarrow(\alpha, x, 1^n, 1^t)\downarrow| \geq |1^t| = t \]

That is, \( n \) and \( t \) are polynomial in \( |\downarrow(\alpha, x, 1^n, 1^t)\downarrow| \)

- **Certificate:** a string \( u \in \{0, 1\}^* \) with \( |u| \leq n \)

- **Verification algorithm:** simulate Turing machine \( M_\alpha \) on input \( (x, u) \) for \( t \) steps, check if it halts and outputs 1
An NP-complete Language

Theorem

TMSAT is NP-complete.

Proof (cont’d):

(ii) TMSAT is NP-hard:

- Let \( L \in \text{NP} \)
- By definition, there is a verifier \( M \) for \( L \) that runs in time \( q(n) \) with certificates of size at most \( p(n) \), where \( p, q \) are polynomial
- **Reduction**: map \( x \mapsto (\bot M \bot, x, 1^p(|x|), 1^q(|x|)) \)
- Correctness follows immediately from definitions
NP-complete Languages

- **TMSAT** is not very interesting example
  - Definition tied directly to the definition of NP
  - Does not really tell us anything new about NP

- Next objective: *find other NP-complete languages*
  - Many *natural* problems are NP-complete
  - In fact, we have already seen many examples
NP-completeness via Reductions

Theorem

Let $L_1, L_2 \in \{0, 1\}^*$ be languages. If $L_1$ is NP-hard and $L_1 \leq_p L_2$, then $L_2$ is NP-hard.

Proof: Follows from the transitivity of $\leq_p$.

Corollary

Let $L_1, L_2 \in \{0, 1\}^*$ be languages. If $L_1$ is NP-complete, $L_1 \leq_p L_2$, and $L_2 \in \text{NP}$, then $L_2$ is NP-complete.
NP-completeness via Reductions

**Next lectures:**
- Prove that a problem called *CNF-SAT* is NP-complete
- Prove other NP-completeness results by building a *tree of reductions* step-by-step, starting from CNF-SAT

**For example:**
- We will prove that *3-colouring* is NP-complete via an intermediate problem called *3-SAT*
- The reduction presented at previous lecture then implies that *4-colouring* is NP-complete
Lecture 5: Summary

- Polynomial-time verifiers
- The class NP
- Nondeterministic Turing machines
- NP-completeness
- Existence of an NP-complete language