Introduction to $\mathbf{NP}$-completeness

Hung Q. Ngô *

January 16, 2005

Abstract

This document is still under construction. It shall be updated whenever I find the time to do so. However, it consists of all the things we have (and will) discussed in class. You can certainly read the entire document regardless of some missing pieces.

1 A brief history of $\mathbf{NP}$-completeness theory

$\boxed{\text{In P or not in P, that's the question!}}$

Looking into introductory algorithm analysis and design textbooks such as [8], we find a lot of interesting, practically useful, 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 a minimum spanning tree of a connected graph $G = (V, E)$ takes $O(|V| \lg |V|)$, computing a single-source 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 and science areas such as networking, databases, electrics, etc. The obvious question is then: "can all *natural* problems be solved in polynomial time?" After decades of research in algorithms, we essentially 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. (For a list of hard optimization problems of this kind, see [9].) 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.
- 01 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]
- EUCLIDEAN TRAVELING SALESMAN PROBLEM (EUCLIDEAN TSP): find a shortest route for a salesman to visit each of the n given cities exactly once, and return to the starting city.

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

*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* (Vietnamese communist) 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.

Georg Cantor (1845–1918)

David Hilbert (1862–1943)

1. Email ask a Prof who taught "introduction to algorithms"
2. Give up
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 to succeed, it is **very** hard to show such a result.
   - For almost all 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. (This approach is somewhat cheating since we did not really showed that the problem is hard or easy.) A few questions arise: (a) what exactly do we mean by a "hard" problem; (b) and how do we show that two problems are "equivalently" hard?

Intuitively, a hard problem is a problem which does not have an efficient algorithm to solve. Two problems are equivalently hard when solving one problem efficiently implies solving the other efficiently. To answer these questions rigorously, we need a *computational model*, which is a formal tool to model computation and algorithms.

In order to see briefly the development of the field, one has to go back to Cantor, Russell, Hilbert, Gödel, Church, Turing, Cook/Levin, and Karp. Part of the following discussion is based on Chaitin's "The Unknowable" [3] (see also [2, 4]).

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*:

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

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

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*. The argument basically goes as follows. If S is finite, then obviously S has less number of members than its power set. The infinite case is much trickier. Consider the set of all natural numbers $\mathbb{N}$ and the set of all even natural numbers $\mathbb{E}$, for instance. There is a one-to-one correspondence between $\mathbb{N}$ and $\mathbb{E}$: for each $n \in \mathbb{N}$, we simply pair n with $2n \in \mathbb{E}$. In this sense, $\mathbb{N}$ and $\mathbb{E}$ have the same cardinality. We say that two sets have the same cardinality iff there is such a correspondence. If there was an injection from X into Y, then we say $|X| \leq |Y|$. Thus, clearly $|S| \leq |2^S|$. Can they have the same cardinality? Suppose there was a one-to-one correspondence $f : S \to 2^S$. Consider the set $T = \{a \mid a \in S, a \notin f(a)\}$. Clearly $T \in 2^S$. Let $b = f^{-1}(T)$. Note that $f(b) = T$. The diagonalization argument lies in the question: "is $b \in T$?" If $b \in T$, then by definition of T it must be the case that $b \notin f(b) = T$. If $b \notin T = f(b)$, then again by definition $b \in T$. Hence, the correspondence f cannot exist. This argument proves to be very influential in later development

Given that $|S| < |2^S|$, we can now 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

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

Why stop there? Keep going on, we get

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

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 proclaimed: "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 suffered long periods of depressions due to various controversies relating to his set theory, in particular because he could not resolve several paradoxes which came out of his theory.

What was the debate about? It has to do with the paradoxes of set theory.

Liar paradox:

Alonzo Church (1903–1995)

Alan Turing (1912–1954)

"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" [26] 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 [11, 21, 22, 24]).
- Formalism (Hilbert): let's eliminate from mathematics the uncertainties and ambiguities of natural language. (See [21]).

  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 mathematical statement is true or not? This is often referred to as **Entscheidungsproblem**, or *the decision problem*.

This is when Kurt Gödel comes into the picture. On Hilbert's second problem, Gödel basically showed in 1931 [14] that "mathematics" is either inconsistent or incomplete!

The next step is done by the great Alan Turing in 1936 [23]. 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 so-called *Church-Turing thesis*. In 1936, Alonzo Church published a paper [5] which showed the same result, basically saying that there is a statement in elementary number theory which is not decidable. He also invented λ*-calculus*, which was the foundation for functional programming languages such as Scheme and Lisp.

Church-Turing Thesis

The intuitive notion of computations and algorithms is captured by the Turing machine model

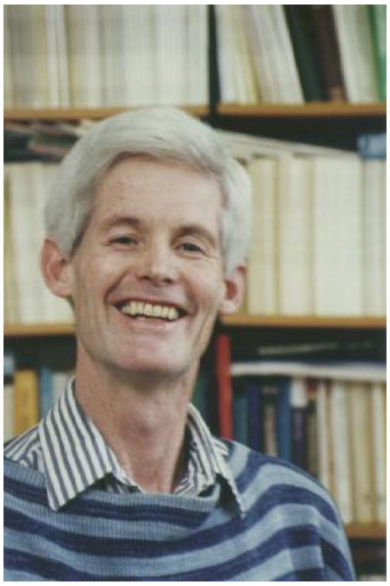
Now that we have a mathematical model for algorithms and computations, the next step is to decide what a hard problem is.

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 [25], Cobham [6], and Edmonds [12]. 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 implications. This is the same as asking: $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$. In the same spirit as Hilbert's problems, Clay Mathematics Institute has offered one million dollars to whoever solves one of the seven outstanding problems of the 20th century mathematics. One of the problems in the $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ question, another is, still, Hilbert's 8th problem – Riemann hypothesis.

