CS 341: Algorithms

Douglas R. Stinson

David R. Cheriton School of Computer Science
University of Waterloo

April 5, 2019
1 Course Information
2 Introduction
3 Divide-and-Conquer Algorithms
4 Greedy Algorithms
5 Dynamic Programming Algorithms
6 Graph Algorithms
7 Intractability and Undecidability
Table of Contents

7 Intractability and Undecidability
  - Decision Problems
  - The Complexity Class P
  - Decision, Optimal Value and Optimization Problems
  - The Complexity Class NP
  - Reductions
  - NP-completeness and NP-complete Problems
  - Undecidability
Decision Problems

Decision Problem: Given a problem instance $I$, answer a certain question “yes” or “no”.

Problem Instance: Input for the specified problem.

Problem Solution: Correct answer ("yes" or "no") for the specified problem instance. $I$ is a yes-instance if the correct answer for the instance $I$ is “yes”. $I$ is a no-instance if the correct answer for the instance $I$ is “no”.

Size of a problem instance: $\text{Size}(I)$ is the number of bits required to specify (or encode) the instance $I$. 
The Complexity Class $P$

**Algorithm Solving a Decision Problem:** An algorithm $A$ is said to solve a decision problem $\Pi$ provided that $A$ finds the correct answer ("yes" or "no") for every instance $I$ of $\Pi$ in finite time.

**Polynomial-time Algorithm:** An algorithm $A$ for a decision problem $\Pi$ is said to be a polynomial-time algorithm provided that the complexity of $A$ is $O(n^k)$, where $k$ is a positive integer and $n = \text{Size}(I)$.

**The Complexity Class $P$** denotes the set of all decision problems that have polynomial-time algorithms solving them. We write $\Pi \in P$ if the decision problem $\Pi$ is in the complexity class $P$. 
Cycles in Graphs

Problem 7.1
Cycle
Instance: An undirected graph $G = (V, E)$.
Question: Does $G$ contain a cycle?

Problem 7.2
Hamiltonian Cycle
Instance: An undirected graph $G = (V, E)$.
Question: Does $G$ contain a hamiltonian cycle?

A hamiltonian cycle is a cycle that passes through every vertex in $V$ exactly once.
Knapsack Problems

**Problem 7.3**

**0-1 Knapsack-Dec**

**Instance:** a list of profits, $P = [p_1, \ldots, p_n]$; a list of weights, $W = [w_1, \ldots, w_n]$; a capacity, $M$; and a target profit, $T$.

**Question:** Is there an $n$-tuple $[x_1, x_2, \ldots, x_n] \in \{0, 1\}^n$ such that $\sum w_i x_i \leq M$ and $\sum p_i x_i \geq T$?

**Problem 7.4**

**Rational Knapsack-Dec**

**Instance:** a list of profits, $P = [p_1, \ldots, p_n]$; a list of weights, $W = [w_1, \ldots, w_n]$; a capacity, $M$; and a target profit, $T$.

**Question:** Is there an $n$-tuple $[x_1, x_2, \ldots, x_n] \in [0, 1]^n$ such that $\sum w_i x_i \leq M$ and $\sum p_i x_i \geq T$?
Polynomial-time Turing Reductions

Suppose $\Pi_1$ and $\Pi_2$ are problems (not necessarily decision problems). A (hypothetical) algorithm $B$ to solve $\Pi_2$ is called an **oracle** for $\Pi_2$.

Suppose that $A$ is an algorithm that solves $\Pi_1$, assuming the existence of an oracle $B$ for $\Pi_2$. ($B$ is used as a subroutine within the algorithm $A$.) Then we say that $A$ is a **Turing reduction** from $\Pi_1$ to $\Pi_2$, denoted $\Pi_1 \leq_T \Pi_2$.

A Turing reduction $A$ is a **polynomial-time Turing reduction** if the running time of $A$ is polynomial, under the assumption that the oracle $B$ has **unit cost** running time.

If there is a polynomial-time Turing reduction from $\Pi_1$ to $\Pi_2$, we write $\Pi_1 \leq_{T_P} \Pi_2$.

Informally: Existence of a polynomial-time Turing reduction means that if we can solve $\Pi_2$ in polynomial time, then we can solve $\Pi_1$ in polynomial time.
Travelling Salesperson Problems

Problem 7.5

TSP-Optimization

Instance: A graph $G$ and edge weights $w : E \to \mathbb{Z}^+$.  
Find: A hamiltonian cycle $H$ in $G$ such that $w(H) = \sum_{e \in H} w(e)$ is minimized.

