CS 341: ALGORITHMS

Lecture 18: intractability I

Readings: see website

Trevor Brown

https://student.cs.uwaterloo.ca/~cs341
trevor.brown@uwaterloo.ca
THIS TIME

- Intractability (hardness of problems)
  - Decision problems
  - Complexity class P
  - Polynomial-time Turing reductions
- Introductory reductions
  - Three flavours of the traveling salesman problem
INTRACTABILITY

Studying the **hardness** of problems
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 \( P \).
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 $\sum w_ix_i \leq M$ and $\sum p_ix_i \geq T$?

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 $\sum w_ix_i \leq M$ and $\sum p_ix_i \geq T$?
## 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.
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 $\Pi_1 \leq_T \Pi_2$.

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.

Example: all-pairs-shortest-paths reduces to single-source-shortest-path

A reduction typically:
1. transforms the larger problem's input so it can be fed to the oracle.
2. transforms the oracle's output into a solution to the larger problem.

Since we assume this, we only account for work done outside of the oracle.
## Travelling Salesperson Problems

### Problem 7.5

**TSP-Optimization**

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

**Return type** “a path/cycle H”

**Positive edge weights**

### Problem 7.6

**TSP-Optimal Value**

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

**Note TSP-Dec $\leq^P_T$ TSP-Optimal Value**

**Return type** “a positive integer T”

### Problem 7.7

**TSP-Decision**

*Instance:* A graph $G$, edge weights $w : E \rightarrow \mathbb{Z}^+$, and a target $T$.  
*Question:* Does there exist a hamiltonian cycle $H$ in $G$ with $w(H) \leq T$?

**Note TSP-Dec $\leq^P_T$ TSP-Optimization**

**Return type** “yes/no”
We will use polynomial-time Turing reductions to show that different versions of the TSP are polynomially equivalent: if one of them can be solved in polynomial time, then all of them can be solved in polynomial time. (However, it is believed that none of them can be solved in polynomial time.)

- We already know
  - TSP-Dec $\leq_T^p$ TSP-Optimal Value
  - TSP-Dec $\leq_T^p$ TSP-Optimization
- We show
  - TSP-Optimal Value $\leq_T^p$ TSP-Dec
  - TSP-Optimization $\leq_T^p$ TSP-Dec
TSP-Optimal Value $\leq^{T_P} TSP$-Dec

TSP-Optimal Value input: $G, w$

TSP-Dec() also needs a target $T$

What if we try $TSP$-Dec($G, w, 100$)?

It returns true. But we don’t learn optimal value… just that it’s $\leq 100$

Problem 7.6
TSP-Optimal Value
Instance: A graph $G$ and edge weights $w : E \rightarrow \mathbb{Z}^+$.
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 $w : E \rightarrow \mathbb{Z}^+$, and a target $T$.
Question: Does there exist a hamiltonian cycle $H$ in $G$ with $w(H) \leq T$?

How can we learn the exact optimal value by making such calls?
Is this a "poly-time reduction?" I.e., if we assume TSP-Dec-Solver runs in $O(1)$ time, is the runtime a polynomial in the input size?

Largest possible cycle could include every edge

0 is smallest possible weight for any cycle

Maybe there is no Hamiltonian cycle, at all

Use binary search! How to define the starting range $(lo, hi)$ to search?

Questions: (1) What’s the input size? (2) What’s the runtime?
**What’s the size of the input $I$?**

$Size(I) = Size(V) + Size(E) + Size(w)$

$I = (G,w)$

$G = (V,E)$

**Suppose** the graph is represented as an array of adjacency lists (one list for each vertex), with each list containing edges to neighbouring vertices, where an edge is represented by a weight and the name of the target vertex.

We would then have:

$Size(I) = |V| + \sum_{(u,v) \in E} (\log (w_{uv} + 1) + \log (|V| + 1))$

Array with one slot per vertex $v$ (0(1) bits to store a pointer to the head of $v$’s adjacency list)

Name of the target vertex (in 1..|V|)

Weight of the edge (+1 since need at least 1 bit to write “1”)

**But wait…** $V$, $E$ and $w$ could be represented in many different ways. Could the choice of representation affect our complexity result?