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

In 1971, Stephen Cook [7] 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



Stephen Cook

Leonid Levin

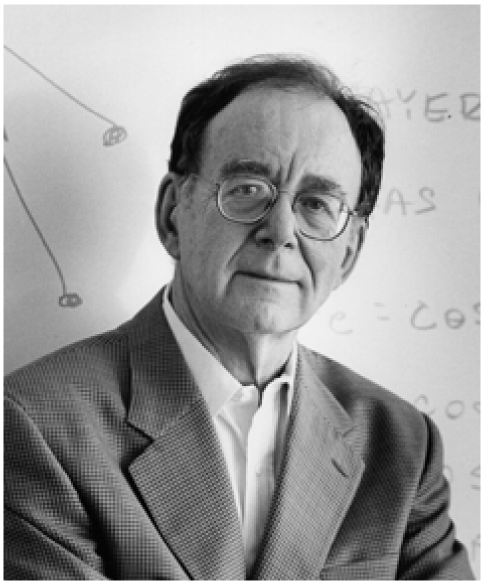
are the hardest ones in the class $\mathbf{NP}$. These problems are called the $\mathbf{NP}$-complete problems. Cook showed that SAT, 3-SAT, and SUBGRAPH ISOMORPHISM are $\mathbf{NP}$-complete. Independently, Leonid Levin published a paper in 1973 [18] which concluded with the same result that there are $\mathbf{NP}$-complete problems. In 1972, Richard Karp [15] found and showed that some 20 other very natural problems are also $\mathbf{NP}$-complete, such as VERTEX COVER, EUCLIDEAN TSP, CLIQUE, INDEPENDENT SET, ... In 1974, Knuth [16, 17] contributed to settling the terminologies on $\mathbf{NP}$-completeness theory as we see them today, partly because he wanted to use good termilonogies for volumn 4 of his famous series *The Art of Computer Programming*. Garey and Johnson's book [13] contains a brief discussion on the terminological changes and proposals at the time. There is one connection to Turing: Cook, Karp, and Knuth all got Turing awards.

TBD: more elaboration on the question $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ and the implications of a positive or a negative answer.

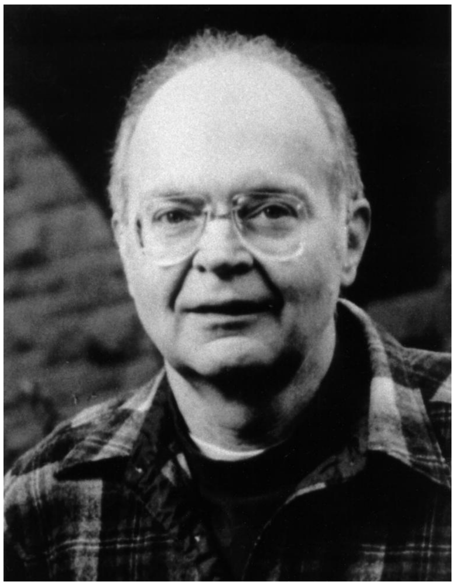
What do we do next? It is possible that the Turing machines are not as powerful as needed to solve hard problems of nature. The first direction to go is to come up with some stronger computational model: quantum computers, DNA computers, ... In fact, INTEGER FACTORIZATION, which is a problem not known 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 [19, 20]. However, when other computational models are still pretty much fictional, 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

TBD



Richard Karp



Donald Knuth

3 Polynomial time reducibility

The classic book on $\mathbf{NP}$-completeness theory is Garey and Johnson's "Computers and intractability" [13]. We shall conform to the convention that problem names are capitalized; e.g., EUCLIDEAN TSP, VC, CLIQUE, ...

3.1 Decision problems

For technical reasons, we only consider *decision problems*, i.e. problems which ask yes/no questions. Every optimization problems has a decision version. For example, the decision version of EUCLIDEAN TSP is: "given n cities and a bound $b \in \mathbb{Z}^+$, is there a TSP tour of length at most b?" Or, the decision version of VC asks if a graph G has a vertex cover of size at most b.

It should be evident that an optimization problem is at least as difficult than its decision version. For example, if we can solve EUCLIDEAN TSP in poly-time, then we only have to check if the optimal solution has length at most b 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 of 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 vertex cover 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 we often use. 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 depends on the input size, and thus 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 seems to be 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 $m = \lceil \lg n \rceil$, then the running time is actually exponential in m: $O(2^m m)$.

Henceforth, we shall assume *reasonable* encoding schemes. The reader is referred to [13] 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 (to be defined later) is $\mathbf{NP}$-complete. The reason for this informality is that rigorous formulations 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

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

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

$$\begin{aligned} x \in \Pi_{\text{YES}} &\implies A(x) = \text{YES}, \\ x \in \Pi_{\text{NO}} &\implies A(x) = \text{NO}. \end{aligned}$$

Here, $A(x)$ is 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 $\mathbf{NP}$ (read "solvable in nondeterministic polynomial time") if there is a poly-time *verification* algorithm $V(\cdot, \cdot)$, such that for any instance $x \in \Pi$,

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

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

For example, in the VERTEX COVER 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 b 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, VERTEX COVER is in $\mathbf{NP}$.

Similarly, given an instance of EUCLIDEAN TSP and a sequence y of cities, 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 b of the problem. Hence, EUCLIDEAN 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 $x \in \Pi$,

$$\begin{aligned} x \in \Pi_{\text{YES}} &\implies f(x) \in \Pi'_{\text{YES}} \\ x \in \Pi_{\text{NO}} &\implies f(x) \in \Pi'_{\text{NO}}. \end{aligned}$$

Note that the above statement is equivalent to saying

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

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 (see Figure 1).