Problem 7.6

TSP-Optimal Value

Instance: A graph $G$ and edge weights $w : E \to \mathbb{Z}^+$.  
Find: The minimum $T$ such that there exists a hamiltonian cycle $H$ in $G$ with $w(H) = T$.

Problem 7.7

TSP-Decision

Instance: A graph $G$, edge weights $w : E \to \mathbb{Z}^+$, and a target $T$.  
Question: Does there exist a hamiltonian cycle $H$ in $G$ with $w(H) \leq T$?
TSP-Optimal Value $\leq_{\mathcal{T}}^{P} \text{TSP-Dec}$

Algorithm: \textit{TSP-OptimalValue-Solver}(G, w)

\begin{align*}
\text{external} \quad \text{TSP-Dec-Solver} \\
hi &\leftarrow \sum_{e \in E} w(e) \\
lo &\leftarrow 0 \\
\text{if not TSP-Dec-Solver}(G, w, hi) \quad \text{then return } (\infty) \\
\text{while } hi > lo \\
\text{do} & \\
& \begin{cases} \\
& \text{mid }\leftarrow \left\lfloor \frac{hi+lo}{2} \right\rfloor \\
& \text{if TSP-Dec-Solver}(G, w, mid) \\
& \text{then } hi \leftarrow mid \\
& \text{else } lo \leftarrow mid + 1 \\
\end{cases} \\
\text{return } (hi)
\end{align*}

This is a standard binary search technique.
TSP-Optimization $\leq_{T_P}^{T} \text{ TSP-Dec}$

**Algorithm:** \( \text{TSP-Optimization-Solver}(G = (V, E), w) \)

external \( \text{TSP-OptimalValue-Solver}, \text{TSP-Dec-Solver} \)

\( T^* \leftarrow \text{TSP-OptimalValue-Solver}(G, w) \)

if \( T^* = \infty \) then return (“no hamiltonian cycle exists”)

\( w_0 \leftarrow w \)

\( H \leftarrow \emptyset \)

for all \( e \in E \)

\[ \begin{cases} 
  w_0[e] \leftarrow \infty \\
  \text{if not } \text{TSP-Dec-Solver}(G, w_0, T^*) \\
  \text{then } \{ w_0[e] \leftarrow w[e], H \leftarrow H \cup \{e\} \}
\] 

return (\( H \))
Proof of Correctness

Clearly $H$ contains a hamiltonian cycle of minimum weight $T^*$ at the end of the algorithm (note that $H$ just consists of the edges that are not deleted from $G$). We claim that $H$ is precisely a hamiltonian cycle.

Suppose not; then $C \cup \{e\} \subseteq H$, where $C$ is a hamiltonian cycle of weight $T^*$ and $e \in G \setminus C$. Consider the iteration when $e$ was added to $H$. Let $G'$ denote the graph $G$ at this point in time. $G'$ contains a hamiltonian cycle of weight $T^*$ but $G' \setminus \{e\}$ does not, so $e$ is included in $H$. We are assuming that

$$C \cup \{e\} \subseteq H,$$

which implies

$$C \subseteq H \setminus \{e\}.$$

Since $H \subseteq G'$, we have

$$C \subseteq H \setminus \{e\} \subseteq G' \setminus \{e\}.$$

Therefore $e$ would not have been added to $H$, which is a contradiction.
Certificates

Certificate: Informally, a certificate for a yes-instance $I$ is some “extra information” $C$ which makes it easy to verify that $I$ is a yes-instance.

Certificate Verification Algorithm: Suppose that $Ver$ is an algorithm that verifies certificates for yes-instances. Then $Ver(I, C)$ outputs “yes” if $I$ is a yes-instance and $C$ is a valid certificate for $I$. If $Ver(I, C)$ outputs “no”, then either $I$ is a no-instance, or $I$ is a yes-instance and $C$ is an invalid certificate.

Polynomial-time Certificate Verification Algorithm: A certificate verification algorithm $Ver$ is a polynomial-time certificate verification algorithm if the complexity of $Ver$ is $O(n^k)$, where $k$ is a positive integer and $n = \text{Size}(I)$. 
The Complexity Class NP

Certificate Verification Algorithm: A certificate verification algorithm \textit{Ver} is said to solve a decision problem \( \Pi \) provided that

- for every yes-instance \( I \), there exists a certificate \( C \) such that \( \text{Ver}(I, C) \) outputs “yes”.
- for every no-instance \( I \) and for every certificate \( C \), \( \text{Ver}(I, C) \) outputs “no”.

