A Tutorial on $\mathbf{NP}$-completeness

Hung Q. Ngô *

January 20, 2003

Abstract

This tutorial is still under construction. It shall be constantly updated throughout the semester.

1 A Brief History of $\mathbf{NP}$-completeness Theory

In P or not in P, that's the question!

Looking into introductory algorithm analysis and design textbooks such as [6], we find a lot of interesting and beautiful problems which can be solved in polynomial time with respect to the input size. For instance, sorting n numbers takes $O(n \lg n)$, computing the minimum spanning tree of a graph $G = (V, E)$ takes $O(|V| \lg |V|)$, computing the shortest path tree of $G = (V, E)$ takes $O(|V| \lg |V| + |E|)$, etc. In fact, none of the polynomial time algorithm one sees in such books takes more than $O(n^5)$.

All of these problems have naturally originated from practical engineering areas such as networking, databases, ... The obvious question is: "can all *natural* problems be solved in polynomial time?" Despite decades of research in algorithms, we still do not know the answer to this question. Many natural problems seem to be difficult, in the sense that despite enormous effort from computer scientists and engineers in many different disciplines, no one knows if there is any polynomial time algorithm to solve any of these 10,000 odd natural problems. Worse yet, no one has a proof that no such algorithm exists either.

Let us introduce a few of those problems here:

- VERTEX COVER (VC[1]): given a graph G, find a smallest size vertex cover.
- 0-1 KNAPSACK: A robber found n items in a store, the ith item is worth v_i dollars and weighs w_i pounds ($v_i, w_i \in \mathbb{Z}$), he can only carry W pounds. Which items should he take?[2]
- TRAVELING SALESMAN PROBLEM (TSP): find the shortest route for a salesman to visit each of the n given cities once, and return to the starting city.

So, what should we do? When your boss asks you to write a program solving a problem which you can't come up with an efficient solution, you could

1. Email ask the prof
2. Give up

*Computer Science and Engineering Department, SUNY at Buffalo, Amherst, NY 14260, USA. E-mail: hungngo@cse.buffalo.edu.

[1] Incidentally, VC also stands for Viet Cong during the Vietnam war.

[2] One can devise a dynamic programming solution to this problem which runs in $O(nW)$-time, but this is not polynomial as we shall see later on.

3. Spend the next 6 months working on the problem
4. Give the boss a brute-force algorithm which takes a century to finish
5. Mathematically show the boss that this problem does not have a poly time solution
    - This approach is highly unlikely, it is **very** hard to show such a result.
    - For the hard problems, the best lower bound people have found is $\Omega(n)$, which is totally useless!
6. Mathematically show that your problem is "equivalent" to some problem which no body knows how to solve

The last approach makes the most sense. However, what exactly do we mean by a "hard" problem, and how do we show that two problems are "equivalently" hard? To answer these questions, we need a *computational model*, which is a formal tool to model computation and algorithms. To see briefly the development of the field, one has to go back to Cantor, Russell, Hilbert, Gödel, Church, Turing, Cook, and Karp. Part of the following discussion is based on Chaitin's "The Unknowable" [2] (see also [1, 3]).

In the later decades of the 19th century, the mathematician Georg Cantor (1845-1918), whose full name is Ferdinand Ludwig Philip Cantor, made one of the boldest moves in the history of mathematics, conquering the concept of infinity and infinite sets. He considered the so-called *ordinal numbers*:

$$0, 1, 2, 3, 4, 5, \dots$$

$$0, 1, 2, 3, 4, 5, \dots, \omega$$
$$0, \dots, \omega, \omega + 1, \omega + 2, \dots$$
$$0, \dots, \omega, \omega + 1, \omega + 2, \dots, 2\omega$$
$$0, \dots, \omega, \omega + 1, \dots, 2\omega, \dots, 3\omega, \dots, \omega^2$$
$$0, \dots, \omega, \dots, 2\omega, \dots, \omega^2, \dots, \omega^3$$
$$0, \dots, \omega, \dots, \omega^2, \dots, \omega^3, \dots, \omega^\omega$$
$$0, \dots, \omega, \dots, \omega^2, \dots, \omega^3, \dots, \omega^\omega, \dots, \omega^{\omega^\omega}$$

Basically, he puts a "number" ω at the infinite end of the set of natural numbers, and kept going on. The sequence of "numbers" above should be self-explanatory. He did not stop there, but we shall stop here as the concept of ordinal numbers are intuitively clear enough.

