How to construct an NP-Complete Proof
Portfolio 7 - Mathematics 441

Mauricio Soroco
60785250

November 2022

Contents

1 Background

There are many problems that we do know how to solve efficiently. If we consider efficient to mean that a problem can be solved in polynomial time, we do not know if there exist algorithms that can solve these problems in polynomial time, and we cannot prove that there are no algorithms that exist to solve these problems in polynomial time. This class of problems that we consider are equivalent in the sense that if we could find a polynomial time algorithm to solve one of them, then we would be able to solve any of them by being able to convert between instances of these problems in polynomial time.

Hereafter we consider the **decision** versions of these problems. A decision problem is a problem posed as a 'yes' or 'no' question. For example "given k, is the graph G k-colourable[1] ?" is a decision problem that could correspond to the optimization problem "Find the smallest k such that the graph is k-colourable". (note that if we can solve this decision problem, we could solve the optimization problem by selection the smallest k among 'yes' instances of the decision problem.)

2 Definitions

- **NP (Nondeterministic polynomial time)**: a class of decision problems where we cannot find the solution efficiently. But if we are provided an instance of a problem and a certificate (a proof or evidence that the instance will give a 'yes' solution.), then we can verify that the instance is indeed a correct solution in polynomial time.
- **NP-Hard**: (informally) A class of problems that are at least as hard as the hardest problems in NP.
- **NP-Complete**: A class of problems that both in NP and NP-Hard.

3 Reductions

We want to be able to define "hardness", for example to say that X is at least as hard as Y. We do this with reductions. Suppose that we had an algorithm that could solve X in polynomial time. If we could transform any instance of Y into an instance of X in polynomial time, then we would be able to solve an instance of Y in polynomial time (ie the time to solve Y would be a polynomial number of steps to convert X to Y, plus a polynomial number of steps to solve X). Note since these are decision problems we don't care about being able to transform the solution to X back into Y since the solution will be 'yes' or 'no'. If such a reduction existed then using X's algorithm we could solve any problem of Y. Thus X is at

[1]The graph colouring optimization problem seeks to find the smallest number, k, of colours that we can use to paint vertices in the graph such that no vertices connected by an edge share the same colour. The decision problem seeks to answer ("yes" or "no") whether the vertices of the graph, G, can be painted with k colours such that no vertices connected by an edge share the same colour.

least as hard as Y. We introduce the notation:

$$Y \leq_P X \tag{1}$$

to mean that **arbitrary instances of Y can be reduced to instances of X in polynomial time**. Therefore, if $Y \leq_P X$ and X can be solved in polynomial time, then Y can be solved in polynomial time. The contrapositive gives that if $Y \leq_P X$ and Y cannot be solved in polynomial time, then X cannot be solved in polynomial time. Therefore if we know that Y is "hard" then X must be at least as hard otherwise it could be used to solve Y. Through another lens, if we could reduce any instance Y to instances of X, these resulting instances of X from Y (call them $\tilde{X}$) may still only be a subset of all possible instances of X. Therefore finding an algorithm to solve Y via $\tilde{X}$ may not help us solve X (if $\tilde{X}$ were a proper subset of all instances of X), and thus again X is at least as hard as Y. (in the section below, it is for this reason that to show that a problem is "hard", we reduce a know "hard" problem *to* it, not the other way around).

4 NP-Complete Proof

To show that a problem is NP-complete, we need to show two things.

1. The problem is in NP
2. The problem is NP-Hard

4.1 $X \in$ NP

One way to show that a decision problem is in NP is to show that a certificate for a 'yes' instance of the decision problem can be checked in polynomial time. Note that this implies that the certificate has to be a polynomial size (or smaller) with respect to the problem instance size for it to be checked in polynomial time.

4.2 $X \in$ NP-Hard

To show a problem X is NP-Hard we need to show a reduction from a known NP-Hard problem, Y, to X. This can be broken down into three steps.

1. Explain the reduction (also consider trivial instances of the problem).
2. Show that the reduction can be done in polynomial time.
3. Show that the reduction is correct (i.e. The answer to an instance of Y gives a 'yes' answer if and only if the corresponding instance of X gives a 'yes' answer).