The Complexity Class \textbf{NP} denotes the set of all decision problems that have polynomial-time certificate verification algorithms solving them. We write \( \Pi \in \text{NP} \) if the decision problem \( \Pi \) is in the complexity class \textbf{NP}.

Finding Certificates vs Verifying Certificates: It is not required to be able to find a certificate \( C \) for a yes-instance in polynomial time in order to say that a decision problem \( \Pi \in \text{NP} \).

Important Fact: \( P \subseteq \text{NP} \).
Certificate Verification Algorithm for Hamiltonian Cycle

A certificate consists of an $n$-tuple, $X = [x_1, \ldots, x_n]$, that might be a hamiltonian cycle for a given graph $G = (V, E)$ (where $n = |V|$).

**Algorithm:** Hamiltonian Cycle Certificate Verification($G, X$)

```plaintext
flag ← true
Used ← \{x_1\}
\text{\textbf{j} ← 2}

\textbf{while} (j \leq n) \textbf{and} flag

\text{\textbf{do}}

\text{\textbf{if}} (j = n) \textbf{then} flag ← flag \textbf{and} (\{x_{n-1}, x_n\} \in E)

Used ← Used \cup \{x_j\}

j ← j + 1

\textbf{return} (flag)
```
Polynomial Transformations

For a decision problem \( \Pi \), let \( \mathcal{I}(\Pi) \) denote the set of all instances of \( \Pi \). Let \( \mathcal{I}_{\text{yes}}(\Pi) \) and \( \mathcal{I}_{\text{no}}(\Pi) \) denote the set of all yes-instances and no-instances (respectively) of \( \Pi \).

Suppose that \( \Pi_1 \) and \( \Pi_2 \) are decision problems. We say that there is a \textbf{polynomial transformation} from \( \Pi_1 \) to \( \Pi_2 \) (denoted \( \Pi_1 \leq_P \Pi_2 \)) if there exists a function \( f : \mathcal{I}(\Pi_1) \rightarrow \mathcal{I}(\Pi_2) \) such that the following properties are satisfied:

- \( f(I) \) is computable in polynomial time (as a function of \( \text{size}(I) \), where \( I \in \mathcal{I}(\Pi_1) \))
- if \( I \in \mathcal{I}_{\text{yes}}(\Pi_1) \), then \( f(I) \in \mathcal{I}_{\text{yes}}(\Pi_2) \)
- if \( I \in \mathcal{I}_{\text{no}}(\Pi_1) \), then \( f(I) \in \mathcal{I}_{\text{no}}(\Pi_2) \)
Polynomial Transformations (cont.)

Polynomial transformations are also known as **Karp reductions** or **many-one reductions**.

A polynomial transformation can be thought of as a (simple) special case of a polynomial-time Turing reduction, i.e., if $\Pi_1 \leq_P \Pi_2$, then $\Pi_1 \leq_P \Pi_2$.

Given a polynomial transformation $f$ from $\Pi_1$ to $\Pi_2$, the corresponding Turing reduction is as follows:

1. Given $I \in I(\Pi_1)$, construct $f(I) \in I(\Pi_2)$.
2. Given an oracle for $\Pi_2$, say $A$, run $A(f(I))$.

We transform the instance, and then make a single call to the oracle.

**Very important point:** We do not know whether $I$ is a yes-instance or a no-instance of $\Pi_1$ when we transform it to an instance $f(I)$ of $\Pi_2$.

To prove the implication “if $I \in I_{\text{no}}(\Pi_1)$, then $f(I) \in I_{\text{no}}(\Pi_2)$”, we usually prove the contrapositive statement “if $f(I) \in I_{\text{yes}}(\Pi_2)$, then $I \in I_{\text{yes}}(\Pi_1)$.”
Two Graph Theory Decision Problems

Problem 7.8

Clique

Instance: An undirected graph $G = (V, E)$ and an integer $k$, where $1 \leq k \leq |V|$.

Question: Does $G$ contain a clique of size $\geq k$? (A clique is a subset of vertices $W \subseteq V$ such that $uv \in E$ for all $u, v \in W$, $u \neq v$.)

Problem 7.9

Vertex Cover

Instance: An undirected graph $G = (V, E)$ and an integer $k$, where $1 \leq k \leq |V|$.

