CIS 5110
Introduction to the Theory of Computation
Motivations, Problems

Jean Gallier

October 17, 2024

Chapter 1

Motivations, Problems

What is the *Theory of Computation*?

Roughly three overlapping topics:

(1) Languages and Automata.

(2) Computability, Decidability, Undecidability.

(3) Complexity.

It started with (2) at the end of the 1800's, mostly by the impetus of Hilbert.

The field takes off in the early 1930's triggered by the incompleteness results of Gödel (1931), and additional seminal work by Church, Kleene, Rosser, Turing, and Post.

1. Languages and Automata.

What is a language?

How do we define a language?

How we recognize words in a language (*parsing*).

Machinery: automata and grammars.

DFA's, NFA's, context-free grammars (Chomsky).

Mature theory with practical applications: construction of the front-end of *compilers*.

Lexical analyzers and lexical analyzer generators.

Parser and parser generators (LALR(1) parsers).

More advanced and still very active: natural language recognization, *computational linguistics*.

2. Computability Decidability, Undecidability

Many of the problems arose from *logic*.

(1) What is a *computable function*?

(2) Can we decide whether a function (or a program) *terminates* for a given input.

(3) What is a *proof*?

(4) Can we decide whether a proposition is provable or not? What does this mean?

(5) Can theorem-proving or proof-checking be automated? What does this mean?

(6) More generally, when is a problem *decidable*, and what does this mean.

Unfortunately, *in general the answer is almost always no!*

Here is an example showing that deciding whether a function is defined for *all* inputs is tricky.

The "$3n + 1$ problem" proposed by Collatz around 1937 is the following:

Given any positive integer $n \geq 1$, construct the sequence $c_i(n)$ as follows starting with $i = 1$:

$$\begin{aligned} c_1(n) &= n \\ c_{i+1}(n) &= \begin{cases} c_i(n)/2 & \text{if } c_i(n) \text{ is even} \\ 3c_i(n) + 1 & \text{if } c_i(n) \text{ is odd.} \end{cases} \end{aligned}$$

Observe that for $n = 1$, we get the infinite periodic sequence

$$1 \Longrightarrow 4 \Longrightarrow 2 \Longrightarrow 1 \Longrightarrow 4 \Longrightarrow 2 \Longrightarrow 1 \Longrightarrow \cdots,$$

so we may assume that we stop the first time that the sequence $c_i(n)$ reaches the value 1 (if it actually does).

Such an index i is called the *stopping time* of the sequence. For our previous example, $i = 4$.

Starting with $n = 3$, we get the sequence

$$3 \Longrightarrow 10 \Longrightarrow 5 \Longrightarrow 16 \Longrightarrow 8 \Longrightarrow 4 \Longrightarrow 2 \Longrightarrow 1.$$

Starting with $n = 5$, we get the sequence

$$5 \Longrightarrow 16 \Longrightarrow 8 \Longrightarrow 4 \Longrightarrow 2 \Longrightarrow 1.$$

Starting with $n = 6$, we get the sequence

$$6 \Longrightarrow 3 \Longrightarrow 10 \Longrightarrow 5 \Longrightarrow 16 \Longrightarrow 8 \Longrightarrow 4 \Longrightarrow 2 \Longrightarrow 1.$$

Starting with $n = 7$, we get the sequence

$$7 \Longrightarrow 22 \Longrightarrow 11 \Longrightarrow 34 \Longrightarrow 17 \Longrightarrow 52 \Longrightarrow 26 \Longrightarrow 13 \Longrightarrow$$
$$40 \Longrightarrow 20 \Longrightarrow 10 \Longrightarrow 5 \Longrightarrow 16 \Longrightarrow 8 \Longrightarrow 4 \Longrightarrow 2 \Longrightarrow 1.$$

One might be surprised to find that for $n = 27$, it takes 111 steps to reach 1, and for $n = 97$, it takes 118 steps.

I computed the stopping times for n up to 10^7 and found that the largest stopping time, 686 (685 steps), is obtained for $n = 8400511$.

The terms of this sequence reach values over 1.5×10^{11}. The graph of the sequence $c(8400511)$ is shown in Figure 1.1.