Let 2^S be the set of all subsets of a set S. Cantor showed that $|S| < |2^S|$ using the so-called *diagonalization argument*. And now, we can talk about the *cardinal numbers*:

$$\aleph_0 = |\mathbb{N}|$$
$$\aleph_1 = |2^{\mathbb{N}}|$$
$$\aleph_2 = |2^{2^{\mathbb{N}}}|.$$

Continue this way, we get

$$\aleph_0, \aleph_1, \dots, \aleph_\omega.$$

Why stop there? Keep going on, we get

$$\aleph_0, \aleph_1, \dots, \aleph_\omega, \dots, \aleph_{\omega^2},$$

and

$$\aleph_0, \aleph_1, \ldots, \aleph_\omega, \ldots, \aleph_{\epsilon_0}.$$

So the ordinal numbers were used to index the cardinal numbers!

Cantor's new ideas gained him numerous enemies, and stirred up the greatest debate in history on the foundation of mathematics. The debate involves two greatest mathematicians of the time. David Hilbert said: "no one shall expel us from the paradise which Cantor has created for us!" On the other hand, Henri Poincaré proclaim: "later generations will regard set theory as a disease from which one has recovered!" And many others: "that's not mathematics, it's theology!" Cantor ended his life in a mental hospital.

Why the debate? It has to do with the paradoxes of set theory.

Liar paradox:

"This statement is false!"

Barber paradox:

"in a village, a barber shaves everyone who does not shave himself"

Russell's paradox:

"consider the set of all sets which are not members of themselves"

Berry's paradox:

"the first natural number which can not be named in less than fifteen words"

What is the solution to the paradoxes? A few approaches were proposed:

- Use symbolic logic to do math (Peano, Russell, Whitehead, ...) In fact, an entire volume of Russell's and Whitehead's 3-volume "Principia Mathematica" [17] was needed to show that ... $1 + 1 = 2$!
- Intuitionism (Brouwer): "the only thing to prove that something exists is to exhibit it or to provide a method for calculating it!" (See [7, 13–15]).
- Formalism (Hilbert): let's eliminate from mathematics the uncertainties and ambiguities of natural language. (See [13]).

  It should be possible to devise a *proof-checking algorithm* which, given a set of axioms and inference rules, shall be able to decide if a proof is correct!

  Plus a few other things, this idea is referred to as the *Hilbert's program*.

David Hilbert posed 23 problems to challenge mathematicians of the 20th century, 10 of which were presented at the second international congress of mathematics in Paris, on Aug 8, 1900.

- *Hilbert's 2nd problem:* Can it be proven that the axioms of logic are consistent?
- *Hilbert's 10th problem:* Does there exist an algorithm to solve Diophantine equations?
- *Hilbert's 8th problem:* Riemann hypothesis.

He also asked if mathematics is decidable, i.e. is there an algorithm which decides if a statement is provable? This is often referred to as **Entscheidungsproblem**.

This is when Kurt Gödel comes into the picture. On Hilbert's second problem, Gödel showed (1931) that "mathematics" is either inconsistent or incomplete!

The next step is done by the great Alan Turing (1936). The **Turing machine** was invented to model the term "algorithm" in Entscheidungsproblem, and in Hilbert's 10th problem. Turing also gave an affirmative answer to Entscheidungsproblem with the so-called *halting problem*, which is undecidable by Turing machines. Of course, this is an answer to Entscheidungsproblem if we believe that the intuitive notion of computations and algorithms is captured by the Turing machine model. This statement is the content of the *Church-Turing thesis*.

Now that we have a mathematical model for algorithms and computations, the next step is to decide what a hard problem is.

In the 1960's, researchers and engineers started to notice that some algorithms require significantly longer time to run. Given the scarce computing resources, it seems natural to classify problems based on the running time of the best algorithms solving the problems. Problems which require longer time to solve are considered to be harder. The notion of *polynomial-time computation* was introduced independently by von Neumann [16], Cobham [4], and Edmonds [8]. Edmonds actually called poly-time algorithms the "good" algorithms. Informally, we define $\mathbf{P}$ to be the class of all problems which have a poly-time algorithm.

The problems in $\mathbf{P}$ are considered to be easier than the problems not in $\mathbf{P}$. The class $\mathbf{P}$ can only be formalized if we have a fixed formal model for the intuitive notion of algorithms. The choice has been the Turing machine. Turing machine also has a non-deterministic version (NTM), which is supposedly stronger than the deterministic one (DTM). The non-deterministic Turing machine gives rise to the corresponding class $\mathbf{NP}$ of problems it can solve. The question if NTMs are actually stronger than DTMs is the central question in theoretical computer science, with enormous implication. This is the same as asking: $\mathbf{P} = \mathbf{NP}$? In the same spirit as Hilbert's problems, Clay Research Institute has offered one million dollars to whoever solves one of a few outstanding problems of the 20th century mathematics. One of the problems in the $\mathbf{P} = \mathbf{NP}$? question, another is, still, Hilbert's 8th problem – Riemann hypothesis.

Let's get back to our history before speculating more on the $\mathbf{P} = \mathbf{NP}$? question.

