Complexity Classes (Ch. 34)

The class P: class of problems that can be solved in time that is polynomial in the size of the input, n.

- if input size is n, then the worst-case running time is \( O(n^c) \) for constant c.
- problems in P are considered “tractable” (if not in P, then not tractable)

Some Examples of P and Non-P Problems

SSSP problem in a directed graph, even with negative edge weights, is in P (i.e., \( O(VE) \) time)

Finding the longest simple path between two nodes is Non-P because exhaustive search is “superpolynomial time”.

An Euler tour of a connected, directed graph is a cycle that traverses each edge of \( G \) exactly once, although it may visit a vertex more than once. We can determine whether a graph has an Euler tour and find the edges of such a tour in \( O(E) \) time.

A Hamiltonian cycle of a directed graph is a simple cycle that contains each vertex in \( V \) (each vertex only once). Determining whether a directed graph has a Hamiltonian cycle is a Non-P problem.

Complexity Classes

The class NP: class of problems with solutions that can be verified in time that is polynomial in the size of the input.

- Imagine we are given a “certificate” of a solution (really a potential solution) of a problem. Then the problem is in NP if we can verify that the certificate is a valid solution in polynomial time.
- Relies on the fact that checking a solution is easier than computing it (e.g., check that a list is sorted, rather than sorting it.)

NP-Completeness

The class NP-Complete (NPC): class of the “hardest” problems in NP.

- this class has property that if any NPC problem can be solved in polynomial time (p-time), then all problems in NP could be solved in p-time.
- actual status of NPC problems is unknown
  - No p-time algorithms have been discovered for any NPC problem
  - No one has been able to prove that no p-time algorithm can exist for any of them
- Informally, a problem is in NPC if it is in NP and is as “hard” as any problem in NP.

P, NP, NPC...how are they related?

Any problem in P is also in NP, since if a problem is in P then we can solve it in polynomial-time without even being given a certificate.

So \( P \subseteq NP \).

By definition, \( NPC \subseteq NP \)

P, NP, NPC...how are they related?

Is \( NP \subseteq P ??? \)

- open problem, but intuition says no
- probably the most famous open problem in CS
- seems plausible that the ability to guess and verify a solution in p-time is more powerful than computing a solution from scratch (deterministic p-time)
- so...we think \( P \neq NP \), but no one has proven it one way or the other (despite enormous effort).
P, NP, NPC...why do we care?

So...why do we care to know whether a problem is NP-Complete?

- if it is, then finding a p-time algorithm to solve it is unlikely.
- better to spend your time looking for:
  - an efficient approximation algorithm to find solution close to optimal
  - heuristics that give correct answer with high probability

• The set of “hardest” problems, the NPC problems, includes many problems of importance in science, engineering, operations research, and business, e.g.:
  - traveling salesman
  - bin packing
  - knapsack problem
  - vertex cover (graph coloring)
  - integer linear programming
  - etc...

Decision Problems

Showing problems are either P or NP is easier if we confine our proofs to work with decision problems (problems with yes/no answers)

Example: Shortest paths
- general problem: What is the length of the shortest x to y path?
- decision problem: Is there an x to y path of length ≤ k?

Decision Problems

Rationale for studying decision problems:
- if the decision problem is hard, the general problem is at least as hard
- for many problems, we only need polynomial extra time to solve the general problem
- decision problems are easier to study and results are easier to prove
- all general problems can be rephrased as decision problems

The general Traveling Salesman Problem:

• instance: a set of cities and the distance between each pair of cities (given as a graph).
• goal: Find a tour of minimum cost.

Traveling Salesman Problem (TSP) Decision Version

• instance: a set of cities, the distance between each pair of connected cities (given as a graph), and a bound B.
• question: is there a “tour” that visits every city exactly once, returns to the start, and has total distance ≤ B?
Traveling Salesman Problem (TSP)

Is TSP ∈ NP?
To determine this, we need to show that we can verify a given solution (list of cities) in p-time (i.e., time O(n), where n is the number of cities).

Given an encoding x of a TSP instance and a certificate y,

- check that each city in x is in y exactly once
- check that the start city in y = end city in y
- check that total distance ≤ B

All can be done in O(n) time, so TSP ∈ NP. Call this algorithm $D_{\text{TSP}}$.

Reductions

Let $L_1$ and $L_2$ be two decision problems.

There is a p-time reduction function, $f$, from $L_1$ to $L_2$ ($L_1 \leq_p L_2$) if:

- $f$ transforms an input for $L_1$ into an input for $L_2$ such that the transformed input is a yes-input for $L_2$ iff the original input is a yes-input for $L_1$
- $f$ is computable in p-time (in the size of the input)

P-time Reduction

Suppose we have a problem B that we know how to solve in p-time and we would like to show that a p-time algorithm exists to solve problem A.

We must demonstrate a procedure that transforms any instance $\alpha$ of A into an instance $\beta$ of B with the following characteristics:

1. The transformation is p-time
2. The answers are the same. That is, the answer for $\alpha$ is “yes” iff the answer for $\beta$ is also “yes”

P-time Reduction

We use reduction in the opposite way to show that a problem is NPC. We can use a p-time reduction to show that no known p-time algorithm exists for a particular problem B.

