3
The Hardness of Computational Problems

While we have seen many techniques to solve computational problems, there are yet other problems for which we have failed to give efficient algorithms, despite much effort. Interestingly, we have also failed to show that these problems cannot be solved efficiently. Given our state of limbo, how can we argue that these problems are difficult? Our discussions will lead us to a central question in Computer Science (and Mathematics and Logic) that formalizes the question: *What is the power of creativity? Can creativity be mechanized?*

3.1 Three Problems

Here are three different problems that we may want to solve, but we don't know efficient (poly-time) algorithms for:

1. Traveling Salesperson Problem. Given a collection of n cities and travel distances between them, is there a tour that visits all the cities and has length at most K? (Formally, given a finite metric space (V, d), does there exist an ordering of the vertices $v_1, v_2, \ldots, v_n$ such that $\sum_{i=1}^{n} d(v_i, v_{i+1}) \leq K$, where we imagine $v_{n+1} = v_1$.)

2. Partition. Given a collection of numbers, can it be partitioned into two sets, each of which sum to the same value? Formally, given numbers $a_1, a_2, \ldots, a_n$, can we partition $\{1, \ldots, n\}$ into two sets L and R such that $\sum_{i \in L} a_i = \sum_{i \in R} a_i$?

3. 3-Coloring. Can we color the vertices of a graph with three colors so that no two vertices connected by an edge have the same color. Formally, given a graph $G = (V, E)$, does there exist a map $f : V \to \{R, B, G\}$ such that for every edge $\{u, v\} \in E$ we have $f(u) \neq f(v)$?

3.2 . . . and Two Routes

If we want to show that a problem is difficult, we would normally do the following:

> *We would take a formal model of computation (say Turing machines) and then prove that the problem cannot be solved in this model of computation.*

If you are not familiar with Turing machines, just think of a regular computer, but with infinite memory.

Sadly, we know this kind of result only for restricted models of computation: say we can show that if we are only counting the number of comparisons, then sorting n numbers requires at least $\log n!$, i.e., $\Omega(n \log n)$ comparisions. Or if we were computing using bounded depth circuits. Or such restricted models. We also know this for artificially constructed problems (or we know there must exist such problems but we don't know explicit problems). Or we know some (problem, model) pairs where such a result is not interesting.

Which puts us in a bind: we feel these problems are difficult, but we cannot prove this fact. So what we show instead is not the impossibility but the *improbability* of efficient algorithms for these problems.

> *We show a large class of problems such that (a) people have tried to give algorithms for these and failed, and (b) all these problems are equivalent to each other: if we have a fast algorithm for one, then we have a fast algorithm for them all. Equivalently, if we can prove one of them does not have a fast algorithm, none of them do.*

The next few sections will introduce the concepts we need to define/formalize this concept.

3.3 Decision and Search Problems

A *decision problem* Q is a problem where the only possibly answers are Yes and No. Our three problems above are all decision problems. Every instance I of a decision problem Q can be classified as either a Yes-instance, or a No-instance, depending on whether the correcr answer on I is Yes or No.

For instance, the first graph above is a Yes-instance for the 3-Coloring problem, where as the second is a No-instance.

While discussing decision problems, let us introduce another fundamental problem, albeit one that requires some notation. Given a set of Boolean variables $x_1, x_2, \ldots, x_n$, each taking on a value of either T (true) or F (false), a *literal* is either a variable or its negation. A *clause* of arity k is the *disjunction* (i.e., OR) of k literals, e.g.,

$$(x_1 \vee \neg x_7 \vee x_{11})$$

is an arity-3 clause. A formula φ is in k-CNF (*conjunctive normal form*) if it is the *conjunction* of arity-k clauses. E.g., here is a 3-CNF formula. Finally, a formula φ is *satisfiable* if there is an assignment of T/F to the variables such that φ evaluates to true. Armed with this terminology, we can define a fourth decision problem, which will be one of the central problems in our story.

4. 3-SATISFIABILITY. Given a 3-CNF formula φ, is it satisfiable?

Decision problems should be contrasted with *search problems*, which finds the object in question: e.g., to find the tour of length at most K, or the partition with equal sums, or the 3-coloring of the graph vertices. If you can solve a search problem, you can also solve the decision problem, but the converse is not always clear.

Talk about theorems here?

Exercise 3.1. Show how to solve the search version of 3-SATISFIABILITY using at most n calls to a black-box algorithm for its decision version.

3.4 *The Class P of Poly-time Decision Problems*

3.5 *The Class NP*

3.6 *NP-completeness*

A problem is *NP-complete* if, loosely speaking, it is a "hardest problem in NP". Formally, the definition is:

Definition 3.2. A problem Q is *NP-complete* if

1. $Q \in NP$, and
2. If $Q \in P$ then every problem in NP belongs to P.

Given a problem Q, the first part of the definition is easy enough to verify, but the second part poses some challenges: how can one show such a property? The key to this is a brilliant theorem of Steven Cook and Leonid Levin:

Some words about the Cook-Levin theorem.

Theorem 3.3. *The* 3-SATISFIABILITY *problem is NP-complete.*

Theorem 3.3 makes the problem of showing NP-completeness much easier—we only need to verify the following simpler-to-argue condition instead of condition (2):

2′. If $Q \in P$ then 3-SATISFIABILITY $\in P$.

In fact, we could consider an even simpler-to-argue condition instead of condition (2′):

2″. There exists some NP-complete problem Q' such that $Q \in P \implies Q' \in P$.

Indeed, since $Q' \in P$ implies every problem in NP belongs to P (by the NP-completeness of Q'), we get back condition-2 from $2''$. And condition-(2") makes our life progressively easier: as more problems get proven to be NP-complete, it becomes easier to prove the NP-completeness of yet others—there are more candidates to use for Q'.

In 1972, a landmark paper of Richard Karp showed that 29 decision problems, including the three above, satisfied both properties (1) and (2"), and hence were NP-complete. Since these problems were now all "hardest problems in NP" as well, this started an avalanche that continues to this day: tens of thousands of problems have since been shown to be NP-complete, to be the "hardest problems in NP".

3.7 *A User's Guide to NP-completeness*

1. First show that the problem Q belongs to NP. This means that if we can solve any NP-complete problem (say SAT), we can solve Q.

2. Then take some NP-complete problem Q', and show that if you can solve Q you can solve Q' as well.