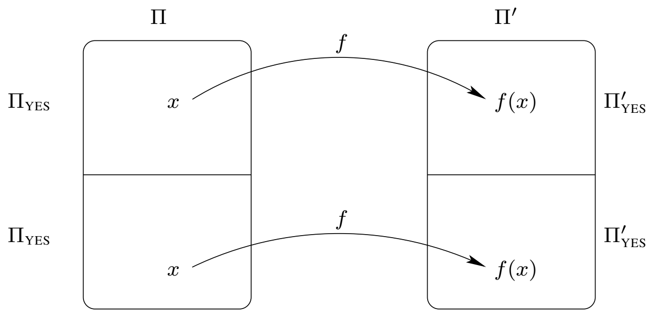


Figure 1: $\Pi \leq_p \Pi'$

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 $x' = (G', b') \in$ IS, such that G has a vertex cover of size at most b if and only if G' has an independent set 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. $\square$

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. Thus, G has an independent set of size at least b if and only if $\bar{G}$ has a clique of size at least b. $\square$

The following lemmas are trivial to show, yet are important properties of the 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* $\Pi' \in \mathbf{NP}$*, and* $\Pi \leq_p \Pi'$*, then* $\Pi \in \mathbf{NP}$.

Lemma 3.5. *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 $\mathbf{NP}$*-hard* if every problem in $\mathbf{NP}$ is polynomial-time reducible to Π.

A problem Π is $\mathbf{NP}$-*complete* (or $\in \mathbf{NPC}$) if and only if $\Pi \in \mathbf{NP}$ and Π is $\mathbf{NP}$-hard.

Lemma 3.6. *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 3.5, which says that the relation $\leq_p$ is transitive. $\square$

4 Cook's theorem

Now that the intuitive notion of an algorithm has been formalized by the concept of a 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 $t : X \to \{\text{TRUE}, \text{FALSE}\}$. Basically, t assigns to each variable in X a value of either TRUE or 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* by a truth assignment t iff at least one of its members is TRUE 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:

$$C = (x_1 \vee \bar{x}_3 \vee x_4).$$

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 $\mathcal{C}$.

In other words, we want to know if there is a truth assignment for which $\phi(t(x_1), \ldots, t(x_n)) =$ TRUE, given a boolean formula ϕ under *conjunctive normal form* (CNF). For instance,

$$\phi(x_1, \ldots, x_n) = (x_1 \vee \bar{x}_3 \vee x_4) \wedge (x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3).$$

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

Proof. **TBD**. □

5 Proving NP-completeness results

Given SAT as a basis, Lemma 3.6 helps show 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 in most cases is 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 the classic text by Garey and Johnson [13].

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 all clauses 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 a subset $S \subseteq V$, $|S| \leq b$, 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 PROBLEM (TSP)

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

QUESTION: Is there a TSP tour with total cost at most b?

EUCLIDEAN TRAVELING SALESMAN (EUCLIDEAN TSP)

INSTANCE: A set $C = \{p_1, \ldots, p_n\}$ of n points on a plane given by their x, y-coordinates $p_i = (x_i, y_i) \in \mathbb{Z} \times \mathbb{Z}$, 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 2 is our road map of the 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.

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 $C_i = \{y_{i_1}, \ldots, y_{i_{j_i}}\}$, namely there are j_i literals in C_i indexed from i_1 to i_{j_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_{i_1}. The corresponding clauses of $\mathcal{C}'$ are

$$\{\mathbf{y_{i_1}}, z_{i_1}, z_{i_2}\}, \{\mathbf{y_{i_1}}, z_{i_1}, \bar{z}_{i_2}\}, \{\mathbf{y_{i_1}}, \bar{z}_{i_1}, z_{i_2}\}, \{\mathbf{y_{i_1}}, \bar{z}_{i_1}, \bar{z}_{i_2}\}.$$

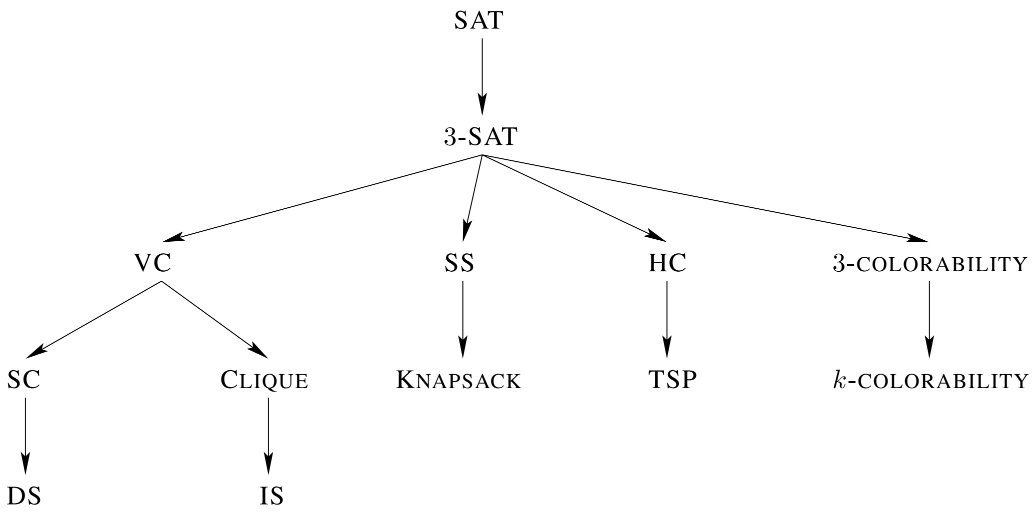


Figure 2: Map of our reductions to show $\mathbf{NP}$-complete problems.

