**Intractability**

Studying the hardness of problems

---

**Relative problem hardness?**

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

---

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

*Polynomial-time Algorithm:* An algorithm \( A \) for a decision problem \( I \) 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 \( I \in P \) if the decision problem \( I \) 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.

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 optimized.

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 $w(H) \leq T$?

Polynomial-time Turing Reductions

Suppose $I_1$ and $I_2$ are problems (not necessarily decision problems). A (hypothetical) algorithm $B$ is an oracle for $I_1$.

Suppose that $A$ is an algorithm that solves $I_2$, assuming the existence of an oracle $B$ for $I_1$ (it is used as a subroutine within the algorithm $A$).

Then we say that $A$ is a Turing reduction from $I_2$ to $I_1$, denoted $I_1 \leq^T I_2$.

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 $I_2$ to $I_1$, we write $I_2 \leq^P I_1$.

Informally: Existence of a polynomial-time Turing reduction means that if we can solve $I_2$ in polynomial time, then we can solve $I_1$ in polynomial time.

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.

- We already know:
  - TSP-Dec $\leq^P$ TSP-Optimal Value
  - TSP-Dec $\leq^P$ TSP-Optimization

- We show:
  - TSP-Optimal Value $\leq^P$ TSP-Dec
  - TSP-Optimization $\leq^P$ TSP-Dec

We will use binary search to define the starting range $(0, 2^T)$ to search!
Exercise: show the variant of this reduction (that are exponential) for our analysis.

Algorithm: TSP-OptimalValue-Solver(G, w)
Let (G, w) be the graph.

// External call
Return TSP-Dec(G).

if not TSP-Dec(G, w, k):
    return (x)

if w_\mid mid > w_\mid mid + 1:
    return (x)

if TSP-Dec(G, w, mid):
    return (x)

mid = \lceil \frac{\log_2 n}{2} \rceil

TSP-OptimalValue = TSP-Dec(G, w, mid)

So, TSP-OptimalValue is a polynomial in the size of the input!

what’s our runtime T(x)

So we will choose a size from the table that is convenient for our analysis.

<table>
<thead>
<tr>
<th>Input Size</th>
<th>T(x) Examples of BAD choices of (x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>\Theta(1)</td>
</tr>
<tr>
<td>10, 100</td>
<td>\Theta(1)</td>
</tr>
<tr>
<td>1000</td>
<td>\Theta(1)</td>
</tr>
<tr>
<td>\infty</td>
<td>\Theta(1)</td>
</tr>
</tbody>
</table>

We pick the most similar looking polynomial combination (e.g., as follows). For any sum of terms above.

So T(x) is polylog(x).

Exercise: show the variant of this reduction (that are exponential) for our analysis.

Algorithm: TSP-OptimalValue-Solver(G, w)
Let (G, w) be the graph.

// External call
Return TSP-Dec(G).

if not TSP-Dec(G, w, k):
    return (x)

if w_\mid mid > w_\mid mid + 1:
    return (x)

if TSP-Dec(G, w, mid):
    return (x)

mid = \lceil \frac{\log_2 n}{2} \rceil

TSP-OptimalValue = TSP-Dec(G, w, mid)

So, TSP-OptimalValue is a polynomial in the size of the input!

what’s our runtime T(x)

So we will choose a size from the table that is convenient for our analysis.

<table>
<thead>
<tr>
<th>Input Size</th>
<th>T(x) Examples of BAD choices of (x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>\Theta(1)</td>
</tr>
<tr>
<td>10, 100</td>
<td>\Theta(1)</td>
</tr>
<tr>
<td>1000</td>
<td>\Theta(1)</td>
</tr>
<tr>
<td>\infty</td>
<td>\Theta(1)</td>
</tr>
</tbody>
</table>

We pick the most similar looking polynomial combination (e.g., as follows). For any sum of terms above.

So T(x) is polylog(x).

Exercise: show the variant of this reduction (that are exponential) for our analysis.

Algorithm: TSP-OptimalValue-Solver(G, w)
Let (G, w) be the graph.

// External call
Return TSP-Dec(G).

if not TSP-Dec(G, w, k):
    return (x)

if w_\mid mid > w_\mid mid + 1:
    return (x)

if TSP-Dec(G, w, mid):
    return (x)

mid = \lceil \frac{\log_2 n}{2} \rceil

TSP-OptimalValue = TSP-Dec(G, w, mid)

So, TSP-OptimalValue is a polynomial in the size of the input!

what’s our runtime T(x)

So we will choose a size from the table that is convenient for our analysis.

<table>
<thead>
<tr>
<th>Input Size</th>
<th>T(x) Examples of BAD choices of (x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>\Theta(1)</td>
</tr>
<tr>
<td>10, 100</td>
<td>\Theta(1)</td>
</tr>
<tr>
<td>1000</td>
<td>\Theta(1)</td>
</tr>
<tr>
<td>\infty</td>
<td>\Theta(1)</td>
</tr>
</tbody>
</table>

We pick the most similar looking polynomial combination (e.g., as follows). For any sum of terms above.

So T(x) is polylog(x).

Exercise: show the variant of this reduction (that are exponential) for our analysis.

Algorithm: TSP-OptimalValue-Solver(G, w)
Let (G, w) be the graph.

// External call
Return TSP-Dec(G).

if not TSP-Dec(G, w, k):
    return (x)

if w_\mid mid > w_\mid mid + 1:
    return (x)

if TSP-Dec(G, w, mid):
    return (x)

mid = \lceil \frac{\log_2 n}{2} \rceil

TSP-OptimalValue = TSP-Dec(G, w, mid)

So, TSP-OptimalValue is a polynomial in the size of the input!

what’s our runtime T(x)

So we will choose a size from the table that is convenient for our analysis.