In 1971, Stephen Cook [5] showed that there exists problems in $\mathbf{NP}$ to which all other problems in $\mathbf{NP}$ can be reduced to in poly-time. The reduction is taken informally in the following sense: if A is reducible to B in poly-time, then whenever one can solve B in poly-time, one can also solve A in poly-time. In the poly-time sense, A is obviously at most as hard as B. Thus, the problems Cook found are the hardest ones in the class $\mathbf{NP}$. These problems are called the $\mathbf{NP}$-complete problems. Cook showed that SAT, 3-SAT, and SUBGRAPH ISOMORPHISM are $\mathbf{NP}$-complete. In 1972, Richard Karp [10] found and showed that some 20 other very natural problems are also $\mathbf{NP}$-complete, such as VC, TSP, CLIQUE, INDEPENDENT-SET, ... In 1974, Knuth [] contributed to settling the terminologies on $\mathbf{NP}$-completeness theory as we see them today. There is one connection to Turing: all these people got Turing awards.

TBD: some speculation on the question $\mathbf{P} = \mathbf{NP}$?

What do we do next? It is possible that the Turing machines are not as powerful as needed to solve hard problems of nature.

The first direction to go is to come up with some stronger computational model: quantum computers, DNA computers, ... In fact, INTEGER FACTORIZATION, which is a problem not known to be in $\mathbf{P}$ or not with respect to the Turing machines, has already been shown to be solvable in poly-time with a quantum computer [11, 12].

When other computational models are still pretty much science-fiction, we need something else to get around those hard problems. Approximation algorithms, randomized algorithms, or just heuristics with no performance guarantees, ... are things we could do at the moment.

2 Turing Machines

3 Polynomial Time Reducibility

The classic book on $\mathbf{NP}$-completeness theory is Garey and Johnson's "Computers and intractability" [9]. We shall conform to the convention that problem names are capitalized; e.g., TSP, VC, CLIQUE, ...

3.1 Decision Problems

For technical reasons, we only consider *decision problems*, i.e. problems which ask yes/no questions. All optimization problems do have a decision version. For example, the decision version of TSP is: "given n cities and a number L, is there a TSP tour of length at most L?" Or, the decision version of VC asks if a graph G has a vertex cover of size at most k.

It should be evident that an optimization problem is more difficult than its decision version. For example, if we can solve TSP in poly-time, then we only have to check if the optimal solution has length at most L to answer the decision question. Hence, if the decision version is already hard, then there is not much hope for solving the optimization problem.

3.2 Encodings of Instances of a Problem

Technically, a problem Π is a set of *instances*. An algorithm for Π runs on instances in Π More precisely, an algorithm runs on *encoded* instances of Π. For example, in the VERTEX-COVER problem a particular graph is an instance, the graph's adjacency matrix is an encoding for the instance. An algorithm for finding a minimum VC has adjacency matrices as its inputs

The encoding scheme is important in the sense that it decides the *size* of the inputs. With the adjacency matrix encoding, the input size is actually about $\Theta(n^2)$, not n as commonly thought. However, any polynomial in n^2 is also a polynomial in n and vice versa. Consequently, we do not distinguish input sizes if they only differ polynomially. For instance, if the input size is $\Theta(n^4)$ and the algorithm runs in $O(n^{100})$, then clearly the running time is still a polynomial in n^4. The running time of an algorithm clearly depends on the input size, hence on the encoding scheme used to encode problem's instances.

The choice of a particular encoding scheme could make a huge difference. For example, PRIMALITY TESTING asks just to check if a given number is a prime. Suppose we run the straightforward algorithm of checking for each $k < n$ if n is divisible by k, then the running time is about $O(n \lg n)$. This is **not** a poly-time algorithm if we use the reasonable encoding scheme of representing the input n by its binary format. The input size is actually the number of bits to represent n, which is $\Theta(\lg n)$. Letting $m = \lg n$, then the running time is actually exponential in m: $O(2^m m)$.

From now on, we shall assume *reasonable* encoding schemes. The reader is referred to [9] for a more detailed discussion on what constitutes a reasonable encoding. In particular, all numbers are assumed to be represented in binary format. With this in mind, the $O(nW)$-time algorithm solving the 01-KNAPSACK problem is an exponential algorithm. Moreover, it is intuitively clear that all reasonable encoding schemes yield encodings of sizes within a polynomial of each other.

We are still being somewhat informal in this section when discussing algorithms and their running time. The only place where we shall be technically precise is when we show Cook's theorem that the problem SAT (defined later) is $\mathbf{NP}$-complete. The reason for this informality is that rigorous formulation and proofs are really wordy, losing touch of what is going on behind the scene.