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

The second case is when $C_i = \{y_{i_1}, y_{i_2}\}$. The corresponding clauses are

$$\{\mathbf{y_{i_1}}, \mathbf{y_{i_2}}, z_i\}, \{\mathbf{y_{i_1}}, \mathbf{y_{i_2}}, \bar{z}_i\}.$$

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 $C_i = \{y_{i_1}, y_{i_2}, \ldots, y_{i_k}\}$, $k \geq 4$. The corresponding clauses are

$$\{\mathbf{y_{i_1}}, \mathbf{y_{i_2}}, z_{i_1}\}, \{\bar{z}_{i_1}, \mathbf{y_{i_3}}, z_{i_2}\}, \{\bar{z}_{i_2}, \mathbf{y_{i4}}, z_{i_3}\}, \ldots, \{\bar{z}_{i_{k-3}}, \mathbf{y_{i_{k-1}}}, \mathbf{y_{i_k}}\}.$$

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

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

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 $C_i = \{y_{i_1}\}$, then we could assign any value to the z_{i_j}, while making the four corresponding clauses in $\mathcal{C}'$ TRUE. When $C_i = \{y_{i_1}, y_{i_2}\}$, assume without loss of generality that the literal y_{i_1} is TRUE under t, then we assign $\bar{z}_i$ to be TRUE and the two corresponding clauses $\{\mathbf{y_{i_1}}, \mathbf{y_{i_2}}, z_i\}$ and $\{\mathbf{y_{i_1}}, \mathbf{y_{i_2}}, \bar{z}_i\}$ 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 $y_{i_{j_0}}$ is TRUE. In this case, we can "propagate" to both sides the TRUE/FALSE assignments of the z_{i_j}, namely we assign $z_{i_{j_0-2}}, \ldots, z_{i_1}$ 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:

$$\{\mathbf{y_{i_1}}, z_{i_1}, z_{i_2}\}, \{\mathbf{y_{i_1}}, z_{i_1}, \bar{z}_{i_2}\}, \{\mathbf{y_{i_1}}, \bar{z}_{i_1}, z_{i_2}\}, \{\mathbf{y_{i_1}}, \bar{z}_{i_1}, \bar{z}_{i_2}\}.$$

If these four clauses are satisfiable, then some pair of z_{i_1}, z_{i_2} and their negations have to be both FALSE, in which case the corresponding clause is TRUE if and only if y_{i_1} was assigned TRUE. ☐

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

Proof. VERTEX COVER is in $\mathbf{NP}$, since given a subset of vertices we can easily verify that it is a vertex cover of size at most b.

To show that VC is $\mathbf{NP}$-complete, we will prove 3-SAT $\leq_p$ VC. Given an instance $\mathcal{C} = \{C_1, \ldots, C_m\}$ of 3-SAT on n variables $X = \{x_1, \ldots, x_n\}$, we will construct a graph G in polynomial time such that $\mathcal{C}$ is satisfiable if and only if G has a vertex cover of size at most $2m + n$. Figure illustrates our construction. The graph G consists of exactly $2n + 3m$ vertices, $2n$ of which correspond to $2n$ literals

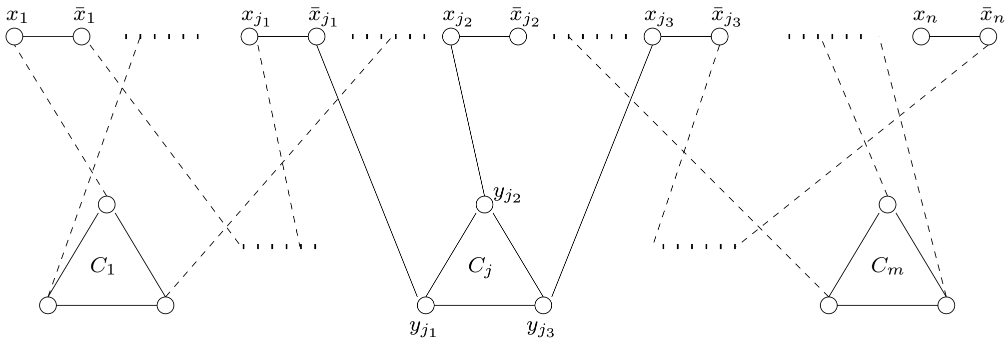


Figure 3: 3-SAT $\leq_p$ VERTEX COVER. Assume $C_j = \{\bar{x}_{j_1}, x_{j_2}, x_{j_3}\}$.

$x_1, \bar{x}_1, \ldots, x_n, \bar{x}_n$, and the other $3m$ vertices correspond to m clauses. Each clause is represented by a triangle in the graph. There is an edge between each vertex labelled x_i and its negation $\bar{x}_i$. If clause C_j consists of literals $y_{j_1}, y_{j_2}, y_{j_3}$, then the tree vertices of the triangle corresponding to C_j will be connected to the three vertices corresponding to the three literals. It is easy to see that the construction of G can be done in polynomial time.

We now show that $\mathcal{C}$ is satisfiable iff G has a vertex cover of size at most $2m + n$.

Suppose first that $\mathcal{C}$ is satisfiable with some truth assignment $t : X \to \{\text{TRUE}, \text{FALSE}\}$. We can construct a vertex cover S of G of size $n + 2m$ as follows. If $t(x_i) = \text{TRUE}$, then vertex x_i is in S, otherwise vertex $\bar{x}_i$ is in S. This accounts for n vertices of S. For each clause C_j, one of its literals, say y_{j_1}, is TRUE since C_j is satisfied by t. We put the other two vertices of the triangle corresponding to y_{j_2} and y_{j_3} in S. This accounts for the other $2m$ vertices of S. To see that S is a vertex cover, first consider the edges connecting a variable x_i and its negation $\bar{x}_i$. Either x_i or $\bar{x}_i$ is in S, hence this edge is covered. Second, consider the edges in the triangles. Two out of three vertices in each triangle are picked, hence all three edges in the triangle are covered by S. Lastly, consider the three edges connecting a triangle C_j and the $x_i, \bar{x}_i$ vertices. Two out of these three edges are covered by the two vertices in S of the C_j-triangle. The other edge is covered since the other literal is in S.