<table>
<thead>
<tr>
<th>Input Size</th>
<th>T(x) Examples of BAD choices of (x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>\Theta(1)</td>
</tr>
<tr>
<td>10, 100</td>
<td>\Theta(1)</td>
</tr>
<tr>
<td>1000</td>
<td>\Theta(1)</td>
</tr>
<tr>
<td>\infty</td>
<td>\Theta(1)</td>
</tr>
</tbody>
</table>

We pick the most similar looking polynomial combination (e.g., as follows). For any sum of terms above.

So T(x) is polylog(x).

Exercise: show the variant of this reduction (that are exponential) for our analysis.

Algorithm: TSP-OptimalValue-Solver(G, w)
Let (G, w) be the graph.

// External call
Return TSP-Dec(G).

if not TSP-Dec(G, w, k):
    return (x)

if w_\mid mid > w_\mid mid + 1:
    return (x)

if TSP-Dec(G, w, mid):
    return (x)

mid = \lceil \frac{\log_2 n}{2} \rceil

TSP-OptimalValue = TSP-Dec(G, w, mid)

So, TSP-OptimalValue is a polynomial in the size of the input!

what’s our runtime T(x)

So we will choose a size from the table that is convenient for our analysis.

<table>
<thead>
<tr>
<th>Input Size</th>
<th>T(x) Examples of BAD choices of (x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>\Theta(1)</td>
</tr>
<tr>
<td>10, 100</td>
<td>\Theta(1)</td>
</tr>
<tr>
<td>1000</td>
<td>\Theta(1)</td>
</tr>
<tr>
<td>\infty</td>
<td>\Theta(1)</td>
</tr>
</tbody>
</table>

We pick the most similar looking polynomial combination (e.g., as follows). For any sum of terms above.

So T(x) is polylog(x).
LOWER BOUNDING Size(I)

- 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 of I, L(I) ≤ Size(J(I)) should hold.
  - This course can be a bit sloppy, and just use the table of valid choices. For a term for each variable in J.
  - Then, if we can show T(I) ≤ poly(L(I)), we have actually shown T(I) ≤ poly(size(I)).

Exercise: T(I) ≤ poly(L(I)) \(\Rightarrow\) T(I) ≤ poly(size(I)).

**The following are valid choices for I for various input types:**

<table>
<thead>
<tr>
<th>Input</th>
<th>L(I)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\inf) (x)</td>
<td>(x^3 + 3)</td>
</tr>
<tr>
<td>Graph (G), (w) possibly with weights (w)</td>
<td>(</td>
</tr>
</tbody>
</table>

**Justifying sloppy analysis:**
- Polynomial differences in choices of I, such as \(|E|\) vs \(\deg(v)\). Don’t matter:
- Both differences cannot change whether a runtime \(T(I)\) is \(\leq poly(size(I))\) or not.

**TSP-Optimal Value \(\leq TSP\-Dec**

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

<table>
<thead>
<tr>
<th>Input</th>
<th>Solver</th>
</tr>
</thead>
<tbody>
<tr>
<td>(G)</td>
<td>TSP-Dec Solver</td>
</tr>
</tbody>
</table>

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

- For the [very stupid] representation, we would then have:
  \[\text{Store}(K) = \sum_{\text{edge } uv} \log w_{uv} + 1\]

- Compare with representation 1:

- For example, in a graph where there are \(\Omega(1)\) nodes and all edges have weight \(w\):
  \[\text{Store}(K) = \log(\text{poly}(w))\] In this case, \(\text{Size}(K) \leq \log^{\Omega(1)}(w)\).

**Problems:** 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?**

**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 neighboring vertices, where each edge is represented by its weight and the name of the target vertex?

We would then have:

\[\text{Size}(K) = |V| + \sum_{v \in V} \log \left(\sum_{u \in V} w_{uv} + 1\right)\]

**Representation 3:** What if we were to represent the graph as a weight matrix \(W\)?

We say \(T\) is polynomial in \(\text{Store}(K)\) (denoted \(T = \text{poly}(\text{Store}(K))\)) if

- \(T = \text{poly}(\text{Store}(K))\)

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

\[\text{Size}(K) = \sum_{v \in V} \log \left(\sum_{u \in V} w_{uv} + 1\right)\]

**Optimal Value \(\leq TSP\-Optimal**

- So, some algorithms could be polynomial in \(\text{Store}(K)\), but exponential in \(\text{Size}(K)\).

**Optimal Value \(\leq TSP\-Dec**

- Could the choice of representation affect our complexity result?

- Consider the edge addition problem, in which \(1\) is added to each weight.

- In this case, \(\text{Size}(K) = O(\log^{\Omega(1)} w)\).

**Exercise:** Show the variant of this reduction would then have:

- \(\text{Size}(K) = \sum_{v \in V} \log \left(\sum_{u \in V} w_{uv} + 1\right)\)

We say \(T\) is polynomial in \(\text{Store}(K)\) (denoted \(T = \text{poly}(\text{Store}(K))\)) if

- \(T = \text{poly}(\text{Store}(K))\)

**Optimal Value \(\leq TSP\-Optimal**

So, some algorithms could be polynomial in \(\text{Store}(K)\), but exponential in \(\text{Size}(K)\).

**Optimal Value \(\leq TSP\-Dec**

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