An Introduction to Complexity Theory*

Adithya Bhaskara

July 22, 2024

1 Motivation

In this set of notes, we seek to understand the relationships between how difficult different problems of interest are to solve. One of the most important open questions in mathematics and theoretical computer science is whether the set of problems that we can efficiently solve is equal to the set of problems that we can efficiently verify a correct answer to. In symbolic notation, does $\mathsf{P} = \mathsf{NP}$? Techniques from theoretical computer science have allowed us to pose this question in many equivalent ways, and computer scientists have defined a zoo of complexity classes [1] to try to better understand this problem. In the following sections, we will start to understand the basics of computational complexity theory. Courses at CU Boulder that study complexity theory, and more broadly theoretical computer science, in detail include CSCI 3434/5444, CSCI 4114/5114, and CSCI 6114.

2 Background

Before we define any complexity classes, we first need to understand the setting in which we study them. We start with the following definitions.

Definition 1 (Alphabet, Strings, Languages)**.** *An alphabet over n letters is a set Σ such that $|\Sigma| = n$. We denote the set of all possible finite strings over the alphabet Σ as Σ^*. Finally, a language is a set L contained by Σ^*; that is, $L \subseteq \Sigma^*$.*

Remark. *We usually take $\Sigma = \{0, 1\}$.*

Definition 2 (Decision Problems)**.** *A decision problem is a question concerning the membership of a string $x \in \Sigma^*$ in L. Given language L, we say that the decision problem encoded by L is the question $x \in L$? We often represent decision problems by providing an* Instance *of the problem and a question to* Decide*.*

We use Definitions 1 and 2 to ground our study of complexity theory mathematically. Any algorithmic problem can be framed as a decision problem over some alphabet Σ. We provide some examples.

Example 1. *The algorithmic problem of finding the shortest path between nodes s and t in a graph $G = (V, E, w)$ can be phrased as $x \in L$ for*

$$L = \{\langle G \rangle : \textsc{there exists a path between } s \textsc{ and } t \textsc{ in } G \textsc{ of cost at most } k\}.$$

*Originally written for CSCI 3104 at CU Boulder in the Spring 2024 semester in my capacity as a course assistant.

The angle brackets around "G" represent an encoding of G in terms of Σ.

Example 2. *The algorithmic problem of finding the maximal independent set in a graph $G = (V, E)$ can be phrased as $x \in L$ for*

$$L = \{\langle G\rangle : \text{THERE EXISTS AN INDEPENDENT SET } N \text{ OF } G \text{ WITH } |N| \geq k\}.$$

The angle brackets around "G" represent an encoding of G in terms of Σ.

Definition 3 (Efficiently Decidable)**.** *We say that a language L is efficiently decidable if there exists an algorithm that can decide if $x \in L$ in polynomial time.*

Remark. *Definition 3 is intentionally vague, and more precise statements are beyond the scope of CSCI 3104.*

We now define the complexity classes P and NP.

Definition 4 (P)**.** *We say that $L \in \mathsf{P}$ if there exists some algorithm $M(x)$ that decides L such that $T_M(n) \in O(n^k)$ for some $k \in \mathbb{N}$.*

Definition 5 (NP)**.** *We say that $L \in \mathsf{NP}$ if there exists some algorithm V with $T_V(n) \in O(n^k)$ for some $k \in \mathbb{N}$ such that for every $x \in L$, there exists some polynomially-sized $C \in \Sigma^*$ with $V(x, C) = 1$. We say that V is a verifier, and C is a certificate.*

In other words, P is the set of problems that we can efficiently decide, and NP is the set of problems that, given some attempt at the solution, we can check if the attempt gives a "yes" answer in polynomial time.