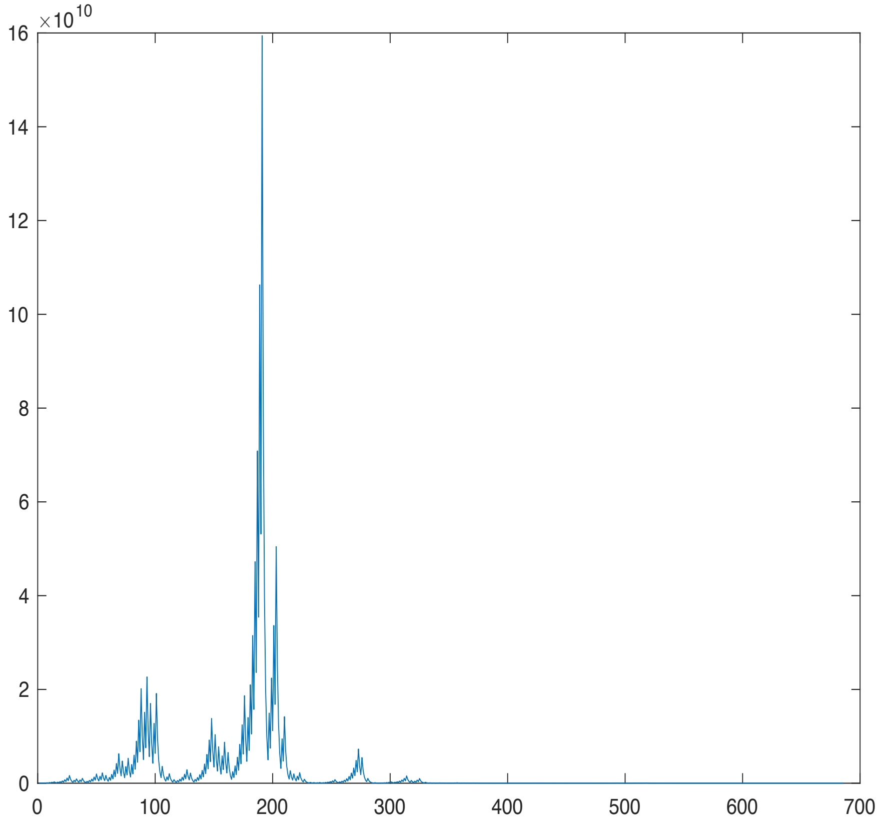

Figure 1.1: Graph of the sequence for $n = 8400511$.

We can define the (partial) function C (with positive integer inputs) defined by

$$C(n) = \text{the smallest } i \text{ such that } c_i(n) = 1 \text{ if it exists.}$$

The graph of the function C for $1 \leq n \leq 10^7$ is shown in Figure 1.2.