As we have mentioned earlier, we think of each (decision) problem Π as a set of instances. Since the answer to each instance of Π is either yes or no, we can write

$$\Pi = \Pi_{YES} \cup \Pi_{NO},$$

where Π_{YES} consists of all yes-instances of Π and Π_{NO} consists of all no-instances.

3.3 P and NP

We now have enough artilleries to define $\mathbf{P}$ and $\mathbf{NP}$.

The class $\mathbf{P}$ consists of all problems Π for which there exists a poly-time algorithm A such that for any instance $x \in \Pi$, we have

$$x \in \Pi_{YES} \iff A(x) = YES.$$

Here, $A(x)$ is understood as the answer we get after running A on input x.

The class $\mathbf{NP}$ is a little more difficult to define informally. Basically, a problem is in $\mathbf{NP}$ if there exists a poly-time algorithm which can check if a particular candidate solution to the problem yields a YES answer or not. This is quite confusing until we examine a few examples. Let us first define $\mathbf{NP}$ a little more formally. A problem Π is in NP (read "solvable in nondeterministic polynomial time") if there is a poly-time *verification* algorithm $V(\cdot, \cdot)$, such that for any instance $x \in \Pi$,

$$x \in \Pi_{YES} \iff \exists \text{certificate } y, |y| = poly(|x|), V(x, y) = YES.$$

The notation $|y| = poly(|x|)$ means that the certificate y has size polynomial in $|x|$.

For example, in the VC problem, if we are given a subset y of vertices of $G = (V, E)$, we can check if Y is a vertex cover of size at most K in poly-time. Here, the checking procedure is the verification procedure in the definition, the graph G is the instance x, and y is the certificate. For these reasons, VC is in $\mathbf{NP}$.

Similarly, given an instance of TSP and a tour y, we can always check in poly-time if y is actually a tour and if the length of y is at most the given bound L of the problem. Hence, TSP is in $\mathbf{NP}$.

3.4 Polynomial Time Reduction

A problem Π is *polynomial time reducible* to a problem Π' if there is a *polynomial time computable* function $f : \Pi \to \Pi'$ such that for any $x \in \Pi$,

$$x \in \Pi_{YES} \iff f(x) \in \Pi'_{YES}.$$

We write $\Pi \leq_p \Pi'$, and think Π is not harder than Π'. Since the function f maps instances of Π to instances of Π', the concept of polynomial time computable f should be understood as follows. Given an instance x of Π, we can construct the instance $f(x)$ of Π' in polynomial time. TBD: a figure here helps!

To make the concepts clearer, let us consider a few problems. We redefine VC here in a more complete form.

VERTEX COVER (VC)

INSTANCE: A graph $G = (V, E)$, and a bound $b \in \mathbb{Z}^+$, $1 \leq b \leq |V|$.

QUESTION: Is there a vertex cover of size at most b?

(A vertex cover of a graph G is a subset S of vertices of G such that each edge of G is incident to at least one vertex in S.)

CLIQUE

INSTANCE: A graph $G = (V, E)$, and a bound $b \in \mathbb{Z}^+$, $1 \leq b \leq |V|$.

QUESTION: Does G contain a clique of size at least b?

(A clique is a complete graph. If a clique has m vertices, we say that the clique is of order m and use K_m to denote it.)

INDEPENDENT SET (IS)

INSTANCE: A graph $G = (V, E)$, and a bound $b \in \mathbb{N}$.

QUESTION: Is there an independent set of G of size at least b?

(An independent set of a graph G is a subset of vertices no two of which are adjacent.)

Example 3.1. VC $\leq_p$ IS.

Proof. To show this, for each instance $x = (G, b) \in$ VC, we would like to construct in poly-time an instance $x' = (G', b') \in$ CLIQUE, such that G has a vertex cover of size at most b if and only if G' has a clique of size at least b'.

Notice that if $S \subseteq V(G)$ is a vertex cover, then clearly $V(G) - S$ is an independent set and vice versa. Hence, letting $G' = G$ and $b' = |V(G)| - b$ completes the reduction. □

Example 3.2. IS $\leq_p$ CLIQUE.

Proof. Given a graph $G = (V, E)$, the complement graph $\bar{G}$ of G is a graph with the same vertex set as G and the edge set $E(G')$ the complement of $E(G)$; namely $(u, v) \in E'$ iff $(u, v) \notin E$. With this construction, an independent set of G is a clique of $\bar{G}$ and vice versa. The rest of the reduction should now be evident. □

The following lemmas is trivial to show, yet are important properties of polynomial time reduction.

Lemma 3.3. *If* $\Pi' \in \mathbf{P}$*, and* $\Pi \leq_p \Pi'$*, then* $\Pi \in \mathbf{P}$.

Lemma 3.4. *If* $\Pi_1 \leq_p \Pi_2$ *, and* $\Pi_2 \leq_p \Pi_3$*, then* $\Pi_1 \leq_p \Pi_3$.