Only for very inefficient representations (that are exponentially larger than optimal).

For example if we store weights in unary

Array with one slot per vertex $v$ (0(1) bits to store a pointer to the head of $v$’s adjacency list)

Weight of the edge (+1 since need at least 1 bit to write “1”)

Name of the target vertex (in 1..|V|)

What would it mean to have a runtime $T$ that is polynomial in $Size(I)$?

We say $T$ is polynomial in $Size(I)$ (denoted $T \in \text{poly}(Size(I))$) iff:

$\exists$ constant $c$ s.t. for all $I$, we have $T \in O(Size(I)^c)$

**We rule out such inefficient representations** for the purpose of proving polynomial runtime.

Polynomial differences in size do not matter.

Exercise: if $T \in \text{poly}(Size(I)^{40})$ then $T \in \text{poly}(Size(I))$
## Input Size Cheat Sheet

<table>
<thead>
<tr>
<th>Input $I$</th>
<th>Perfectly fine choices of $\text{Size}(I)$</th>
<th>Examples of <strong>BAD</strong> choices of $\text{Size}(I)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>int $x$</td>
<td>1 or $\lceil \log(x + 1) \rceil$ (can simplify to $\log(x + 1)$ or $\log x$)</td>
<td>int $x$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\times$</td>
</tr>
<tr>
<td>Graph $(V, E)$</td>
<td>$</td>
<td>V</td>
</tr>
<tr>
<td>with weights $W$:</td>
<td>$</td>
<td>V</td>
</tr>
<tr>
<td>$A[1..n]$ of int</td>
<td>$n$ or $\sum_{i} \log(A[i] + 1)$</td>
<td></td>
</tr>
<tr>
<td>$n \times n$ matrix $m$</td>
<td>$n^2$ or $\sum_{i,j} \log(m_{ij} + 1)$</td>
<td></td>
</tr>
</tbody>
</table>

**Exponentially** larger than optimal representation!

To write down $x=1$, need $\log(1+1)=1$ bit. For $x=2$ this is 2 bits. For $x=4$, 3 bits.

Pick any expression that makes your analysis easy

Pseudo-polynomial $\approx$ no exponentiation of non-constant terms

Technically any **pseudo-polynomial combination** of these terms is fine. For example, the following is fine: $(|E|^{100} + |V|^2) \cdot \sum_{e \in E} \log(w(e) + 1)$
Let's assume $O(1)$ time for operations on weights. Later we'll see this isn't needed to show polytime.

**Algorithm: TSP-Optimal Value-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 ($\infty$)
- while $hi > lo$
  - $mid \leftarrow \left\lceil \frac{hi + lo}{2} \right\rceil$
  - if TSP-Dec-Solver($G, w, mid$) then $hi \leftarrow mid$
  - else $lo \leftarrow mid + 1$
- return $(hi)$

**What's our runtime $T(I)$?**

$$T(I) = O(|E| + \log \sum_{e \in E} w(e))$$

We pick the most similar looking input size expression for G:

$$\text{Size}(I) = |E| + \sum_{e \in E} \log(w(e) + 1)$$

**How are $T(I)$ and $\text{Size}(I)$ related?**

- $\log \sum_{e \in E} w(e) \text{ vs } \sum_{e \in E} \log(w(e) + 1)$
- $\log \sum_{e \in E} w(e) \leq \log \prod_{e \in E} (w(e) + 1)$
  $$= \sum_{e \in E} \log(w(e) + 1)$$

So, $T(I) \in O(\text{Size}(I))$

So we have $T(I) \in poly(\text{Size}(I))$ trivially.

**Exercise: show the variant of this reduction where linear search is used instead of binary search is not poly($\text{Size}(I)$)**
So TSP-OptimalValue-Solver is polytime... But is it a correct reduction from TSP-Optimal Value to TSP-Dec?

Need to prove: TSP-OptimalValue-Solver(G,w) returns \( W \), the weight \( W \) of the shortest Hamiltonian Cycle (HC) in \( G \).