Question: Does $G$ contain a vertex cover of size $\leq k$? (A vertex cover is a subset of vertices $W \subseteq V$ such that $\{u, v\} \cap W \neq \emptyset$ for all edges $uv \in E$.)
Clique \leq P Vertex-Cover

Suppose that \( I = (G, k) \) is an instance of **Clique**, where \( G = (V, E) \), \( V = \{v_1, \ldots, v_n\} \) and \( 1 \leq k \leq n \).

Construct an instance \( f(I) = (H, \ell) \) of **Vertex Cover**, where \( H = (V, F) \), \( \ell = n - k \) and

\[
\forall i, j \in V, \quad v_iv_j \in F \iff v_iv_j \notin E.
\]

\( H \) is called the **complement** of \( G \), because every edge of \( G \) is a non-edge of \( H \) and every non-edge of \( G \) is an edge of \( H \).

We have \( \text{Size}(I) = n^2 + \log_2 k \in \Theta(n^2) \). Computing \( H \) takes time \( \Theta(n^2) \) and computing \( \ell \) takes time \( \Theta(\log n) \), so \( f(I) \) can be computed in time \( \Theta(\text{Size}(I)) \), which is polynomial time.
Clique $\leq_P$ Vertex-Cover (cont.)

Suppose $I$ is a yes-instance of \textbf{Clique}. Therefore there exists a set of $k$ vertices $W$ such that $uv \in E$ for all $u, v \in W$. Define $W' = V \setminus W$. Clearly $|W'| = n - k = \ell$. We claim that $W'$ is a vertex cover of $H$. Suppose $uv \in F$ (so $uv \notin E$). If $\{u, v\} \cap W' \neq \emptyset$, we're done, so assume $u, v \notin W'$. Therefore $u, v \in W$. But $uv \notin E$, so $W$ is not a clique. This is a contradiction and hence $f(I)$ is a yes-instance of \textbf{Vertex Cover}.

Suppose $f(I)$ is a yes-instance of \textbf{Vertex Cover}. Therefore there exists a set of $\ell = n - k$ vertices $W'$ that is a vertex cover of $H$. Define $W = V \setminus W'$. Clearly $|W| = k$. We claim that $W$ is a clique in $G$ ...
Properties of Polynomial-time Transformations

Theorem 7.10

If \( \Pi_1 \) and \( \Pi_2 \) are decision problems, \( \Pi_1 \leq_P \Pi_2 \) and \( \Pi_2 \in \mathsf{P} \), then \( \Pi_1 \in \mathsf{P} \).

Proof.

Suppose \( A \) is a poly-time algorithm for \( \Pi_2 \), having complexity \( O(m^\ell) \) on an instance of size \( m \). Suppose \( f \) is a transformation from \( \Pi_1 \) to \( \Pi_2 \) having complexity \( O(n^k) \) on an instance of size \( n \). We solve \( \Pi_1 \) as follows:

1. Given \( I \in \mathcal{I}(\Pi_1) \), construct \( f(I) \in \mathcal{I}(\Pi_2) \).
2. Run \( A(f(I)) \).

It is clear that this yields the correct answer. We need to show that these two steps can be carried out in polynomial time as a function of \( n = \text{Size}(I) \). Step (1) can be executed in time \( O(n^k) \) and it yields an instance \( f(I) \) having size \( m \in O(n^k) \). Step (2) takes time \( O(m^\ell) \). Since \( m \in O(n^k) \), the time for step (2) is \( O(n^{k\ell}) \), as is the total time to execute both steps.
Theorem 7.11

Suppose that $\Pi_1$, $\Pi_2$ and $\Pi_3$ are decision problems. If $\Pi_1 \leq_P \Pi_2$ and $\Pi_2 \leq_P \Pi_3$, then $\Pi_1 \leq_P \Pi_3$.

Proof.

We have a polynomial transformation $f$ from $\Pi_1$ to $\Pi_2$, and another polynomial transformation $g$ from $\Pi_2$ to $\Pi_3$. We define $h = f \circ g$, i.e., $h(I) = g(f(I))$ for all instances $I$ of $\Pi_1$. (Exercise: fill in the details.)
The Complexity Class **NPC**

The complexity class **NPC** denotes the set of all decision problems $\Pi$ that satisfy the following two properties:

- $\Pi \in \text{NP}$
- For all $\Pi' \in \text{NP}$, $\Pi' \leq_P \Pi$.