Given an instance $\alpha$ of A (an NPC problem) find a p-time function to transform the instance $\alpha$ to an instance $\beta$ of B (the unknown problem),

1. Convert the input $\alpha$ for A into input $\beta$ for B
2. Run the decision algorithm for B on the instance $\beta$
3. If B has a p-time algorithm, then using the polynomial transformation algorithm, we could convert an instance of A into an instance of B and solve A in p-time, a contradiction to A being NPC.

P-time Reduction Ex: Hamiltonian Circuit Problem to TSP

The Hamiltonian Circuit Decision Problem (HC):

- **Instance**: An undirected graph $G = (V, E)$
- **Question**: Is there a path in $G$ that includes every node exactly once and returns to the start?

The Traveling Salesperson Decision Problem (TSP):

- **Instance**: A set of cities, distances between each city-pair, bound B
- **Question**: Is there a “tour” that visits every city exactly once, returns to the start, and has total distance ≤ B?
P-time Reduction Ex: HC to TSP

Claim: \( HC \leq_p TSP \)

Proof: To prove this, we need to do 2 things:

1. Define the transformation \( f \) mapping inputs for HC into inputs for TSP, and show the mapping can be computed in \( p \)-time in size of HC input.
   - \( f \) must map the input \( G = (V, E) \) for HC into a list of cities, distances, and a bound \( B \) for input to TSP.
2. Prove the transformation is correct.

1. Definition of transformation \( f \) for HC \( \leq_p \) TSP:
   - Given the HC input graph \( G = (V, E) \) with \( n \) nodes:
     - create a set of \( n \) cities labeled with names of nodes in \( V \).
     - set intercity distances \( d(u, v) = \begin{cases} 1 & \text{if } (u, v) \in E \\ 2 & \text{if } (u, v) \notin E \end{cases} \)
     - set bound \( B = n \) (since HC circuit must be of length \( n \)).
   - Note: \( f \) can be computed in \( O(n^2) \) time. Why?

2. Prove the transformation \( f \) for HC \( \leq_p \) TSP is correct

   We will prove this by showing that \( x \in HC \) iff \( f(x) \in TSP \)

2(a) if \( x \in HC \), then \( f(x) \in TSP \)

   Proof of 2(a):
   - \( x \in HC \) means HC input \( G = (V, E) \) has a hamiltonian circuit. Wlog, suppose it is the ordering \( (v_1, v_2, ..., v_n, v_1) \).
   - \( v_i, v_{i+1}, ..., v_{i+j}, v_{i+j+1} \) is also a tour of the cities in \( f(x) \), the transformed TSP instance.
   - The distance of the tour \( (v_1, v_2, ..., v_n, v_1) \) is \( n \) (\( B \)), since each consecutive pair is connected by an edge and all original edges have wt = 1.
   - Thus, \( f(x) \in TSP \), as required.

2(b) if \( f(x) \in TSP \), then \( x \in HC \)

   Proof of 2(b): if \( f(x) \in TSP \), then \( x \in HC \)
   - \( f(x) \in TSP \) means there exists a tour in TSP input that has a total distance \( \leq n = B \). Wlog, suppose the tour goes through cities \( (v_1, v_2, ..., v_n, v_1) \).
   - Since all intercity distances are either 1 or 2 in \( f(x) \), and there are \( n \) intercity "legs" in the tour, each "leg" in tour must have distance 1.
   - So \( G \) must have an edge between each consecutive pair of cities on the tour, and therefore \( (v_1, v_2, ..., v_n, v_1) \) must be a hamiltonian circuit in \( G \).
   - Thus, \( x \in HC \), as required. ■

Since \( HC \leq_p TSP \), then
- if there exists a \( p \)-time algorithm for TSP, then there exists a \( p \)-time algorithm for HC.
  (i.e., HC is no harder than TSP)
- If there does not exist a \( p \)-time algorithm for HC, then there does not exist a \( p \)-time algorithm for TSP
  (i.e., TSP is at least as hard as HC)

NP-Completeness

Definition: A decision problem \( L \) is NP-Complete (NPC) if:

1. \( L \in \text{NP} \), and
2. for every \( L' \in \text{NP}, L' \leq_p L \) (i.e., every \( L' \) in NP can be transformed to \( L \) -- so \( L \) is at least as hard as every problem in NP).
Theorem: Suppose L is NPC:

- if there exists a p-time algorithm for L, then there exists a p-time algorithm for every \( L' \in \text{NP} \), i.e., \( P = \text{NP} \)
- if there does not exist a p-time algorithm for L, then there does not exist a p-time algorithm for any \( L' \in \text{NP} \), i.e., \( P \neq \text{NP} \)

**Theorem 34.4:** If any NPC problem is p-time solvable, Then \( P = \text{NP} \). Equivalently, if any problem in NP is not p-time solvable, then no NPC problem is p-time solvable.

**Proof:** Suppose that \( L \in P \) and that \( L \in \text{NPC} \). Then, for any \( L' \in \text{NP} \), we know \( L' \leq_p L \) (by defn of class NPC). Thus, we know that \( L' \) is no harder than \( L \), so \( L' \in P \).

The second statement is the contrapositive of the first.

It is a well-known logical equivalence that an implication and its contrapositive are logically equivalent. So both parts of the Theorem are true.