Conversely, suppose there is a vertex cover S of G of size at most $n + 2m$. In order to cover three edges of a triangle, at least two vertices in the triangle have to be in S. In order to cover the $(x_i, \bar{x}_i)$ edge, at least one of x_i and $\bar{x}_i$ has to be in S. Hence, in fact $|S| = n+2m$ and S consists of exactly two vertices out of each triangle and one out of each pair $x_i, \bar{x}_i$. Consider a truth assignment $t : X \to \{\text{TRUE}, \text{FALSE}\}$ where $t(x_i) = \text{TRUE}$ if vertex x_i is in S, and $t(x_i) = \text{FALSE}$ if vertex $\bar{x}_i$ is in S. We claim that all clauses are satisfied by this truth assignment. Consider a clause C_j. Let's look at the three edges coming out of triangle C_j. The two vertices in S of this triangle can only cover two of these edges. Hence, the other edge has to be covered by a vertex corresponding to the other literal, say y_{j_1}, which means $y_{j_1} = \text{TRUE}$ under t, i.e. C_j is satisfied. □

Remark 5.3. The $\mathbf{NP}$-completeness proofs of 3-SAT and VC were presented in their full rigor. You can see that the actual ideas were much simpler than the lengths of the proofs would otherwise indicate. From this point on, I will present proofs so that they convey the main ideas without too many notational details. You should follow the formats of the previous two proofs when working on homework problems. The wording can be shorter, as long as the main idea gets across.

Corollary 5.4. *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 5.5. SUBSET SUM *is* $\mathbf{NP}$*-complete.*

Proof. SUBSET SUM is obviously in $\mathbf{NP}$. We will show that 3-SAT $\leq_p$ SUBSET SUM. Consider an instance $\mathcal{C} = \{C_1, \dots, C_m\}$ of 3-SAT on n variables $X = \{x_1, \dots, x_n\}$. We will construct an instance of SUBSET SUM as follows.

The set S of natural numbers consists of $2n + 2m$ numbers:

- For each literal y there corresponds a member $a^{(y)}$ of S whose binary representation consists of $n+3m$ bits. Suppose y is either x_i or $\bar{x}_i$ for some $i \in \{1, \dots, n\}$. Write $a^{(y)} = a^{(y)}_{n+3m} a^{(y)}_{n+3m} \dots a^{(y)}_1$. The n most significant bits are all 0, except for $a^{(y)}_{n+3m-i+1} = 1$. If literal y appears in clause C_j, then $a^{(y)}_{3j-2} = 1$. The rest of the bits of $a^{(y)}$ are 0.
- There are $2m$ other "auxiliary" numbers $a^{(j,1)}$ and $a^{(j,2)}$ for $j = 1, \dots, m$. In binary format, these numbers have $3m$ bits each. Write $a^{(j,1)} = a^{(j,1)}_{3m} \dots a^{(j,1)}_1$ and $a^{(j,2)} = a^{(j,2)}_{3m} \dots a^{(j,2)}_1$, then only $a^{(j,1)}_{3j-2} = 1$ and $a^{(j,2)}_{3j-1} = 1$, while the rest of their bits are 0.

The target t is also a $(n + 3m)$-bit binary number. The n most significant bits of t are all 1. The last $3m$ bits go $100100 \dots 100$.

We claim that $\mathcal{C}$ has a satisfying truth assignment if and only if there is a subset of numbers in S whose sum is t. The basic idea is as follows.

We have to make sure that we cannot take $a^{(x_i)}$ and $a^{\bar{x}_i}$ at the same time. This is guaranteed by the fact that the n most significant bits of t are all 1. If we take both $a^{(x_i)}$ and $a^{\bar{x}_i}$, then the corresponding bit in the sum will be 0 (at least for the largest i for which both $a^{(x_i)}$ and $a^{\bar{x}_i}$ are taken).

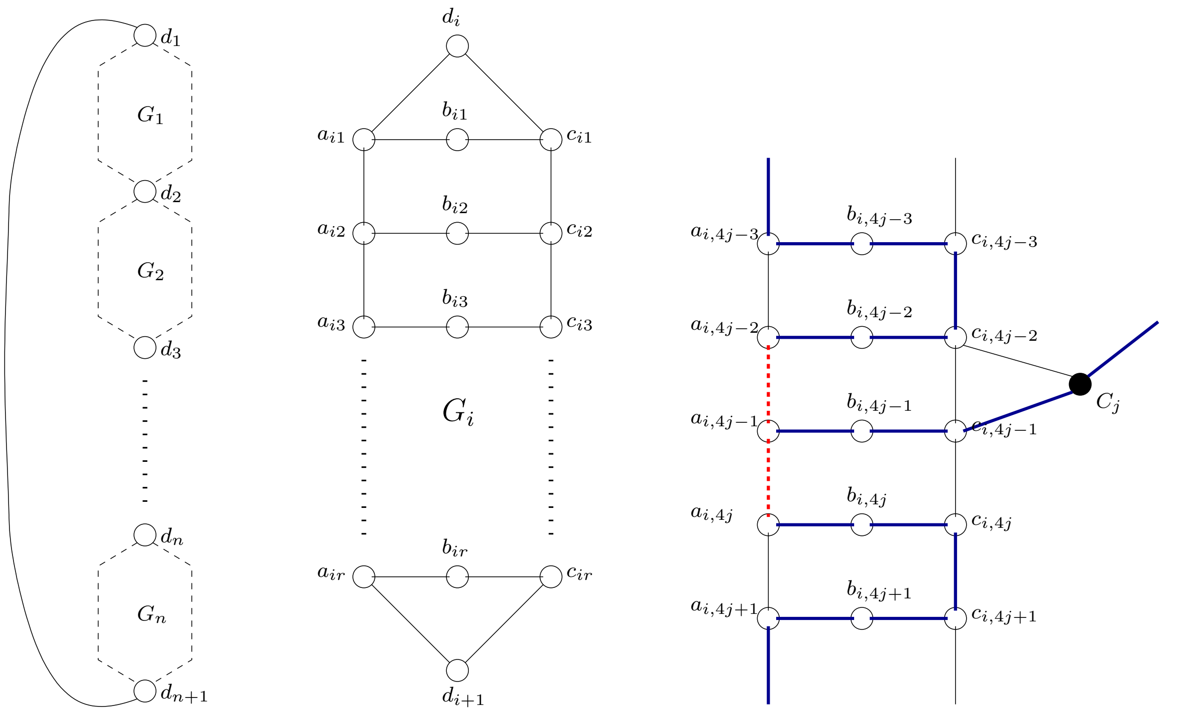
The $3m$ least significant bits are divided into blocks of size 3 each, where the jth block (from the right) is used to verify that C_j is satisfied. If some $a^{(y)}$ is chosen, then there is a 1 in all the blocks corresponding the C_j in which y occurs, and C_j is satisfied. However, C_j can be satisfied by one, two, or three of its literal. The corresponding block in t is 100, representing number 4. If C_j is satisfied by 1 literal, then we can pick up both $a^{(j,1)}$ and $a^{(j,2)}$ so that the sum in the block is 4. If C_j is satisfied by 2 or 3 literals, then we pick either $a^{(j,2)}$ or $a^{(j,1)}$ to "fill in." If C_j is not satisfied, then the sum of $a^{(j,2)}$ and $a^{(j,1)}$ is not sufficient to make the result 4.