Remark. *The set of problems that we can efficiently verify a "yes" answer to, which is NP, is not necessarily the same as the set of problems that we can efficiently verify a "no" answer to. The latter set is referred to as coNP, and whether $\mathsf{NP} = \mathsf{coNP}$ is a major open problem in theoretical computer science and mathematics.*

Remark. *There is an alternate formulation of Definition 5 in terms of nondeterministic Turing machines. We do not state this definition as it introduces mathematical rigor beyond the scope of this course; however, this formulation is more foundational. See [4] for more detail.*

We now give some examples.

Example 3. *Finding the minimum-spanning-tree of a graph is in P since the algorithms of Prim and Kruskal are polynomial-time.*

Example 4. *Finding the minimum-spanning-tree of a graph is in NP since, given the graph and a tree, we can efficiently compute the size of the tree.*

Example 5. *Determining the maximum flow of a flow network is in P since the Ford-Fulkerson procedure gives rise to polynomial-time algorithms.*

Example 6. *Determining if a number is prime is in P by the polynomial-time AKS primality test [2].*

Example 7. *Determining if a Boolean formula has a satisfying assignment of variables is in NP. Given a certificate of variable assignments, we can efficiently check if the assignments are satisfying.*

Example 8. *Determining if a graph $G = (V, E)$ has a vertex cover of size at most k is in NP. Given a certificate $C \subseteq V$, we can efficiently check if C is a vertex cover and if $|V| \leq k$.*

Consider the following theorem.

Theorem 1. *We have that* $\mathsf{P} \subseteq \mathsf{NP}$.

Proof. Suppose $L \in \mathsf{P}$. Then, there exists some algorithm $M(x)$ that decides L such that $T_M(n) \in O(n^k)$ for $k \in \mathbb{N}$. Take $V = M(x, \cdot)$. That is, we can verify a "yes" answer to membership by deciding membership from scratch. ■

Example 9. *Determining if a number is prime is in* NP *since it is in* P.

The million dollar question is whether $\mathsf{P} = \mathsf{NP}$, or equivalently by Theorem 1, whether $\mathsf{NP} \subseteq \mathsf{P}$. Most computer scientists and mathematicians intuitively believe that $\mathsf{P} \neq \mathsf{NP}$, but no one to date has provided a proof. If indeed $\mathsf{P} = \mathsf{NP}$, we would be able to solve problems of great interest efficiently, but many procedures that depend on being difficult, like cryptography, would break. For a popular and not-so-technical introduction to P and NP, see [3].

To better understand how the complexities of solving different decision problems are related, we now introduce the notion of reductions.

3 Reductions and NP-Completeness

I can get to the Hale Science Building efficiently if I can get a bike. So, if I can get a bike efficiently, I can get to the Hale Science Building efficiently. Note that not being able to get a bike efficiently tells me nothing about if I can get to the Hale Science Building efficiently–maybe I drive there instead.

Let L_1 and L_2 be languages. If we can show that we can decide L_1 efficiently if we can decide L_2, the complexity of L_2 tells us a lot about the complexity of L_1. To formalize this idea, consider the following definitions and theorems.

Definition 6 (Reductions)**.** *We say that* L_1 *reduces to* L_2 *if there exists a function* $f : \Sigma^* \to \Sigma^*$ *such that* $x \in L_1$ *if and only if* $f(x) \in L_2$*; if* f *is computable in polynomial time, we say that* L_1 *polynomially reduces to* L_2 *and write* $L_1 \leq_p L_2$.

Lemma 1 (Transitivity of $\leq_p$)**.** *If* $L_1 \leq_p L_2$ *and* $L_2 \leq_p L_3$*, then* $L_1 \leq_p L_3$.