**NPC** is an abbreviation for **NP-complete**.

Note that the definition does not imply that NP-complete problems exist!
The Complexity Class NPC (cont.)

**Theorem 7.12**

If $\mathbf{P} \cap \mathbf{NPC} \neq \emptyset$, then $\mathbf{P} = \mathbf{NP}$.

**Proof.**

We know that $\mathbf{P} \subseteq \mathbf{NP}$, so it suffices to show that $\mathbf{NP} \subseteq \mathbf{P}$. Suppose $\Pi \in \mathbf{P} \cap \mathbf{NPC}$ and let $\Pi' \in \mathbf{NP}$. We will show that $\Pi' \in \mathbf{P}$.

1. Since $\Pi' \in \mathbf{NP}$ and $\Pi \in \mathbf{NPC}$, it follows that $\Pi' \leq_P \Pi$ (definition of NP-completeness).

2. Since $\Pi' \leq_P \Pi$ and $\Pi \in \mathbf{P}$, it follows that $\Pi' \in \mathbf{P}$ (see Theorem 7.10 on slide # 279).
Satisfiability and the Cook-Levin Theorem

Problem 7.13

CNF-Satisfiability

Instance: A boolean formula $F$ in $n$ boolean variables $x_1, \ldots, x_n$, such that $F$ is the conjunction (logical “and”) of $m$ clauses, where each clause is the disjunction (logical “or”) of literals. (A literal is a boolean variable or its negation.)

Question: Is there a truth assignment such that $F$ evaluates to true?

Theorem 7.14 (Cook-Levin Theorem)

CNF-Satisfiability $\in$ NPC.
Proving Problems NP-complete

Now, given any NP-complete problem, say $\Pi_1$, other problems in $\text{NP}$ can be proven to be NP-complete via polynomial transformations from $\Pi_1$, as stated in the following theorem:

**Theorem 7.15**

Suppose that the following conditions are satisfied:

- $\Pi_1 \in \text{NPC}$,
- $\Pi_1 \leq_P \Pi_2$, and
- $\Pi_2 \in \text{NP}$.

Then $\Pi_2 \in \text{NPC}$. 
More Satisfiability Problems

Problem 7.16

3-CNF-Satisfiability

Instance: A boolean formula $F$ in $n$ boolean variables, such that $F$ is the conjunction of $m$ clauses, where each clause is the disjunction of exactly three literals.

Question: Is there a truth assignment such that $F$ evaluates to true?

Problem 7.17

2-CNF-Satisfiability

Instance: A boolean formula $F$ in $n$ boolean variables, such that $F$ is the conjunction of $m$ clauses, where each clause is the disjunction of exactly two literals.

Question: Is there a truth assignment such that $F$ evaluates to true?

3-CNF-Satisfiability $\in$ NPC, while 2-CNF-Satisfiability $\in$ P.
CNF-Satisfiability $\leq_P$ 3-CNF-Satisfiability

Suppose that $(X, C)$ is an instance of CNF-SAT, where $X = \{x_1, \ldots, x_n\}$ and $C = \{C_1, \ldots, C_m\}$. For each $C_j$, do the following:

**case 1** If $|C_j| = 1$, say $C_j = \{z\}$, construct four clauses

$$\{z, a, b\}, \{z, a, \overline{b}\}, \{z, \overline{a}, b\}, \{z, \overline{a}, \overline{b}\}.$$

**case 2** If $|C_j| = 2$, say $C_j = \{z_1, z_2\}$, construct two clauses

$$\{z_1, z_2, c\}, \{z_1, z_2, \overline{c}\}.$$

**case 3** If $|C_j| = 3$, then leave $C_j$ unchanged.

**case 4** If $|C_j| \geq 4$, say $C_j = \{z_1, z_2, \ldots, z_k\}$, then construct $k - 2$ new clauses

$$\{z_1, z_2, d_1\}, \{\overline{d_1}, z_3, d_2\}, \{\overline{d_2}, z_4, d_3\}, \ldots,$$

$$\{\overline{d_{k-4}}, z_{k-2}, d_{k-3}\}, \{\overline{d_{k-3}}, z_{k-1}, z_k\}.$$
Correctness of the Transformation

Suppose $I$ is a yes-instance of **CNF-SAT**. We show that $f(I)$ is a yes-instance of **3-CNF-SAT**. Fix a truth assignment for $X$ in which every clause contains a true literal. We consider each clause $C_j$ of the instance $I$.

