CS 341: Algorithms

Douglas R. Stinson

David R. Cheriton School of Computer Science
University of Waterloo

Winter, 2015

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: *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 Π provided that *A* finds the correct answer ("yes" or "no") for every instance I of Π in finite time.

Polynomial-time Algorithm: An algorithm *A* for a decision problem Π 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 Π is in the complexity class *P*.

Cycles in Graphs

Problem

Cycle

Instance: *An undirected graph* $G = (V, E)$.

Question: *Does* G *contain a cycle?*

Problem

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

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

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 Π_1 and Π_2 are problems (not necessarily decision problems). A (hypothetical) algorithm $\mathbf{A_2}$ to solve Π_2 is called an **oracle** for Π_2.

Suppose that $\mathbf{A}$ is an algorithm that solves Π_1, assuming the existence of an oracle $\mathbf{A_2}$ for Π_2. ($\mathbf{A_2}$ is used as a subroutine within the algorithm $\mathbf{A}$.)

Then we say that $\mathbf{A}$ is a **Turing reduction** from Π_1 to Π_2, denoted $\Pi_1 \leq^T \Pi_2$.

A Turing reduction $\mathbf{A}$ is a **polynomial-time Turing reduction** if the running time of $\mathbf{A}$ is polynomial, under the assumption that the oracle $\mathbf{A_2}$ has **unit cost** running time.

If there is a polynomial-time Turing reduction from Π_1 to Π_2, we write $\Pi_1 \leq_P^T \Pi_2$.

Informally: Existence of a polynomial-time Turing reduction means that if we can solve Π_2 in polynomial time, then we can solve Π_1 in polynomial time.

Travelling Salesperson Problems

Problem

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

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

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 $\leq_P^T$ TSP-Dec

Algorithm: *TSP-OptimalValue-Solver*(G, w)
external *TSP-Dec-Solver*
$hi \leftarrow \sum_{e \in E} w(e)$
$lo \leftarrow 0$
if not *TSP-Dec-Solver*(G, w, hi) **then return** (∞)
while $hi > lo$
do $\begin{cases} mid \leftarrow \left\lfloor \frac{hi+lo}{2} \right\rfloor \\ \textbf{if } \textit{TSP-Dec-Solver}(G, w, mid) \\ \quad \textbf{then } hi \leftarrow mid \\ \quad \textbf{else } lo \leftarrow mid + 1 \end{cases}$
return (hi)

TSP-Optimization $\leq_P^T$ TSP-Dec

Algorithm: *TSP-Optimization-Solver*$(G = (V, E), w)$
external *TSP-OptimalValue-Solver*, *TSP-Dec-Solver*
$T^* \leftarrow$ *TSP-OptimalValue-Solver*(G, w)
if $T^* = \infty$ **then return** ("no hamiltonian cycle exists")
$w_0 \leftarrow w$
$H \leftarrow \emptyset$
for all $e \in E$
do $\begin{cases} w_0[e] \leftarrow \infty \\ \textbf{if not } \textit{TSP-Dec-Solver}(G, w_0, T^*) \\ \quad \textbf{then } \begin{cases} w_0[e] \leftarrow w[e] \\ H \leftarrow H \cup \{e\} \end{cases} \end{cases}$
return (H)

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 $n =$ *Size*(I).

The Complexity Class NP

Certificate Verification Algorithm for a Decision Problem: A certificate verification algorithm *Ver* is said to **solve** a decision problem Π provided that

- **for every** yes-instance I, **there exists** a certificate C such that *Ver*(I, C) outputs "yes".
- **for every** no-instance I and **for every** certificate C, *Ver*(I, C) outputs "no".

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

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)

$\mathit{flag} \leftarrow$ **true**
$\mathit{Used} \leftarrow \{x_1\}$
$j \leftarrow 2$
while $(j \leq n)$ **and** flag

$$\textbf{do} \begin{cases} \mathit{flag} \leftarrow (x_j \notin \mathit{Used}) \textbf{ and } (\{x_{j-1}, x_j\} \in E) \\ \textbf{if } (j = n) \textbf{ then } \mathit{flag} \leftarrow \mathit{flag} \textbf{ and } (\{x_n, x_1\} \in E) \\ \mathit{Used} \leftarrow \mathit{Used} \cup \{x_j\} \end{cases}$$

return (flag)