Proof. Suppose $L_1 \leq_p L_2$ and $L_2 \leq_p L_3$. So, there exist functions f and g, computable in polynomial time, with $x \in L_1$ if and only if $f(x) \in L_2$ and $y \in L_2$ if and only if $g(y) \in L_3$. We note $g \circ f$ is computable in polynomial time. Then, $x \in L_1$ if and only if $g(f(x)) \in L_3$. So, $L_1 \leq_p L_3$. ■

Remark. *When giving a reduction from* L_1 *to* L_2*. To prove correctness of the reduction, one must explicitly show how to compute* f*, and* $x \in L_1 \implies f(x) \in L_2$*,* $f(x) \in L_2 \implies x \in L_1$*. For polynomial reductions, one must additionally show that* f *is computable in polynomial time.*

Theorem 2. *If* $L_1 \leq_p L_2$ *and* $L_2 \in \mathsf{P}$*, then* $L_1 \in \mathsf{P}$.

Proof. To decide if $x \in L_1$ in polynomial time. Compute the reduction function $f(x)$ in polynomial time, and then decide $f(x) \in L_2$ in polynomial time since $L_2 \in \mathsf{P}$. Then, $x \in L_1$ if and only if $f(x) \in L_2$. ■

Example 10. *Finding the minimum cut of a graph is in* P*. We can reduce the minimum cut problem to the maximum flow problem. Given a maximum flow, we can always find the minimum cut.*

We can use reductions to find the hardest problems in NP. Intuitively, if we could solve one of the hardest problems in NP, we should be able to solve all the other problems in NP. This motivates the following definitions.

Definition 7 (NP-Hardness)**.** *We say L is* NP*-hard if for every $K \in \mathsf{NP}$, $K \leq_p L$.*

Definition 8 (NP-Completeness)**.** *We say L is* NP*-complete if $L \in \mathsf{NP}$ and L is* NP*-hard.*

We now state and prove the fundamental theorems of NP-completeness.

Theorem 3 (Fundamental Theorem of NP-Completeness I, Importance)**.** *If L is* NP*-complete and $L \in \mathsf{P}$, then $\mathsf{P} = \mathsf{NP}$.*

Proof. We need only show $\mathsf{NP} \subseteq \mathsf{P}$. Take $K \in \mathsf{NP}$. By definition of NP-completeness, $K \leq_p L$ so we are guaranteed the existence of a polynomial-time algorithm to decide L. So, to decide membership of K in polynomial time for any $x \in \Sigma^*$, we efficiently compute $f(x)$ and then determine $f(x) \in L$. Recall $x \in K$ if and only if $f(x) \in L$. So, we have given a polynomial-time algorithm to decide K, so $K \in \mathsf{P}$. Therefore, $\mathsf{NP} \subseteq \mathsf{P}$, as desired. ■

So, if we can show a polynomial-time algorithm for an NP-complete problem, we would have shown that $\mathsf{P} = \mathsf{NP}$.

Theorem 4 (Fundamental Theorem of NP-Completeness II, Practicality)**.** *Let $L_2 \in \mathsf{NP}$. If L_1 is* NP*-complete, and $L_1 \leq_p L_2$, then L_2 is* NP*-complete.*

Proof. Take $K \in \mathsf{NP}$. We wish to show $K \leq_p L_2$ and we are done. We know that $K \leq_p L_1$, and by assumption, $L_1 \leq L_2$, so $K \leq_p L_2$ by transitivity of $\leq_p$. So, L_2 is NP-complete. ■

Theorem 3 helps us understand why the definition of NP-completeness is so important in that a polynomial-time solution to any NP-complete problem would give rise to a polynomial-time solution for all problems in NP. Theorem 4 helps us find NP-complete problems.

For our first NP-complete problem, we define the Boolean satisfiability problem.

Definition 9 (Boolean Satisfiability (SAT))**.** *The Boolean Satisfiability problem, or SAT, is given by the following.*

- *Instance: A Boolean formula $\varphi : \{0,1\}^n \to \{0,1\}$ confined to the $\wedge$, $\vee$, and $\neg$ operations.*
- *Decide: Does there exist $\vec{x} \in \{0,1\}^n$ with $\varphi(\vec{x}) = 1$?*

Note that we call $\vec{x}$ a satisfying assignment of φ. Formally, we can write $SAT = \{\langle\varphi\rangle : \varphi \text{ is satisfiable}\}$.