5 Example

We begin by defining two problems that we will need for this example

5.1 Traveling Salesman Problem

An instance of the TSP is a set $\{C_1, \cdots, C_n\}$ of cities with non-negative costs c_{ij} to travel from city C_i to C_j or from city C_j to C_i. A tour is a path starting at some city, that then travels trough every other city exactly once and finally returns to the starting city. In other words, only the starting city appears twice in the tour. The optimization problem is to find the tour through all the cities will the lowest cost.
The corresponding decision problem to the TSP is "is there a tour through all the cities with cost at most K?" Therefore, an instance to the decision problem for the TSP involves $\{C_1, \cdots, C_n\}, K, c_{ij} \quad \forall i \neq j$.

5.2 Hamiltonian Cycle Problem

Given a graph $G = (V, E)$ where V is the set of vertices, and E is the set of edges. The decision problem: "is there a simple cycle in the graph that visits all the nodes?"
A simple cycle is a cycle in a graph with no repeated vertices except for the beginning and ending vertex.

5.3 Proof: TSP is NP-Complete

We will prove that the Traveling Salesman problem (TSP) is NP-Complete.

5.3.1 TSP $\in$ NP

We begin by first showing that the TSP in in NP. Consider an instance of the decision problem $\{C_1, \cdots, C_n\}, K, c_{ij} \quad \forall i \neq j$, and a certificate $T = (C_{i_1}, C_{i_2}, \cdots C_{i_n})$ for $i \in [1, ..., n]$. Here the certificate is just a tour of the cities. To check that the instance will produce a 'yes' solution to the decision problem, it is sufficient to sum the cost of the tour T and confirm it is less than or equal to K. Note that the tour length is polynomial with respect to the size of the instance, in particular it is $O(n)$ with respect to the number of cities n. Therefore the overall time to compute the sum and confirm it's less than K is $O(n)$. Therefore the TSP is in NP.

5.3.2 TSP $\in$ NP-Hard

To show that the TSP is NP-hard, we will show a reduction from the Hamiltonian Cycle problem—a known NP-Hard problem—to the TSP.

The reduction is as follows. For every node $v_i \in V$, construct a corresponding city C_i. If there is an edge $(i, j) \in E$, then set the cost c_{ij} of travelling between C_i and C_j to 1. Otherwise, set the cost c_{ij} to 2. Finally set $K = n$ where $n = |V|$, the number of nodes in V.

Note that this reduction can be done in polynomial time since it takes $O(n)$ time to generate a list of cities. The costs between cities can be stored in a matrix of size $n \times n$ with entry (i, j) is the cost of travelling between matrix i and j. We could generate the matrix filled

with 2s and replace the entries that should be 1. This has $O(n^2)$ entries and so the overall time for this reduction is $O(n^2)$.

We now show that the TSP instance is 'yes' if and only if the Hamiltonian cycle problem is 'yes'.

Forward implication:

Suppose that we have a 'yes' instance of the TSP such that there is a tour of cities $T = (C_{i_1}, C_{i_2}, \cdots C_{i_n})$ for $i \in [1, ..., n]$ with cost at most $K = n$. It must be the case that the cost between adjacent cities on the tour is 1, plus 1 to get back to the start. (If any cost were 2, then the overall tour cost would have to be at least n + 1 since there are n costs, of at least 1). From our reduction, the only costs that are 1 are those that have an edge between the corresponding nodes of G. Therefore there is an edge between (i_n, i_1) and $(i_j, i_{j+1}) \quad \forall j \in [1, n)$. By the definition of Hamiltonian cycle, G has one given by (i_n, i_1) and $(i_j, i_{j+1}) \quad \forall j \in [1, n)$, and so G is a 'yes' instance.

Reverse implication:

Suppose that we have a 'yes' instance of the Hamiltonian cycle problem such that G has such a cycle given by (i_n, i_1) and $(i_j, i_{j+1}) \quad \forall j \in [1, n)$. The reduction sets $K = n$ and gives us cities $(C_{i_1}, C_{i_2}, \cdots C_{i_n})$ where the cost to get from one to the next is 1 plus 1 to get to the last city. If we consider this to be the solution to the TSP, this tour has cost n and so this is a yes instance of the TSP.

Since the TSP is in NP and NP-Hard, we conclude that the TSP is NP-Complete.

■

6 Sources

- Stack-Exchange - What is the definition of P, NP, NP-complete and NP-hard?
- Wikipedia - NP (complexity)
- Wikipedia - Decision Problem
- Wikipedia - Graph coloring
- Wikipedia - Hamiltonian path problem
- Wikipedia - Travelling salesman problem
- Algorithm Design by Jon Kleinberg and Éva Tardos