Finally, we are ready to formally define the $\mathbf{NPC}$ class. A problem Π is $\mathbf{NP}$*-complete* (or $\in \mathbf{NPC}$) if and only if $\Pi \in \mathbf{NP}$ and every problem in $\mathbf{NP}$ is reducible to Π in poly-time.

Lemma 3.5. *Suppose* $\Pi \in \mathbf{NP}$*, and* $\Pi' \leq_p \Pi$ *where* Π' *is* $\mathbf{NP}$*-complete, then* Π *is* $\mathbf{NP}$*-complete.*

Proof. This follows trivially from Lemma 3.4, which says that the relation $\leq_p$ is transitive. □

4 Cook Theorem

Now that the intuitive notion of an algorithm has been formalized by the concept of a deterministic Turing machine, we are ready to prove Cook's theorem that the set of $\mathbf{NP}$-complete (NPC) problems is not empty. The first problem which has been shown to be in NPC is the SATISFIABILITY (SAT) problem.

Let $X = \{x_1, \ldots, x_n\}$ be a set of Boolean variables. A *truth assignment* for X is a function $t : X \to \{T, F\}$. Basically, t assigns to each variable in X a value of either T (true) or F (false). For a variable x, we use $\bar{x}$ to denote the negation of x. If x is a variable, x and $\bar{x}$ are called *literals*.

A set C of literals over X is called a *clause* over X, e.g. $C = \{x_1, \bar{x}_3, x_4\}$ is a clause. A clause C is *satisfied* be a truth assignment t iff at least one of its member is true (T) under t. One thinks of a clause, say $C = \{x_1, \bar{x}_3, x_4\}$, as the OR of the literal members of the clause. In effect, one can think of C as a boolean formula:

$$C = x_1 + \bar{x}_3 + x_4.$$

SATISFIABILITY (SAT)

INSTANCE: A set X of variables and a collection $\mathcal{C} = \{C_1, \ldots, C_m\}$ of clauses over X.

QUESTION: Is there a truth assignment which satisfies all clauses in $\mathcal{C}$.

Intuitively, we want a truth assignment for which $f = T$, given f under conjunctive normal form (CNF). For instance,

$$f(x_1, \ldots, x_n) = (x_1 + \bar{x}_3 + x_4)(x_2 + x_3)(\bar{x}_1 + x_2 + \bar{x}_3)$$

Theorem 4.1 (Cook - 1971). *SAT is* $\mathbf{NP}$*-complete.*

5 Proving NP-completeness Results

Given SAT as a basis, Lemma 3.5 helps showing many other problems are $\mathbf{NP}$-complete. To show that a problem Π is $\mathbf{NP}$-complete, we first have to show that $\Pi \in \mathbf{NP}$, which is in most cases trivial; then, we find another problem Π' which has been shown to be $\mathbf{NP}$-complete, and show that $\Pi' \leq_p \Pi$. Note the direction of reduction here. Many students have done the reduction in the wrong direction, due to the somewhat confusing nature of the word "reducible" in this context.

Another point worth noticing is that finding a reduction often helps us understand the structure of the problem at hand much better. In effect, we show that our problem is a special case of another known problem. If there was a good approximation algorithm to solve the other problem, then our problem can benefit from it via the reduction.

Beside the problems defined earlier like VC, CLIQUE, IS, ..., we define a few more problems here and show that they are also $\mathbf{NP}$-complete. The proofs shall also illustrate several $\mathbf{NP}$-completeness proof techniques. More examples and a classification of techniques can be found in [9].

3-SAT

INSTANCE: A collection $\mathcal{C}$ of clauses $\{C_1, \ldots, C_m\}$ over $X = \{x_1, \ldots, x_n\}$, where each clause C_i consists of exactly 3 literals.

QUESTION: Is there a truth assignment satisfying each clause in $\mathcal{C}$.

SET COVER (SC)

INSTANCE: A family $\mathcal{S}$ of subsets $\{S_1, \ldots, S_m\}$ of a finite universe U ($|U| = n$), and a bound $b \in \mathbb{Z}^+$.

QUESTION: Is there $I \subseteq \{1, \ldots, m\}$, $|I| \leq b$, such that

$$U = \bigcup_{i \in I} S_i.$$

The collection $\{S_i \mid i \in I\}$ satisfying the above condition is called a set cover of U.

DOMINATING SET (DS)

INSTANCE: A graph $G = (V, E)$, a bound $b \in \mathbb{Z}^+$.

QUESTION: Is there $S \subseteq V$, $|S| \leq b$, such that every vertex v not in S is incident to some vertex in S. The vertices in S *dominates* all vertices in V.

3-COLORABILITY

INSTANCE: A graph $G = (V, E)$.