Theorem 5 (Cook-Levin)**.** *SAT is* NP*-complete.*

By Theorem 3, if we could find a polynomial-time algorithm for SAT, we would have concluded $\mathsf{P} = \mathsf{NP}$. By Theorem 4, if we can reduce SAT to some language L, then L is also NP-complete. The proof of Theorem 5 is beyond the scope of CSCI 3104, but interested students may refer to [4] for a proof.

We now state another useful theorem.

Theorem 6. *3-SAT, given by*

- *Instance: A Boolean formula $\varphi : \{0,1\}^n \to \{0,1\}$ in conjunctive normal form where all clauses have three literals.*
- *Decide: Does there exist $\vec{x} \in \{0,1\}^n$ with $\varphi(\vec{x}) = 1$?*

is NP*-complete.*

Proof. Exercise. ■

The proof of Theorem 6 is beyond the scope of CSCI 3104, but is not unapproachable for interested students. Can you find a way to convert a Boolean formula to an equivalent one in conjunctive normal form with the constraint that all clauses have three literals?

Remark. *Many students have difficulty with the intuition on which "direction" reductions to show* NP*-completeness go. To conclude some problem $L_2 \in$ NP is* NP*-complete, we must reduce from another problem L_1 that we already know to be* NP*-complete. That is, we must show $L_1 \leq_p L_2$. The idea is similar to a proof by contradiction. That is, we suppose for contradiction, that we can solve L_2 in polynomial time. But then, we must be able to solve L_1 in polynomial time. Since L_1 is* NP*-complete, then* P = NP*. If* P = NP*, then, of course we can solve L_2 in polynomial time since $L_2 \in$* NP*. If* P ≠ NP*, we have a contradiction and so we cannot solve L_2 in polynomial time. The contrapositive of this is that if we can solve L_2 in polynomial time, then* P = NP*. We have shown that we can solve L_2 in polynomial time if and only if* P = NP*. So, L_2 is* NP*-complete.*

4 Showing that Problems are NP-Complete: Examples

In this section, we will present several examples showing that given problems are NP-complete. Before doing so, we define these problems.

Definition 10 (Independent Set)**.** *We say $N \subseteq V$ forms an independent set of graph $G = (V, E)$ if no $v_1, v_2 \in N$ are adjacent in G. The Independent Set problem is given by the following.*

- *Instance: A graph $G = (V, E)$.*
- *Decide: Does there exist an independent set of G of size k?*

Definition 11 (Clique)**.** *We say $N \subseteq V$ forms a clique of graph $G = (V, E)$ if every $v_1, v_2 \in N$ are adjacent in G. The Clique problem is given by the following.*

- *Instance: A graph $G = (V, E)$.*
- *Decide: Does there exist a clique of G of size k?*

Definition 12 (Subset Sum)**.** *The Subset Sum problem is given by the following.*

- *Instance: A finite set $S \subseteq \mathbb{N}$.*
- *Decide: Does there exist $N \subseteq S$ with $\sum_{n \in N} n = k$?*

Definition 13 (Knapsack)**.** *The Knapsack problem is given by the following.*

- *Instance: A set of items $I \subseteq \mathbb{N}$, weights $W = \{w_1, \ldots, w_n\} \subseteq \mathbb{N}$, values $V = \{v_1, \ldots, v_n\} \subseteq \mathbb{N}$, capacity c, and desired profit v^*. Note that w_i and v_i correspond, respectively, to the weight and value of item i.*
- *Decide: Does there exist $N \subseteq I$ with $\sum_{w_i \in N} n \leq c$ such that $\sum_{n \in N} v_i \geq v^*$?*

Proposition 1. *Independent Set is* NP*-Complete.*

Proof. We reduce from 3-SAT. We wish to show that Independent Set is in NP, and is NP-hard.