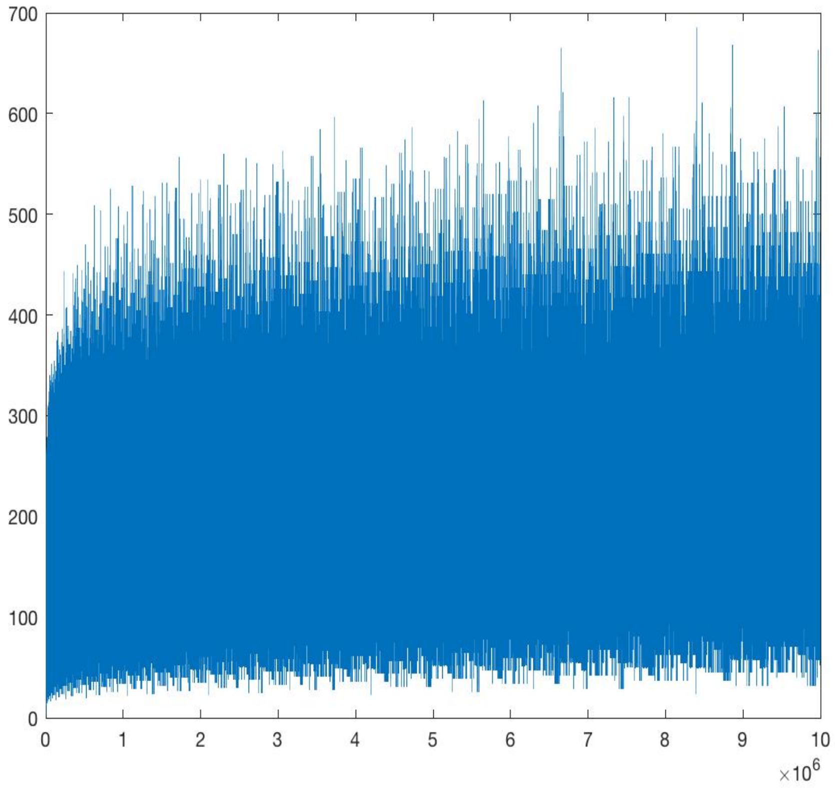

Figure 1.2: Graph of the function C for $1 \leq n \leq 10^7$.

Conjecture (Collatz):

For any starting integer value $n \geq 1$, the sequence $(c_i(n))$ always reaches 1.

So far, the conjecture remains open. It has been checked by compuer for all integers less than or equal to 87×2^{60}.

Another deceptively hard problem:

Deciding wheher a polynomial *with integer coefficients* has some *integer solution* which gives it the value zero.

The equation

$$x^2 + y^2 - z^2 = 0$$

has the solution $x = 3$, $y = 4$, $z = 5$, since $3^2 + 4^2 = 9 + 16 = 25 = 5^2$.

The equation

$$x^3 + y^3 + z^3 - 29 = 0$$

has the solution $x = 3$, $y = 1$, $z = 1$.

What about the equation

$$x^3 + y^3 + z^3 - 30 = 0?$$

Amazingly, the only known integer solution is

$$(x, y, z) = (-283059965, -2218888517, 2220422932),$$

discovered in 1999 by E. Pine, K. Yarbrough, W. Tarrant, and M. Beck, following an approach suggested by N. Elkies.

And what about solutions of the equation

$$x^3 + y^3 + z^3 - 33 = 0?$$

Until 2019 it was still an open problem.

Andrew Booker found the following amazing solution:

$$\begin{aligned}&(8,866,128,975,287,528)^3\\&+(-8,778,405,442,862,239)^3\\&+(-2,736,111,468,807,040)^3=33.\end{aligned}$$

In 1900, at the International Congress of Mathematicians held in Paris, the famous mathematician David Hilbert presented a list of ten open mathematical problems.

Soon after, Hilbert published a list of 23 problems. The tenth problem is this:

Hilbert's tenth problem (H10)

Find an algorithm that solves the following problem:

Given as input a polynomial $P \in \mathbb{Z}[x_1, \ldots, x_n]$ with integer coefficients, return YES or NO, according to whether there exist integers $a_1, \ldots, a_n \in \mathbb{Z}$ so that $P(a_1, \ldots, a_n) = 0$; that is, the Diophantine equation $P(x_1, \ldots, x_n) = 0$ has a solution.

It is important to note that at the time Hilbert proposed his tenth problem, *a rigorous mathematical definition of the notion of algorithm did not exist.*

In fact, the machinery needed to even define the notion of algorithm did not exist.

It is only around 1930 that precise definitions of the notion of computability due to Turing, Church, and Kleene, were formulated, and soon after shown to be all equivalent.

In 1970, the following somewhat surprising resolution of Hilbert's tenth problem was reached:

Theorem (Davis-Putnam-Robinson-Matiyasevich)

Hilbert's tenth problem is undecidable; that is, there is no algorithm for solving Hilbert's tenth problem.

Amazingly, *all known models of computation (to define the notion of computable function) are equivalent*.

They all define the same class, the *partial computable functions* in the sense of Gödel and Kleene.

We will discuss the following computation models and sketch some of their equivalences:

(1) RAM programs.

(2) Turing Machines.

(3) The class of *partial computable functions* in the sense of Gödel and Kleene.

(4) The functions definable in the *λ-calculus of Church*. Most functional programming languages (*e.g.*, OCaml, see CIS 120) are based on the λ-calculus

(5) Diophantine definability (integer solutions of polynomials with integer coefficients).