QUESTION: Is G 3-colorable, i.e. is there a way to assign each vertex of G one of 3 colors such that two adjacent vertices have different colors.

k-COLORABILITY

INSTANCE: A graph $G = (V, E)$, $k \geq 3$.

QUESTION: Is G k-colorable, i.e. is there a way to assign each vertex of G one of k colors $\{1, \ldots, k\}$ such that two adjacent vertices have different colors.

HAMILTONIAN CYCLE (HC)

INSTANCE: A graph $G = (V, E)$.

QUESTION: Does G contain a Hamiltonian Cycle? (An HC is a cycle containing all vertices of G.)

TRAVELING SALESMAN (TSP)

INSTANCE: A finite set C of n cities $\{1, \ldots, n\}$ on a plane, their distances $d(i, j) \in \mathbb{Z}^+$, and a bound $b \in \mathbb{Z}^+$.

QUESTION: Is there a TSP tour with total length at most b.

SUBSET-SUM (SS)

INSTANCE: A finite set S of natural numbers, and a target $t \in \mathbb{N}$.

QUESTION: Is there a subset $S' \subseteq S$, whose elements sum up to t.

KNAPSACK

INSTANCE: n items, their values $v_i \in \mathbb{Z}^+$, their corresponding weights $w_i \in \mathbb{Z}^+$, a weight limit $W \in \mathbb{Z}^+$, and a value limit $V \in \mathbb{Z}^+$.

QUESTION: Is there a subset of items with total weight at most W, and total value at least V?

Figure 1 is our road map of reductions to show all aforementioned problems are $\mathbf{NP}$-complete. The arrows point to the direction of reduction. We shall present many of the reductions shown, and left a few easier ones as exercises in the next section.

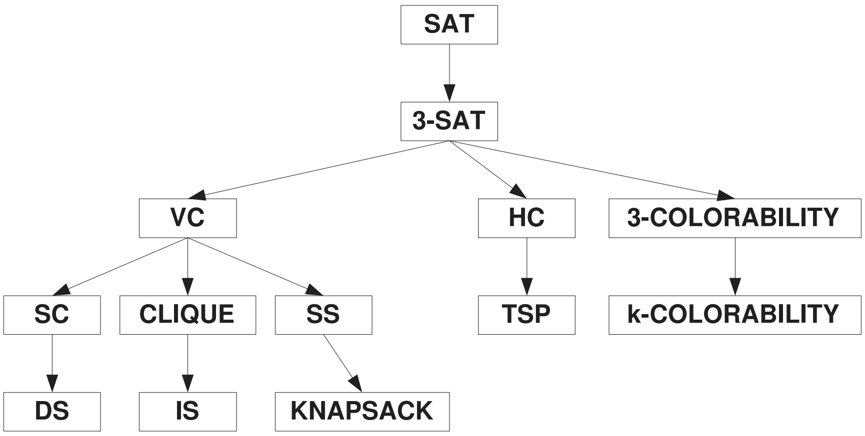

Figure 1: Map of reductions to show $\mathbf{NP}$-complete problems.

Theorem 5.1. *3-SAT is* $\mathbf{NP}$*-complete.*

Proof. 3-SAT is in $\mathbf{NP}$ because given a truth assignment $t : X \to \{T, F\}$, we can easily check if the given boolean function f gets value TRUE under t. All one has to do is to check, for each clause C_i, if one of the 3 literals in C_i was assigned T.

To complete the proof, we show that SAT $\leq_p$ 3-SAT.

Given an instance $\mathcal{C} = \{C_1, \ldots, C_m\}$ of SAT, we would like to construct an instance $\mathcal{C}' = \{C'_1, \ldots, C'_{m'}$ of 3-SAT in polynomial time such that $\mathcal{C}$ is satisfiable if and only if $\mathcal{C}'$ is satisfiable.

Suppose $C_i = \{y_{i1}, \ldots, y_{ij_i}\}$, namely there are j_i literals in C_i indexed from $i1$ to ij_i. For each clause C_i of $\mathcal{C}$ we construct a few clauses of $\mathcal{C}'$, each with exactly 3 literals. The union of the clauses constructed from the C_i would form $\mathcal{C}'$.

We distinguish several cases.

The first case is when C_i has only one literal y_{i1}. The corresponding clauses of $\mathcal{C}'$ are

$$\{\mathbf{y_{i1}}, z_{i1}, z_{i2}\}, \{\mathbf{y_{i1}}, z_{i1}, \bar{z}_{i2}\}, \{\mathbf{y_{i1}}, \bar{z}_{i1}, z_{i2}\}, \{\mathbf{y_{i1}}, \bar{z}_{i1}, \bar{z}_{i2}\}.$$

The variables z_{ij} are auxiliary variables, which is not in the set X of variables of $\mathcal{C}$.