Sketch: We return \( \infty \) iff there is no HC. Loop invariant: \( W \in [lo, hi] \). So, at termination when \( hi = lo \), we return exactly \( hi = W \).

In more complex reductions where we transform the input before calling the oracle, we will need a more complex proof:

(A) If there is a(n optimal) solution in the input, our transformation will preserve that solution so the oracle can find it, and

(B) Our transformation doesn’t introduce new solutions that are not present in the original input (i.e., if we find a solution in the transformed input, there was a corresponding solution in the original input).

More on this later...
We have therefore shown: 

\[ TSP-Optimal\text{-}Value \leq_T \text{TSP-Dec} \]

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

1. \( hi \leftarrow \sum_{e \in E} w(e) \)
2. \( lo \leftarrow 0 \)
3. if not \( \text{TSP-Dec-Solver}(G, w, hi) \) then return \((\infty)\)
4. while \( hi > lo \)
   - \( \text{mid} \leftarrow \frac{hi + lo}{2} \)
   - if \( \text{TSP-Dec-Solver}(G, w, \text{mid}) \)
     - then \( hi \leftarrow \text{mid} \)
     - else \( lo \leftarrow \text{mid} + 1 \)
5. return \((hi)\)

But **multiplying polynomials** of degrees \( d_1, d_2 \) 
results in a polynomial of degree \( \leq d_1d_2 \). **Example:**

\[
P_1(x) = 5x^2 + 10x + 100
\]
\[
P_2(x) = 20x^3 + 20
\]
\[
P_1(P_2(x)) = 5P_2(x)^2 + 10P_2(x) + 100
\]
\[
= 5(20x^3 + 20)^2 + 10(20x^3 + 20) + 100
\]

So, \( TSP-\text{OptimalValue}-\text{Solver} \) is **polytime**, and is a **correct** reduction.

We have therefore shown: 

\[ TSP-\text{Optimal Value is polytime reducible to TSP-Dec} \]

So, if an \( O(1) \) implementation of \( \text{TSP-Dec-Solver} \) exists, then we have a **polytime** implementation of \( \text{TSP-Optimal-Value-Solver}! \)

In fact, \( TSP-\text{OptimalValue-Solver} \) remains **polytime** even if the implementation of the **oracle runs in polytime** instead of \( O(1) \)!

**The key idea is:** Consider polynomials \( P_R(s) \) and \( P_O(s) \) representing the runtime of a reduction and its oracle, respectively, on an input of size \( s \).

**Worst possible runtime** happens if every step in the reduction is a call to the oracle.

This is \( P_R(s) \cdot P_O(s) \) --- **multiplication of polynomials**.
BONUS SLIDES

Advanced material on efficient vs inefficient input representations
What's the size of the input $I$?

$$\text{Size}(I) = \text{Size}(V) + \text{Size}(E) + \text{Size}(w)$$

$I = (G, w)$  
$G = (V, E)$

**But wait...** $V$, $E$ and $w$ could be represented in many different ways. Could the choice of representation affect our complexity result?

**Representation 1:** What if the entire graph is simply represented as a **weight matrix** $W$ which contains a weight $w_{uv}$ for each $u, v \in V$ ($\infty$ if an edge does not exist)

Consider weight $w_{uv}$. It takes $\Theta(\log w_{uv})$ bits to store this weight.

We would then have: 
$$\text{Size}(R_1) = \sum_{u \in V} \sum_{v \in V} \log(w_{uv} + 1)$$

What would it mean to have a **runtime** $T$ that is **polynomial in** $\text{Size}(R_1)$?

We say $T$ is polynomial in $\text{Size}(R_1)$ (denoted $T \in \text{poly}(\text{Size}(R_1))$) iff:

$$\exists \text{ constant } c \text{ s.t. for all } I, \text{ we have } T \in O(\text{Size}(R_1)^c)$$
Representation 2: What if the graph were represented as an **array of adjacency lists** (one list for each vertex), with each list containing **edges** to neighbouring vertices, where an edge is represented by a **weight** and the **name** of the target vertex?