Lastly, notice that even when C_j is satisfied by 3 literals, and when both $a^{(j,2)}$ and $a^{(j,1)}$ are picked, the total is only 7 – or 111 in binary. Hence, there cannot be any carry-over bit from the $3m$ least significant bits to the higher order bits.

The rest of the proof follows straightforwardly from the above analysis. □

Theorem 5.6. HAMILTONIAN CYCLE *is* $\mathbf{NP}$*-complete.*

Proof. HC is obviously in $\mathbf{NP}$. We will show that 3-SAT $\leq_p$ HC with a reduction by Du and Ko [10]. Given an instance $\mathcal{C} = \{C_1, \ldots, C_m\}$ of 3-SAT on n variables $x_1, \ldots x_n$, we construct a graph G which has a Hamiltonian cycle iff $\mathcal{C}$ is satisfiable. Figure illustrates our construction.



(a) General view (b) Component G_i corresponds to x_i (c) Connect to C_j

Figure 4: 3-SAT $\leq_p$ HAMILTONIAN CYCLE.

The construction goes as follows. The graph G consists of a series of components $G_1, \ldots, G_n$ which corresponds to variables $x_1, \ldots, x_n$ as shown in Figure 4(a). Each of G_i looks that that in Figure 4(b), where $r = 4m$.

Consider the G_i component. There are exactly two possible paths from d_i to d_{i+1} which cover all vertices of the component. Either we enter with (d_i, a_{i1}) or with (d_i, b_{i1}). The rest of such path is determined by the first move: just go zig-zag'ing to the end. Hence, there are 2^n Hamiltonian paths through G, corresponding to 2^n possible truth assignments to the variables x_i. If a path enters G_i at a_{i1}, then consider x_i to be TRUE, otherwise $\bar{x}_i$ is TRUE.

Note that a Hamiltonian path from d_i to d_{i+1} corresponding to setting $x_i = $ TRUE must pass through edge $(a_{i,4j-2}, a_{i,4j-1})$, for all $j = 1, \ldots, m$.

Now, we add m more vertices labelled $C_1, \ldots, C_m$. If C_j contains x_i, then we add two edges from C_j to $a_{i,4j-2}$ and $a_{i,4j-1}$. If C_j contains $\bar{x}_i$, then we add two edges from C_j to $c_{i,4j-2}$ and $c_{i,4j-1}$. Thus, each C_j has degree exactly 6.

The crucial point to notice is that, we cannot construct a Hamiltonian cycle which passes through some C_j from a vertex in some G_i to a vertex in some $G_{i'}$ with $i \neq i'$. In other words, in order to cover C_j we must go through vertices as if the C_j didn't exist, and then replace "locally" some edge $(a_{i,4j-2}, a_{i,4j-1})$ or $(c_{i,4j-2}, c_{i,4j-1})$ by "stopping by" C_j in between. To see that this claim is true, suppose – as shown in Figure 4(c) – that we passes through C_j from some other component and enters G_i at $c_{i,4j-1}$, say. Note that any Hamiltonian cycle has to pass through the horizontal edges to cover the $b_{p,q}$. Hence, both edges $(c_{i,4j-1}, b_{i,4j-1})$ and $(b_{i,4j-1}, a_{i,4j-1})$ are in the current path. Moreover, it is easy to see that both $(c_{i,4j-2}, c_{i,4j-3})$ and $(c_{i,4j}, c_{i,4j+1})$ are in the cycle. We now have a dilemma at vertex $a_{i,4j-1}$ since we do not know where to go from it. Note that, x_i and $\bar{x}_i$ cannot both be in C_j.

The rest of the proof now follows straightforwardly. A Hamiltonian cycle corresponds to a truth assignment where occasionally the cycle "stop by" C_j to say that my assignment satisfies C_j. The reader should have no trouble completing the proof. $\square$

Theorem 5.7. 3-COLORABILITY *is* $\mathbf{NP}$*-complete.*

Proof. It's obvious that 3-COLORABILITY is in $\mathbf{NP}$. We will reduce 3-SAT to it. Given an instance $\mathcal{C} = \{C_1, \ldots, C_m\}$ of 3-SAT on n variables $X = \{x_1, \ldots, x_n\}$, we construct an instance of 3-COLORABILITY as follows. Figure illustrates our construction. There is a *top vertex* v. For each variable