We will also define what it means for a problem to be *decidable* (or *undecidable*).

We will see that essentially all nontrivial problems are undecidable; see Rice's theorem.

3. Complexity

Even if a problem is *theoretically solvable* (there is an algorithm for it), *in practice*, all known algorithms may take *too much time or too much space* (say exponential).

In the 1970's, Cook, Karp, and Levin argued that a practical algorithm should run in *polynomial time* (in the length of the input).

This yields the class $\mathcal{P}$.

It was observed that there are many problems for which *no known polynomial-time algorithm exists*, but if we allow *guessing a solution* (for free) and if *we can check the solution in polynomial time*, then these problems are solvable.

This yields the class $\mathcal{NP}$.

Obviously $\mathcal{P} \subseteq \mathcal{NP}$ but

nobody knows whether $\mathcal{P} = \mathcal{NP}$!

This is **The** big question of theoretical computer science. A price of a million dollars is offered for its solution!

Two examples of problems in $\mathcal{NP}$.

1. 0-1-Integer Programming

Consider the 5×6 matrix A and the vector b shown below:

$$A = \begin{pmatrix} 1 & -2 & 1 & 3 & -1 & 4 \\ 2 & 2 & -1 & 0 & 1 & -1 \\ -1 & 1 & 2 & 3 & -2 & 3 \\ 3 & 1 & -1 & 2 & -1 & 4 \\ 0 & 1 & -1 & 1 & 1 & 2 \end{pmatrix}, \qquad b = \begin{pmatrix} 9 \\ 0 \\ 7 \\ 8 \\ 2 \end{pmatrix}.$$

Problem: Is there a solution $x = (x_1, x_2, x_3, x_4, x_5, x_6)$ of the linear system

$$\begin{pmatrix} 1 & -2 & 1 & 3 & -1 & 4 \\ 2 & 2 & -1 & 0 & 1 & -1 \\ -1 & 1 & 2 & 3 & -2 & 3 \\ 3 & 1 & -1 & 2 & -1 & 4 \\ 0 & 1 & -1 & 1 & 1 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{pmatrix} = \begin{pmatrix} 9 \\ 0 \\ 7 \\ 8 \\ 2 \end{pmatrix}$$

with $x_i \in \{0, 1\}$?

Yes!

$x = (1, 0, 1, 1, 0, 1)$.

If the system has n variables, it takes 2^n guesses for x, but checking that x works is cheap (polynomial time).

2. Finding a Hamiltonian Cycle

Given a (directed) graph, G, a *Hamiltonian cycle* is a cycle that passes through all the nodes exactly once (note, some edges may not be traversed at all).

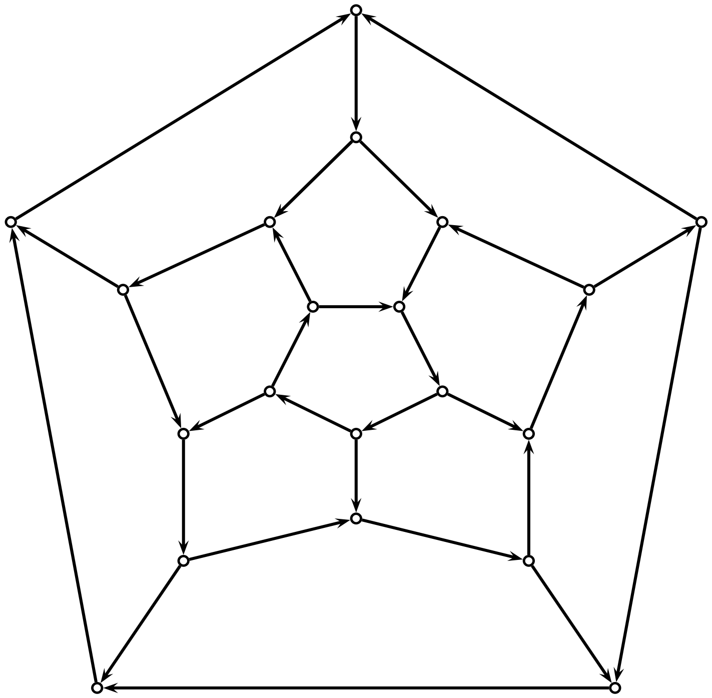

Figure 1.3: A tour "around the world."