1. If $C_j = \{z\}$, then $z$ must be true. The corresponding four clauses in $f(I)$ each contain $z$, so they are all satisfied.
2. If $C_j = \{z_1, z_2\}$, then at least one of the $z_1$ or $z_2$ is true. The corresponding two clauses in $f(I)$ each contain $z_1, z_2$, so they are both satisfied.
3. If $C_j = \{z_1, z_2, z_3\}$, then $C_j$ occurs unchanged in $f(I)$.
4. Suppose $C_j = \{z_1, z_2, z_3, \ldots, z_k\}$ where $k > 3$ and suppose $z_t \in C_j$ is a true literal. Define $d_i = \text{true}$ for $1 \leq i \leq t - 2$ and define $d_i = \text{false}$ for $t - 1 \leq i \leq k$. It is straightforward to verify that the $k - 2$ corresponding clauses in $f(I)$ each contain a true literal.
Correctness of the Transformation (cont.)

Conversely, suppose \( f(I) \) is a yes-instance of \( 3\text{-CNF-SAT} \). We show that \( I \) is a yes-instance of \( \text{CNF-SAT} \).

1. Four clauses in \( f(I) \) having the form \( \{z, a, b\} \), \( \{z, a, \overline{b}\} \), \( \{z, \overline{a}, \overline{b}\} \) 
   \( \{z, \overline{a}, \overline{b}\} \) are all satisfied if and only if \( z = \text{true} \). Then the corresponding clause \( \{z\} \) in \( I \) is satisfied.

2. Two clauses in \( f(I) \) having the form \( \{z_1, z_2, c\} \), \( \{z_1, z_2, \overline{c}\} \) are both satisfied if and only if at least one of \( z_1, z_2 = \text{true} \). Then the corresponding clause \( \{z_1, z_2\} \) in \( I \) is satisfied.

3. If \( C_j = \{z_1, z_2, z_3\} \) is a clause in \( f(I) \), then \( C_j \) occurs unchanged in \( I \).
Correctness of the Transformation (cont.)

Finally, consider the $k - 2$ clauses in $f(I)$ arising from a clause $C_j = \{z_1, z_2, z_3, \ldots, z_k\}$ in $I$, where $k > 3$. We show that at least one of $z_1, z_2, \ldots, z_k = \text{true}$ if all $k - 2$ of these clauses contain a true literal.

Assume all of $z_1, z_2, \ldots, z_k = \text{false}$. In order for the first clause to contain a true literal, $d_1 = \text{true}$. Then, in order for the second clause to contain a true literal, $d_2 = \text{true}$. This pattern continues, and in order for the second last clause to contain a true literal, $d_{k-3} = \text{true}$. But then the last clause contains no true literal, which is a contradiction.

We have shown that at least one of $z_1, z_2, \ldots, z_k = \text{true}$, which says that the clause $\{z_1, z_2, z_3, \ldots, z_k\}$ contains a true literal, as required.
3-CNF-Satisfiability $\leq_P$ Clique

Let $I$ be the instance of 3-CNF-SAT consisting of $n$ variables, $x_1, \ldots, x_n$, and $m$ clauses, $C_1, \ldots, C_m$. Let $C_i = \{z_{i1}^i, z_{i2}^i, z_{i3}^i\}$, $1 \leq i \leq m$.

Define $f(I) = (G, k)$, where $G = (V, E)$ according to the following rules:

- $V = \{v_{ij}^i : 1 \leq i \leq m, 1 \leq j \leq 3\}$,
- $v_{ij}^i v_{ij'}^{i'} \in E$ if and only if $i \neq i'$ and $z_j^i \neq z_{j'}^{i'}$, and
- $k = m$.

Non-edges of the constructed graph correspond to

1. “inconsistent” truth assignments of literals from two different clauses; or

2. any two literals in the same clause.
Example

\[ I : \begin{cases} 
C_1 = \{x_1, \overline{x_2}, \overline{x_3}\} \\
C_2 = \{\overline{x_1}, x_2, x_3\} \\
C_3 = \{x_1, x_2, x_3\} 
\end{cases} \]

\[ x_1 = \text{true}, \quad x_2 = \text{true}, \quad x_3 = \text{false} \]
Subset Sum

Problem 7.18

Subset Sum

Instance: A list of sizes $S = [s_1, \ldots, s_n]$; and a target sum, $T$. These are all positive integers.