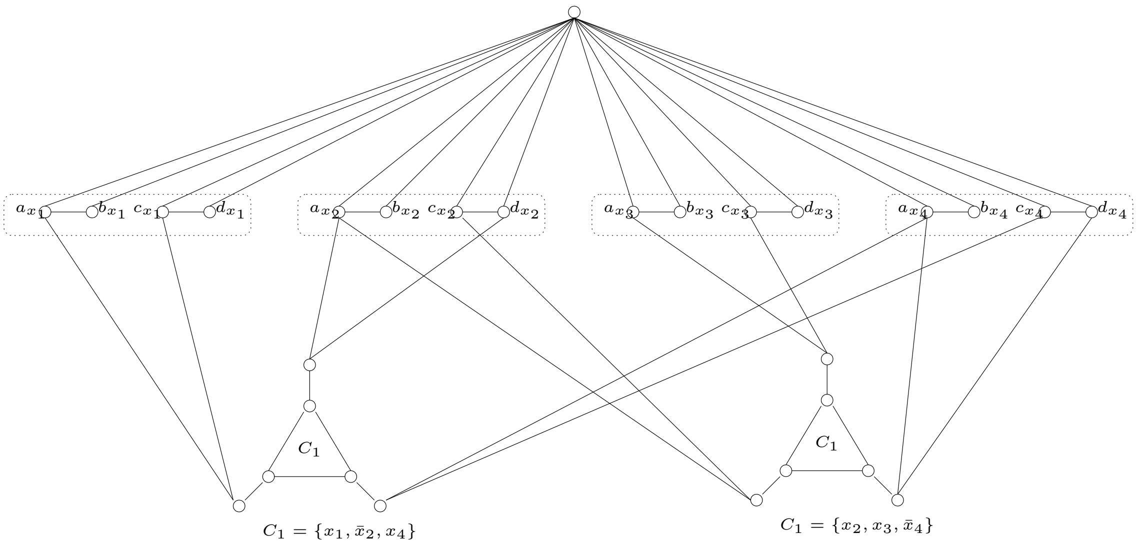


Figure 5: 3-SAT $\leq_p$ 3-COLORABILITY.

x_i, create 4 vertices $a_{x_i}, b_{x_i}, c_{x_i}$, and d_{x_i}. For each clause C_j, create a triangle with three extra edges pointing out to three extra vertices as shown in the figure. Each extra vertex corresponds to one of the three literals y_{j_1}, y_{j_2}, or y_{j_3} of the clause. If the literal is not a negation, say the literal is x_i, then connect the extra vertex x_i (of the clause' triangle) to a_{x_i} and c_{x_i}. If the literal is $\bar{x}_i$, then connect it to a_{x_i} and d_{x_i}.

We will show that $\mathcal{C}$ is satisfiable iff G is 3-colorable.

The main idea is that, for each triangle C_j, if the tree extra vertices get the same color then there is no way to color the inner three vertices with three colors. On the other hand, if there are at least two outside colors, then the inner vertices can always be colored.

Suppose first that $\mathcal{C}$ is satisfiable with truth assignment t. Color the top vertex v with color 1. If $t(x_i) =$ TRUE then color a_{x_i} and c_{x_i} with 2, and b_{x_i} and d_{x_i} with 3. If $t(x_i) =$ FALSE then color a_{x_i} and d_{x_i} with 2, and b_{x_i} and c_{x_i} with 3. The idea is that if a literal, say x_i, is true then there will be two choices of colors for the triangles' extra vertices which correspond to the literal, because a_{x_i} and c_{x_i} get the same color. Because all C_j are satisfiable, each C_j-triangle has at least one outside vertex with two color choices. Thus the outside vertices of C_j can be colors so that at least two colors are used. Then, the inner vertices of C_j can be properly colored due to the observation above.

Conversely, suppose the graph G is 3-colorable, we can find a truth assignment by assigning x_i to be TRUE if a_{x_i} and c_{x_i} get the same color, and FALSE otherwise. It is easy to check that this assignment satisfies all of the clauses. Consider a clause C_j, for instance. If C_j is not satisfiable then all three outside vertices of its triangle must have the same color as the top vertex, which means the inner vertices of C_j were not properly colored.

$\square$

6 Exercises

Exercise 1. Show that SET COVER is $\mathbf{NP}$-complete.

Exercise 2. Show that DOMINATING SET is $\mathbf{NP}$-complete.

Exercise 3. Show that KNAPSACK is $\mathbf{NP}$-complete.

Exercise 4. Show that TSP is $\mathbf{NP}$-complete.

Exercise 5. Show that k-COLORABILITY is $\mathbf{NP}$-complete.

For the following exercises, assume that you only know the problems in our roadmap are $\mathbf{NP}$-complete.

Exercise 6. Give a polynomial time reduction from 3-SAT to CLIQUE.

Exercise 7 (**)**.** Give a polynomial time reduction from HAMILTONIAN CYCLE to VERTEX COVER.

Exercise 8 (*)**.** A *monotone boolean formula* is a boolean formula which involves only AND's and OR's. There is no negation. For instance, $\phi = (x_1 \wedge x_2) \vee (x_3 \vee (x_4 \wedge x_5))$ is monotone. Prove that the following problem is $\mathbf{NP}$-complete.

MONOTONE SATISFIABILITY (MON-SAT)

INSTANCE: A set X of variables, a monotone boolean formula ϕ over X, and a bound $b \in \mathbb{Z}^+$.

QUESTION: Is there a truth assignment which satisfies ϕ in which at most b variables are TRUE?

Exercise 9. Prove that the following problem is $\mathbf{NP}$-complete.

LONGEST PATH

INSTANCE: A graph G and a bound $b \in \mathbb{Z}^+$.

QUESTION: Is there a path in G of length at least b?

Exercise 10. Prove that the following problem is $\mathbf{NP}$-complete.

HITTING SET

INSTANCE: A collection $\mathcal{C} = \{S_1, \dots, S_m\}$ of non-empty subsets of a universe set S, and a bound $b \in \mathbb{Z}^+$.

QUESTION: Is there a subset $H \subseteq S$ with $|H| \leq h$, for which $S_i \cap H \neq \emptyset$, for all $i = 1, \dots, m$?

Exercise 11. Prove that the following problem is $\mathbf{NP}$-complete.

PARTITION

INSTANCE: A set $A = \{a_1, \dots, a_n\}$ of positive integers

QUESTION: Is there a partition of A into two subsets A_1 and A_2 such that

$$\sum_{a \in A_1} a = \sum_{a \in A_2} a.$$

Exercise 12. Prove that the following problem is $\mathbf{NP}$-complete.

i-PARTITION

INSTANCE: A set $S = \{s_1, \ldots, s_n\}$ of non-negative integers and a positive integer $i \le n$.

QUESTION: Is there a partition of S into i subsets $S_1, \ldots, S_i$ such that

$$\sum_{x \in S_1} x = \sum_{x \in S_2} x = \cdots = \sum_{x \in S_i} x.$$

Exercise 13. Prove that the following problem is $\mathbf{NP}$-complete.