Finding a Hamiltonian cycle in this graph does not appear to be so easy!

A solution is shown in Figure 1.4 below.

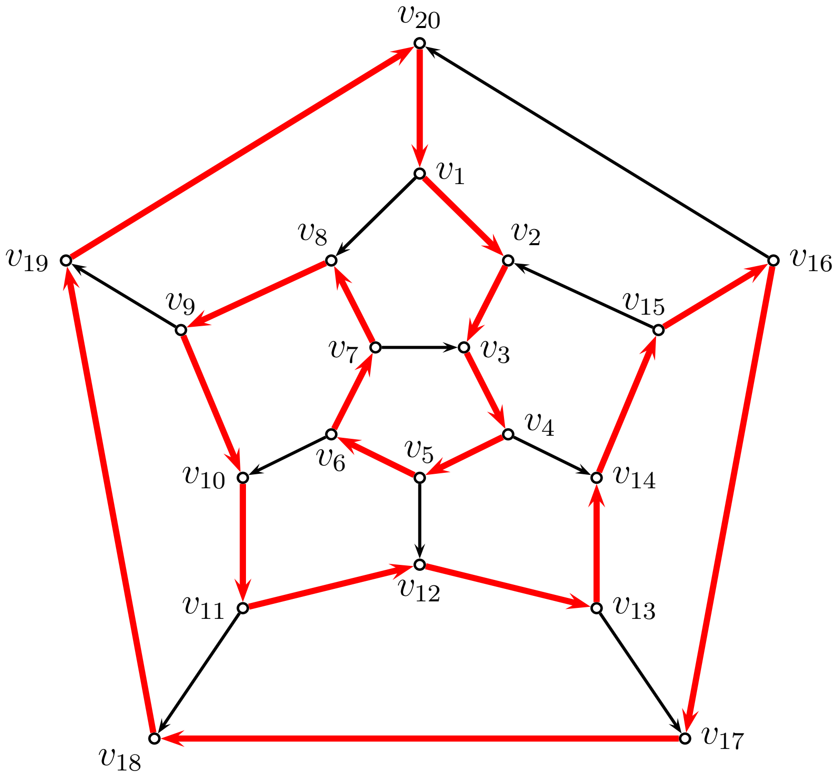

Figure 1.4: A Hamiltonian cycle in D.

Again, in general, there is an exponential number of guesses, but given a cycle, by following it we check that it works in polynomial time.

Amazingly, these problems are equivalent, in the sense that it is possible to translate one into the other *in polynomial time*!

These problems are *$\mathcal{NP}$-complete*.

This means that *every* problem in $\mathcal{NP}$ can be translated into them in polynomial time.

We will investigate a number of $\mathcal{NP}$-complete problems.

One of the most important ones is the *satisfiability problem (SAT)* for propositions in CNF.

We will also look at other complexity classes besides $\mathcal{P}$ and $\mathcal{NP}$, namely co$-\mathcal{NP}$ and $\mathcal{PS}$ (Pspace).

They also have complete problems (TAUT and QBF).

A problem of particular interest is *primality testing*.

Is $3\,215\,031\,751$ prime?

No, because

$$3\,215\,031\,751 = 151 \cdot 751 \cdot 28351.$$

If we can guess a factorization, then checking that it works is cheap.

But if our number n is prime, what do we guess and then check in polynomial time to show that n is indeed prime.

Is $474\,397\,531$ prime?

Yes!

Around 1975, V. Pratt found a method (based on the Lucas test). This shows that primality testing *is* in $\mathcal{NP}$.

Remarkably, in 2002, it was shown by Agrawal, Kayal and Saxena that primality testing *is actually in* $\mathcal{P}$.

This is a very hard theorem, and it is not really practical.

Randomized methods are much cheaper.

Unfortunately we probably won't have time to discuss primality testing, but you should be aware of it.