Polynomial-time Reductions

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

Suppose that Π_1 and Π_2 are decision problems. We say that there is a **polynomial-time reduction** (AKA **polynomial transformation**) from Π_1 to Π_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 *size*(I), where $I \in \mathcal{I}(\Pi_1)$)
- if $I \in \mathcal{I}_{\mathbf{yes}}(\Pi_1)$, then $f(I) \in \mathcal{I}_{\mathbf{yes}}(\Pi_2)$
- if $I \in \mathcal{I}_{\mathbf{no}}(\Pi_1)$, then $f(I) \in \mathcal{I}_{\mathbf{no}}(\Pi_2)$

Two Graph Theory Decision Problems

Problem

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

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

$$v_i v_j \in F \Leftrightarrow v_i v_j \notin E.$$

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.

Properties of Polynomial-time Reductions

Suppose that $\Pi_1, \Pi_2, \ldots$ are decision problems.

Theorem

If $\Pi_1 \leq_P \Pi_2$ *and* $\Pi_2 \in$ *P, then* $\Pi_1 \in$ *P.*

Theorem

$\Pi_1 \leq_P \Pi_2$ *and* $\Pi_2 \leq_P \Pi_3$*, then* $\Pi_1 \leq_P \Pi_3$*.*

The Complexity Class NPC

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

- $\Pi \in$ *NP*
- For all $\Pi' \in$ *NP*, $\Pi' \leq_P \Pi$.

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

Theorem

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

Satisfiability and the Cook-Levin Theorem

Problem

CNF-Satisfiability

Instance: *A boolean formula F in n boolean variables $x_1, \dots, x_n$, 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 $\mathbf{true}$?*

Theorem

CNF-Satisfiability $\in$ *NPC*.

Proving Problems NP-complete

Now, given any NP-complete problem, say Π_1, other problems in *NP* can be proven to be NP-complete via polynomial reductions **from** Π_1, as stated in the following theorem:

Theorem

Suppose that the following conditions are satisfied:

- $\Pi_1 \in$ *NPC,*
- $\Pi_1 \leq_P \Pi_2$, *and*
- $\Pi_2 \in$ *NP.*

Then $\Pi_2 \in$ *NPC.*

More Satisfiability Problems

Problem

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

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 $(X, \mathcal{C})$ is an instance of **CNF-SAT**, where $X = \{x_1, \ldots, x_n\}$ and $\mathcal{C} = \{C_1, \ldots, C_m\}$. 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

$$\{z_1, z_2, d_1\}, \{\overline{d_1}, z_3, d_2\}, \{\overline{d_2}, z_4, d_3\}, \ldots,$$
$$\{\overline{d_{k-4}}, z_{k-2}, d_{k-3}\}, \{\overline{d_{k-3}}, z_{k-1}, z_k\}.$$

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 $C_i = \{z_1^i, z_2^i, z_3^i\}$, $1 \leq i \leq m$.

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

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

Subset Sum

Problem

Subset Sum

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

Question: *Does there exist a subset* $J \subseteq \{1, \dots, n\}$ *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

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

Define $n + m$ sizes and a target sum W as follows:

$$\begin{aligned} a_i &= 10^m + \sum_{j=0}^{m-1} c_{ij} 10^j \quad (1 \leq i \leq n) \\ b_j &= 10^j \quad (0 \leq j \leq m-1) \\ W &= k \cdot 10^m + \sum_{j=0}^{m-1} 2 \cdot 10^j \end{aligned}$$

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

Reductions among NP-complete Problems (summary)

$$
\begin{array}{cl}
\textbf{CNF-SAT} & \\
\Downarrow & \text{slide 185} \\
\textbf{3-CNF-SAT} & \\
\Downarrow & \text{slide 186} \\
\textbf{Clique} & \\
\Downarrow & \text{slide 179} \\
\textbf{Vertex Cover} & \\
\Downarrow & \text{slide 188} \\
\textbf{Subset Sum} &
\end{array}
$$

In the above diagram, arrows denote polynomial reductions.

Undecidable Problems

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

If Π 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

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 $\begin{cases} i \leftarrow 1 \\ \textbf{while } i \neq 0 \textbf{ do } i \leftarrow i + 1 \end{cases}$

What happens when we run *Strange*(*Strange*)?