SUBGRAPH ISOMORPHISM

INSTANCE: Two graphs G and H.

QUESTION: Is there a subgraph of G which is isomorphic to H?

Exercise 14. Prove that the following problem is $\mathbf{NP}$-complete.

LONGEST CYCLE

INSTANCE: A graph G and a bound $b \in \mathbb{Z}^+$.

QUESTION: Is there a cycle in G of length at least b?

Exercise 15 (*)**.** Prove that the following problem is $\mathbf{NP}$-complete.

3-DIMENSIONAL MATCHING (3-DM)

INSTANCE: Three disjoint sets A, B, C of size n each. A subset $W \subseteq A \times B \times C$.

QUESTION: Is there $M \subseteq W$ such that each element of $A \cup B \cup C$ appears in M exactly once.

For all problems we have seen so far, the fact that they are in $\mathbf{NP}$ are often trivial. This is not the case with a few important problems, as the following exercises show.

Exercise 16 (Subtlety of $\mathbf{NP}$ membership)**.** Consider the following problem (known to be in $\mathbf{NP}$):

INTEGER PROGRAMMING

INSTANCE: An $m \times n$ integer matrix A and a vector $b \in \mathbb{Z}^m$.

QUESTION: Is there a vector $x \in \mathbb{Z}^n$ such that $Ax \ge b$?

It is very tempting to say that INTEGER PROGRAMMING is in $\mathbf{NP}$ because, given a certificate vector $y \in \mathbb{Z}^n$ we can verify in polynomial time if $Ay \ge b$. What is wrong with the above reasoning?

Exercise 17 (INTEGER PROGRAMMING is in $\mathbf{NP}$ **)**.** Let A be an $m \times n$ integer matrix and b be a vector in $\mathbb{Z}^m$. Let α be the maximum value of integers in A and b. Let $q = \max\{m, n\}$.

(i) Let B be any square submatrix of A, prove that $|\det(B)| \le (\alpha q)^q$.

(ii) Prove that, if $\text{rank}(A) = r < m$, then there exists a non-zero integer vector z such that $Az = 0$ and the maximum absolute value of integers in z is at most $(\alpha q)^q$.

(iii) Suppose $Ax \ge b$ has an integer solution x. Let a_i denote the ith row of A, b_i the ith component of b, and e_j the jth m-dimensional unit vector. Let x be a solution for $Ax \ge b$ that maximizes the number of elements in the following set:

$$\begin{aligned} \mathcal{A}_x &= \{a_i \mid b_i \le a_i x \le b_i + (\alpha q)^{q+1}, i \in [n]\} \\ &\quad \cup \{e_j \mid |x_j| \le (\alpha q)^q, j \in [m]\}. \end{aligned}$$

Prove that $\text{rank}(\mathcal{A}_x) = m$.

(iv) Prove that, if there is an integer vector x satisfying $Ax \geq b$, then there is an integer vector y satisfying $Ay \geq b$ for which absolute values of integers in y are at most $2(\alpha q)^{2q+2}$.

(v) Prove that INTEGER PROGRAMMING is in $\mathbf{NP}$.

Exercise 18 (PRIME is in $\mathbf{NP}$ **)**.** The PRIME problem is the problem of determining whether an input number n is prime or composite. In August of 2002, Agrawal, Kayal, and Saxena of IIT Kanpur showed that PRIME is in $\mathbf{P}$. Their proof is actually accessible to any one with rudimentary knowledge of abstract algebra. (See [1] for an interesting story.)

Prove a much weaker result that PRIME $\in \mathbf{NP}$ using the following known fact: an odd integer $p > 2$ is a prime if and only if there exists an integer $a \in \{1, 2, \ldots, p-1\}$ such that

(i) $a^{p-1} \equiv 1 \pmod{p}$.

(ii) $a^{(p-1)/q} \not\equiv 1 \pmod{p}$ for all prime factors q of $p-1$.

References

[1] F. BORNEMANN, *PRIMES is in P: a breakthrough for "Everyman"*, Notices Amer. Math. Soc., 50 (2003), pp. 545–552.

[2] 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.

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

[4] ———, *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.

[5] A. CHURCH, *An unsolvable problem in elementary number theory*, American Journal of Mathematics, 58 (1936), pp. 345–363.

[6] 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.

[7] 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.

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

[9] P. CRESCENZI AND V. K. (EDS.), *A compendium of NP-optimization problems*. http://www.nada.kth.se/

[10] D.-Z. DU AND K.-I. KO, *Theory of computational complexity*, Wiley-Interscience Series in Discrete Mathematics and Optimization, Wiley-Interscience, New York, 2000.

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

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

[13] 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.

[14] K. GÖDEL, *Über formal unentscheidbare sätze der principia mathematica und verwandter systeme i*, Monatshefte für Mathematik und Physik, 38 (1931), pp. 173–198. English translation, "On formally undecidable propositions of Principia mathematica and related systems I.", in van Heijenoort, J. (ed.) (1967) From Frege to Gdel: A Source Book in Mathematical Logic 1879–1931. Harvard University Press. pp 592–618.

[15] 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.

[16] D. KNUTH, *Postscript about np-hard problems*, SIGACT News, 6 (1974), pp. 15–16.

[17] ———, *A terminological proposal*, SIGACT News, 6 (1974), pp. 12–18.

[18] L. LEVIN, *Universal search problems (in russian)*, Problemy Peredachi Informatsii, 9 (1973), pp. 265–266. English translation in Trakhtenbrot, B. A.: A survey of Russian approaches to Perebor (brute-force search) algorithms. Annals of the History of Computing, 6 (1984), pp 384–400.

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

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

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

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

[23] A. TURING, *On computable numbers with an application to the entscheidungsproblem*, Proc. London Math. Soc., 42 (1936), pp. 230–265. “A correction” ibid, 43, 544-546.

[24] 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.

[25] 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.

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