We would then have:

\[
\text{Size}(R_2) = |V| + \sum_{(u,v) \in E} (\log(w_{uv} + 1) + \log|V|)
\]

**Array** with one slot per vertex \( v \) (\( O(1) \) bits to store a **pointer** to the head of \( v \)'s adjacency list)

**Weight** of the edge

**Name** of the target vertex

Compare with **representation 1**:

\[
\text{Size}(R_1) = \sum_{u \in V} \sum_{v \in V} \log(w_{uv} + 1)
\]
**Representation 3:** What if we were to represent the graph as a weight matrix $W$ but write all weights in *unary*, instead of binary *(so it takes $w_{uv}$ bits to store weight $w_{uv}$)*.

For this (very stupid) representation, we would then have:

$$\text{Size}(R_3) = \sum_{u \in V} \sum_{v \in V} (w_{uv} + 1)$$

This can be exponentially larger than $\text{Size}(R_1)$!

For example, in a graph where there are $O(1)$ nodes and all edges have weight $w$:

$$\text{Size}(R_1) = \Theta(\log_2 w) \quad \text{and} \quad \text{Size}(R_3) = \Theta(w).$$

In this case, $\text{Size}(R_3) \in \Theta(2^{\text{Size}(R_1)})$

We should **rule out this highly inefficient representation** for the purpose of proving polynomial runtime.

Idea: determine whether runtime is polynomial in the size of the **optimal representation of the input**

Problem: it’s not clear what the **optimal** representation is…

What if we can argue the runtime is polynomial in some **lower bound** on the size of the input?
To prove that a reduction’s runtime $T(I)$ on input $I$ is polynomial in the size of $I$:

- Define a lower bound $L(I)$ on the size of $I$.
- For every possible representation $I_R$ of $I$, $L(I) \leq \text{Size}(I_R)$ should hold.
  - Can be proved with information theory, or ad-hoc; **outside the scope of the course**.
  - In this course, we can be a bit **sloppy**, and just use the **table of valid choices** here to obtain a term for each variable in $I$.
- Then, if we can show $T(I) \leq \text{poly}(L(I))$, we have **actually** shown $T(I) \leq \text{poly}(\text{size}(I))$.

<table>
<thead>
<tr>
<th>Input $I$</th>
<th>$L(I)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>int $x$</td>
<td>1 or $\log(x + 1)$</td>
</tr>
<tr>
<td>Graph $(V, E)$ possibly with weights $W$</td>
<td>1 or $</td>
</tr>
<tr>
<td>$A[1..n]$ of int</td>
<td>$n$ or $\sum_i \log(A[i] + 1)$</td>
</tr>
<tr>
<td>$n \times n$ matrix $m$</td>
<td>$n^2$ or $\sum_{i,j} \log(m_{ij} + 1)$</td>
</tr>
</tbody>
</table>

**Justifying sloppy analysis:**

**Polynomial differences** in choices of $L(I)$, such as $|V|$ vs $|V|^2$ vs $(|E| + |V|)^{40}$ don’t matter.

Such differences cannot change whether a runtime $T(I)$ is in $\text{poly}(L(I))$ or not.

**Exercise:** $T(I) \in \text{poly}(L(I)^{40})$ iff $T(I) \in \text{poly}(L(I))$
So what's a valid \( L(I) \) for an input \( I \) to TSP-OptimalValue-Solver?

**Algorithm:** TSP-OptimalValue-Solver \((G, w)\)

external TSP-Dec-Solver

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

Input is a graph \( G \) with weight matrix \( w \).

From the table of valid \( L(I) \) choices, we let

\[
L(I) = |E| + \sum_{e \in E} \log(w(e) + 1).
\]

What's the relationship between the reduction's runtime \( T(I) \) and \( L(I) \)?

\[
T(I) = O(|E| + \log \sum_{e \in E} w(e))
\]

and \( L(I) = O(|E| + \sum_{e \in E} \log(w(e) + 1)) \)

As we argued earlier, \( T(I) \in \text{poly}(L(I)) \)

and thus \( T(I) \in \text{poly}(\text{Size}(I)) \)

Exercise: show the variant of this reduction where linear search is used instead of binary search is not polynomial.

So this reduction has runtime that is polynomial in the input size!