Question: Does there exist a subset $J \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in J} s_i = T$?
**Vertex Cover \( \leq^P \) Subset Sum**

Suppose \( I = (G, k) \), where \( G = (V, E) \), \(|V| = n\), \(|E| = m\) and \( 1 \leq k \leq n \).

Suppose \( V = \{v_1, \ldots, v_n\} \) and \( E = \{e_0, \ldots, e_{m-1}\} \). For \( 1 \leq i \leq n \), \( 0 \leq j \leq m - 1 \), let \( C = (c_{ij}) \), where

\[
c_{ij} = \begin{cases} 
1 & \text{if } e_j \text{ is incident with } v_i \\
0 & \text{otherwise}.
\end{cases}
\]

Define \( n + m \) sizes and a target sum \( W \) as follows:

\[
a_i = 10^m + \sum_{j=0}^{m-1} c_{ij} \cdot 10^j \quad (1 \leq i \leq n)
\]

\[
b_j = 10^j \quad (0 \leq j \leq m - 1)
\]

\[
W = k \cdot 10^m + \sum_{j=0}^{m-1} 2 \cdot 10^j
\]

Then define \( f(I) = (a_1, \ldots, a_n, b_0, \ldots, b_{m-1}, W) \).
Correctness of the Transformation

Suppose $I$ is a yes-instance of Vertex Cover. There is a vertex cover $V' \subseteq V$ such that $|V'| = k$. For $i = 1, 2$, let $E^i$ denote the edges having exactly $i$ vertices in $V'$. Then $E = E^1 \cup E^2$ because $V'$ is a vertex cover. Let

$$A' = \{a_i : v_i \in V'\} \quad \text{and} \quad B' = \{b_j : e_j \in E^1\}.$$  

The sum of the sizes in $A'$ is

$$k \cdot 10^m + \sum_{\{j : e_j \in E^1\}} 10^j + \sum_{\{j : e_j \in E^2\}} 2 \times 10^j.$$  

The sum of the sizes in $B'$ is

$$\sum_{\{j : e_j \in E^1\}} 10^j.$$  

Therefore the sum of all the chosen sizes is

$$k \cdot 10^m + \sum_{\{j : e_j \in E\}} 2 \cdot 10^j = k \cdot 10^m + \sum_{j=1}^{m} 2 \cdot 10^j = W.$$
Correctness of the Transformation (cont.)

Conversely, suppose $f(I)$ is a yes-instance of Subset Sum. We show that $I$ is a yes-instance of Vertex Cover. Let $A' \cup B'$ be the subset of chosen sizes. Define $V' = \{v_i : a_i \in A'\}$. We claim that $V'$ is a vertex cover of size $k$. In order for the coefficient of $10^m$ to be equal to $k$, we must have $|V'| = k$ (there can’t be any carries occurring). The coefficient of any other term $10^j$ ($0 \leq j \leq m - 1$) must be equal to 2. Suppose that $e_j = v_i v_i'$. There are two possible situations that can occur:

1. $a_i$ and $a_i'$ are both in $A'$. Then $V'$ contains both vertices incident with $e_j$.

2. exactly one of $a_i$ or $a_i'$ is in $A'$ and $b_j \in B'$. In this case, $V'$ contains exactly one vertex incident with $e_j$.

In both cases, $e_j$ is incident with at least one vertex in $V'$. 
Subset Sum \( \leq_P \) 0-1 Knapsack

Let \( I \) be an instance of **Subset Sum** consisting of sizes \([s_1, \ldots, s_n]\) and target sum \( T \).

Define

\[
\begin{align*}
p_i &= s_i, \quad 1 \leq i \leq n \\
w_i &= s_i, \quad 1 \leq i \leq n \\
M &= T
\end{align*}
\]

Then define \( f(I) \) to be the instance of **0-1 Knapsack** consisting of profits \([p_1, \ldots, p_n]\), weights \([w_1, \ldots, w_n]\), capacity \( M \) and target profit \( T \).

**Exercise:** Prove the correctness of this transformation.
Hamiltonian Cycle $\leq_p$ TSP-Dec

Let $I$ be an instance of Hamiltonian Cycle consisting of a graph $G = (V, E)$.

For the complete graph $K_n$, where $n = |V|$, define edge weights as follows:

$$w(uv) = \begin{cases} 1 & \text{if } uv \in E \\ 2 & \text{if } uv \notin E. \end{cases}$$

Then define $f(I)$ to be the instance of TSP-Dec consisting of the graph $K_n$, edge weights $w$ and target $T = n$.