To show that Independent Set is in NP, our certificate is a set N of vertices. In polynomial time, we can check if N is of size k and if N is indeed an independent set.

For hardness, since 3-SAT is NP-complete, we will show a reduction from 3-SAT to Independent Set. Let

$$\varphi = (\alpha_1 \vee \beta_1 \vee \gamma_1) \wedge \cdots \wedge (\alpha_k \vee \beta_k \vee \gamma_k)$$

be a 3CNF formula. We wish to construct a graph G such that G has an independent set of size k if and only if φ is satisfiable. Let $G = (V, E)$ with

$$V = \{\alpha_1, \beta_1, \gamma_1, \ldots, \alpha_k, \beta_k, \gamma_k\}$$

and

$$E = \{(u, v) : u \iff \neg v; u, v \in V\} \cup \{(\alpha_i, \beta_i), (\beta_i, \gamma_i), (\alpha_i, \gamma_i) : i \in \{1, \ldots, k\}\}.$$

That is, G has edge (u, v) if and only if either the negation of v is u, or u and v are in the same clause. We now show G has an independent set of size k if and only if φ is satisfiable.

- ($\Rightarrow$) Suppose $G = (V, E)$ has an independent set $N \subseteq V$ of size k. Then, since literals in clauses have edges between their corresponding vertices, N must contain a vertex corresponding to at most one literal from each clause. Since the independent set is of size k, N must contain at least one vertex corresponding to a literal from each clause. Thus, N contains exactly one vertex corresponding to a literal from each clause. Now, we will set truth assignments to the literals. For each $n_i \in N$, let the literal corresponding to n_i be TRUE. For vertices in the set $V \backslash E$, set the truth assignments arbitrarily. No contradictions arise since N does not contain vertices corresponding to both x and $\bar{x}$ for any literal x, since $(x, \bar{x})$ is an edge in G. We conclude that φ is satisfiable since N contains exactly one vertex corresponding to a literal from each clause of φ, and since we set those literals to TRUE, we have produced a satisfying assignment for φ.
- ($\Leftarrow$) Suppose φ is satisfiable. Then, at least one literal in each clause must be true. Choose N to be a set of k vertices in G such that the corresponding literals are all true and no two literals are in the same clause. The existence of N is guaranteed since φ is satisfiable. Since literals corresponding to the vertices of N are in different clauses, and are all TRUE N has no edges between its elements. Therefore, we have an independent set $N \subseteq V$ of size k.

Importantly, our reduction is polynomial-time. So, as we have shown a polynomial-time reduction from an NP-complete problem to Independent Set, which we have verified to be in NP, Independent Set is also NP-complete. ■

Proposition 2. *Clique is* NP*-Complete.*

Proof. We wish to show that Clique is in NP, and is NP-hard.

To show that Clique is in NP, our certificate is a set N of vertices. In polynomial time, we can check if N is of size k and if N is indeed a clique.

For hardness, since Independent Set is NP-complete, as was shown in the previous example, we will show a reduction from Independent Set to Clique. Let $G = (V, E)$ be a graph, and let $\langle G, k\rangle$ be an instance of Independent Set. We wish to construct an instance of Clique, $\langle G', k\rangle$, such that graph G' has a clique of size k if and only if G has an independent set of size k. Let $G' = (V, E')$ where

$$E' = \{(u, v) : (u, v) \notin E\} \subseteq V \times V.$$

That is, G' has edge (u, v) if and only if (u, v) is not an edge in G. We now show G' has a clique of size k if and only if G has an independent set of size k.