The second case is when $C_i = \{y_{i1}, y_{i2}\}$. The corresponding clauses are

$$\{\mathbf{y_{i1}}, \mathbf{y_{i2}}, z_i\}, \{\mathbf{y_{i1}}, \mathbf{y_{i2}}, \bar{z}_i\}.$$

The third case is when C_i has exactly 3 literals, in which case we include C_i in $\mathcal{C}'$.

The last case is when $C_i = \{y_{i1}, y_{i2}, \ldots, y_{ik}\}$, $k \geq 4$. The corresponding clauses are

$$\{\mathbf{y_{i1}}, \mathbf{y_{i2}}, z_{i1}\}, \{\bar{z}_{i1}, \mathbf{y_{i3}}, z_{i2}\}, \{\bar{z}_{i2}, \mathbf{y_{i4}}, z_{i3}\}, \ldots, \{\bar{z}_{i(k-3)}, \mathbf{y_{i(k-1)}}, \mathbf{y_{ik}}\}.$$

For example, if

$$\mathcal{C} = \{\{\bar{x}_3\}, \{x_1, \bar{x}_4\}, \{x_1, x_3, x_4\}, \{x_1, \bar{x}_2, \bar{x}_3, x_4, x_5, \bar{x}_6\}\},$$

then

$$\begin{aligned} \mathcal{C}' &= \big\{\{\bar{\mathbf{x}}_\mathbf{3}, a, b\}, \{\bar{\mathbf{x}}_\mathbf{3}, \bar{a}, b\}, \{\bar{\mathbf{x}}_\mathbf{3}, a, \bar{b}\}, \{\bar{\mathbf{x}}_\mathbf{3}, \bar{a}, \bar{b}\}\big\} \\ &\bigcup \ \{\{\mathbf{x_1}, \bar{\mathbf{x}}_\mathbf{4}, c\}\{\mathbf{x_1}, \bar{\mathbf{x}}_\mathbf{4}, \bar{c}\}\} \\ &\bigcup \ \{\{\mathbf{x_1}, \mathbf{x_3}, \mathbf{x_4}\}\} \\ &\bigcup \ \big\{\{\mathbf{x_1}, \bar{\mathbf{x}}_\mathbf{2}, +d_1\}\{\bar{d}_1, \bar{\mathbf{x}}_\mathbf{3}, d_2\}\{\bar{d}_2, \mathbf{x_4}, d_3\}\{\bar{d}_3, \mathbf{x_5}, \bar{\mathbf{x}}_\mathbf{6}\}\big\}. \end{aligned}$$

To this end, we show that this construction yields a polynomial time reduction. The fact that this construction could be done in poly-time is obvious. We are left to show that $\mathcal{C}$ is satisfiable if and only if $\mathcal{C}'$ is satisfiable. To show this, it is sufficient to show that a particular clause C_i is satisfiable if and only if its corresponding size-3 clauses in $\mathcal{C}'$ are satisfiable.

For the forward direction, let assume that C_i is satisfiable with some truth assignment t. If $C_i = \{y_{i1}\}$, then we could assign any value to the z_{ij}, while making the four corresponding clauses in $\mathcal{C}'$ TRUE. When $C_i = \{y_{i1}, y_{i2}\}$, assume without loss of generality that the literal y_{i1} is TRUE under t, then we assign $\bar{z}_i$ to be TRUE and the two corresponding clauses $\{\mathbf{y_{i1}}, \mathbf{y_{i2}}, z_i\}$ and $\{\mathbf{y_{i1}}, \mathbf{y_{i2}}, \bar{z}_i\}$ are also TRUE. If C_i contains exactly three literals, then we can keep the truth assignment the same. Lastly, when C_i has four or more literals, without loss of generality we assume some y_{ij_0} is TRUE. In this case, we can "propagate" to both sides the TRUE/FALSE assignments of the z_{ij}, namely we assign $z_{i(j_0-2)}, \ldots, z_{i1}$ to be TRUE, and $\bar{z}_{i(j_0-1)}, \ldots, \bar{z}_{i(k-3)}$ to be TRUE.

Conversely, suppose $\mathcal{C}'$ is satisfiable with a truth assignment t'. A simple case by case analysis implies that the same assignment restricted only to the y's satisfies $\mathcal{C}$. Let us consider one of the cases, the reader is encouraged to complete the rest of the proof. Consider the four clauses corresponding to some C_i of size one:

$$\{\mathbf{y_{i1}}, z_{i1}, z_{i2}\}, \{\mathbf{y_{i1}}, z_{i1}, \bar{z}_{i2}\}, \{\mathbf{y_{i1}}, \bar{z}_{i1}, z_{i2}\}, \{\mathbf{y_{i1}}, \bar{z}_{i1}, \bar{z}_{i2}\}.$$

If these four clauses are satisfiable, then some pair of z_{i1}, z_{i2} and their negations have to be both FALSE, in which case the corresponding clause is TRUE if and only if y_{i1} was assigned TRUE. □