Exercise: Prove the correctness of this transformation.
Summary of Polynomial Transformations

CNF-SAT
↓
3-CNF-SAT
↓
Clique
↓
Vertex Cover

Subset Sum
↓
0-1 Knapsack

Hamiltonian Cycle
↓
TSP-Decision

In the above diagram, arrows denote polynomial transformations. The transformation \( \text{Vertex Cover} \leq_P \text{Hamiltonian Cycle} \) is complicated and is described in a supplementary note.
NP-hard Problems

A problem \( \Pi \) is \textbf{NP-hard} if there exists a problem \( \Pi' \in \text{NPC} \) such that \( \Pi' \leq_T \Pi \).

Every NP-complete problem is automatically NP-hard, but there exist NP-hard problems that are not NP-complete.

Typical examples of NP-hard problems are optimization problems corresponding to NP-complete decision problems.

For example, \( \text{TSP-Decision} \leq_T \text{TSP-Optimization} \) and \( \text{TSP-Decision} \in \text{NPC} \), so \( \text{TSP-Optimization} \) is NP-hard.

This is a “trivial” Turing reduction; the reduction in the reverse direction, which was given on slide \# 269, is more complex.
Undecidability and Undecidability

Undecidability

A decision problem $\Pi$ is **undecidable** if there does not exist an algorithm that solves $\Pi$.

If $\Pi$ is undecidable, then for every algorithm $A$, there exists at least one instance $I \in \mathcal{I}(\Pi)$ such that $A(I)$ does not find the correct answer ("yes" or "no") in finite time.

**Problem 7.19**

**Halting**

**Instance:** A computer program $A$ and input $x$ for the program $A$.

**Question:** When program $A$ is executed with input $x$, will it halt in finite time?
Undecidability of the Halting Problem

Suppose that $Halt$ is a program that solves the Halting Problem. Consider the following algorithm $Strange$.

Algorithm: $Strange(A)$

- external $Halt$
- if not $Halt(A, A)$
  - then return $(!)$
- else
  - $i \leftarrow 1$
  - while $i \neq 0$ do $i \leftarrow i + 1$

What happens when we run $Strange(Strange)$?
Undecidability of the Halting Problem (cont.)

The statement "Halt solves the Halting problem" means that

\[
\text{Halt}(A, x) = \begin{cases} 
\text{true} & \text{if } A(x) \text{ halts} \\
\text{false} & \text{if } A(x) \text{ doesn't halt.}
\end{cases}
\]

Note that \( A \) (the "algorithm") and \( I \) (the "input" to \( A \)) are both strings over some finite alphabet.

What happens when we run \( \text{Strange(Strange)} \)?

We have

\[
\text{Strange(Strange)} \text{ halts } \iff \text{Halt(Strange, Strange)} = \text{false} \\
\iff \text{Strange(Strange)} \text{ does not halt.}
\]

The only conclusion we can make is that the program \( Halt \) does not exist!
Another Undecidable Problem

Here is another example of an undecidable problem. The problem Halt-All takes a program $A$ as input and asks if $A$ halts on all inputs $x$.

We describe a Turing reduction $\text{Halting} \leq_T \text{Halt-All}$, which proves that Halt-All is undecidable.

Assume we have a program $\text{HaltAllSolver}$.

For a fixed program $A$ and input $x$, let $B_x()$ be the program that executes $A(x)$ (so $B_x$ has no input).

Here is the reduction:

1. Given $A$ and $x$ (an instance of Halting), construct the program $B_x$.
2. Run $\text{HaltAllSolver}(B_x)$,

We have

$$\text{HaltAllSolver}(B_x) = \text{true} \iff A(x) \text{ halts},$$

so we can solve the halting problem.
The Post Correspondence Problem

The following problem is also undecidable.

Problem 7.20

Post Correspondence

Instance: two finite lists $\alpha_1, \ldots, \alpha_N$ and $\beta_1, \ldots, \beta_N$ of words over some alphabet $A$ of size $\geq 2$.

Question: Does there exist a finite list of indices, say $i_1, \ldots, i_K$, where $i_j \in \{1, \ldots, N\}$ for $1 \leq j \leq K$, such that

$$\alpha_{i_1} \cdots \alpha_{i_K} = \beta_{i_1} \cdots \beta_{i_K},$$

where a “product” of words denotes their concatenation.