- ($\Rightarrow$) Suppose G' has a clique $N \subseteq V$ of size k. By definition, for all $n_i, n_j \in N$, (n_i, n_j) is an edge in G'. But by the way G' was constructed, (n_i, n_j) is not an edge in G. Therefore, in G, there exist no edges between vertices in N. So, $N \subseteq V$ is an independent set of size k for G.
- ($\Leftarrow$) Suppose G has an independent set $N \subseteq V$ of size k. By definition, for all $n_i, n_j \in N$, (n_i, n_j) is not an edge in G. But by the way G' was constructed, (n_i, n_j) is an edge in G'. Therefore, in G', all choices of two vertices in N share an edge. So, $N \subseteq V$ is an clique of size k for G'.

Importantly, our reduction is polynomial-time. So, as we have shown a polynomial-time reduction from an NP-complete problem to Clique, which we have verified to be in NP, Clique is also NP-complete. ■

Proposition 3. *Subset Sum is* NP*-complete.*

Proof. Exercise. Hint: Reduce from Independent Set. ■

Proposition 4. *Knapsack is* NP*-complete.*

Proof. We wish to show that Knapsack is in NP and is NP-hard.

We first show that Knapsack is in NP. The certificate consists of the subset of items chosen. The verifier sums up the weights and checks to see if the total weight is at most c. Then, the verifier sums the values and checks to see if the total value is at least v^*. This is done in polynomial time.

For hardness, since Subset Sum is NP-complete, we will show a reduction from Subset Sum to Knapsack[1]. Let $\mathcal{S} = \langle S, k\rangle$ be an instance of the subset sum problem. We wish to construct an instance $\mathcal{K} = \langle W, V, c, v^*\rangle$ of the knapsack problem such that there exists a subset of weights with total weight at most c, such that the corresponding profit is at least v^*, if and only if there is a subset of S with total sum k. Let

$$\mathcal{K} = \langle W, V, c, v^*\rangle = \langle S, S, k, k\rangle.$$

We will now show that $\mathcal{K}$ has a subset of weights with total weight at most k, such that the corresponding profit is at least k, if and only if there exists a subset of S with sum k.

- ($\Rightarrow$) Suppose $\mathcal{K}$ has a subset of weights with total weight at most k, such that the corresponding profit is at least k. Let this subset be denoted by S^*. By the construction of $\mathcal{K}$, $S^* \subseteq W = V = S$. We have that

  $$\sum_{s\in S^*} s \leq k, \quad \sum_{s\in S^*} s \geq k.$$

  So, $\sum_{s\in S^*} s = k$. We have shown that a subset of S, S^* has total sum k, so there exists a subset of S with total sum k.

- ($\Leftarrow$) Suppose there exists a subset of S with total sum k. Let this subset be denoted by S^*. By the construction of $\mathcal{K}$, $S^* \subseteq W = V = S$. Since we have that $\sum_{s\in S^*} s = k$, we immediately have

  $$\sum_{s\in S^*} s \leq k, \quad \sum_{s\in S^*} s \geq k,$$

  so there exists a subset of weights, S^* with total weight at most k, such that the corresponding profit is at least k.

Importantly, our reduction is polynomial-time. So, as we have shown a polynomial-time reduction from an NP-complete problem to Knapsack, which we have verified to be in NP, Knapsack is also NP-complete. ■

[1]Without loss of generality, we will omit the set of items I in Definition 13 from this reduction, as the set of items is solely to make indexing more explicit. The Decision can be equivalently changed to "Does there exist a subset of items such that the sum of the corresponding weights is at most c, while the sum of the corresponding values is at least v^*."

References

[1] Complexity zoo, https://complexityzoo.net/Complexity_Zoo, accessed July 24, 2024

[2] Agrawal, M., Kayal, N., Saxena, N.: Primes is in p. Annals of Mathematics **160**, 781–793 (2004). https://doi.org/10.4007/annals.2004.160.781

[3] Fortnow, L.: The Golden Ticket: P, NP and the search for the impossible. Princeton University Press, Princeton (2013), https://goldenticket.fortnow.com

[4] Sipser, M.: Introduction to the Theory of Computation. Cengage Learning (2012), https://books.google.com/books?id=H94JzgEACAAJ