Theorem 5.2. *VERTEX-COVER is* $\mathbf{NP}$*-complete.*

Proof. □

Corollary 5.3. *Both CLIQUE and IS are* $\mathbf{NP}$*-complete.*

Proof. This follows easily from the reductions we have done in Section 3 and the previous theorem. (One should not forget to check that CLIQUE and IS are in $\mathbf{NP}$. ☐

Theorem 5.4. *SUBSET-SUM is* $\mathbf{NP}$*-complete.*

Proof. ☐

Theorem 5.5. *HAMILTONIAN CYCLE is* $\mathbf{NP}$*-complete.*

Proof. ☐

Theorem 5.6. 3*-COLORABILITY is* $\mathbf{NP}$*-complete.*

Proof. ☐

6 Practice exercises

Exercise 6.1. Show that SET COVER is $\mathbf{NP}$-complete.

Exercise 6.2. Show that DOMINATING SET is $\mathbf{NP}$-complete.

Exercise 6.3. Show that KNAPSACK is $\mathbf{NP}$-complete.

Exercise 6.4. Show that TSP is $\mathbf{NP}$-complete.

Exercise 6.5. Show that TSP is $\mathbf{NP}$-complete.

Exercise 6.6. Show that k-COLORABILITY is $\mathbf{NP}$-complete.

References

[1] G. J. CHAITIN, *The limits of mathematics*, Springer Series in Discrete Mathematics and Theoretical Computer Science, Springer-Verlag Singapore, Singapore, 1998. A course on information theory and the limits of formal reasoning, With a foreword by Cristian Calude.

[2] ———, *The unknowable*, Springer Series in Discrete Mathematics and Theoretical Computer Science, Springer-Verlag Singapore, Singapore, 1999.

[3] ———, *A century of controversy over the foundations of mathematics*, in Finite versus infinite, Discrete Math. Theor. Comput. Sci. (Lond.), Springer, London, 2000, pp. 75–100.

[4] A. COBHAM, *The intrinsic computational difficulty of functions*, in Logic, Methodology and Philos. Sci. (Proc. 1964 Internat. Congr.), North-Holland, Amsterdam, 1965, pp. 24–30.

[5] S. A. COOK, *The complexity of theorem-proving procedures*, in Conference Record of the Third Annual ACM Symposium on the Theory of Computing, 1971, pp. 151–158.

[6] T. H. CORMEN, C. E. LEISERSON, R. L. RIVEST, AND C. STEIN, *Introduction to algorithms*, MIT Press, Cambridge, MA, second ed., 2001.

[7] M. DUMMETT, *Elements of intuitionism*, vol. 39 of Oxford Logic Guides, The Clarendon Press Oxford University Press, New York, second ed., 2000.

[8] J. EDMONDS, *Minimum partition of a matroid into independent subsets*, J. Res. Nat. Bur. Standards Sect. B, 69B (1965), pp. 67–72.

[9] M. R. GAREY AND D. S. JOHNSON, *Computers and intractability*, W. H. Freeman and Co., San Francisco, Calif., 1979. A guide to the theory of NP-completeness, A Series of Books in the Mathematical Sciences.

[10] R. M. KARP, *Reducibility among combinatorial problems*, in Complexity of computer computations (Proc. Sympos., IBM Thomas J. Watson Res. Center, Yorktown Heights, N.Y., 1972), Plenum, New York, 1972, pp. 85–103.

[11] P. W. SHOR, *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer*, SIAM J. Comput., 26 (1997), pp. 1484–1509.

[12] ———, *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer*, SIAM Rev., 41 (1999), pp. 303–332 (electronic).

[13] E. SNAPPER, *The three crises in mathematics: logicism, intuitionism and formalism*, Math. Mag., 52 (1979), pp. 207–216.

[14] R. TIESZEN, ed., *Perspectives on intuitionism*, Canadian Society for History and Philosophy of Mathematics, Winnipeg, MB, 1998. Philos. Math. (3) **6** (1998), no. 2.

[15] D. VAN DALEN, *The development of Brouwer's intuitionism*, in Proof theory (Roskilde, 1997), vol. 292 of Synthese Lib., Kluwer Acad. Publ., Dordrecht, 2000, pp. 117–152.

[16] J. VON NEUMANN, *A certain zero-sum two-person game equivalent to the optimal assignment problem*, in Contributions to the theory of games, vol. 2, Annals of Mathematics Studies, no. 28, Princeton University Press, Princeton, N. J., 1953, pp. 5–12.

[17] A. N. WHITEHEAD AND B. RUSSELL, *Principia mathematica to *56*, Cambridge Mathematical Library, Cambridge University Press, Cambridge, 1997. Reprint of the second (1927) edition.