CSC 373: Algorithm Design and Analysis
Lecture 13

Allan Borodin

February 6, 2013

Some materials are from Stephen Cook's IIT Mumbai (Bombay) slides.

Announcements and Outline

Announcements

- I hope Term Test 1 went well for everyone.
- No lecture this Friday
- I have posted the first two questions for Problem Set 2.
  - Both questions relate to flow networks and they can be done now.
  - Please start on the problem set and make sure the questions make sense to you.

Today's outline

- We begin complexity theory and NP completeness
- Warning: today's lecture starts with many definitions and conventions and hence may seem boring. I apologize in advance but we need to set the stage.
- I am going to use some slides from a recent talk by Stephen Cook to hopefully better motivate the topic.

What is computable?

- One of the greatest ideas in computer science is that we basically all agree on what it means for a (discrete) computational problem to be computable.
- Namely, we equate the intuitive concept of “computable” with the mathematically precisely defined concept of “Turing computable”
  - Turing computable = computable by a precise model called a Turing machine, named for Alan Turing.
- This is the so-called Church-Turing thesis (from 1936) as both Church and Turing independently established formal definitions for the concept of a computable function.
  - These formulations turned out to be equivalent.
- Although the Church-Turing thesis can never be proved, it is almost universally believed. **Why and why?**
- Using diagonalization (the same proof technique used to show that there are more reals than rationals), Turing showed that there are (explicitly defined) functions that are not (Turing) computable.

Turing Machines

- In 1936, Alan Turing ("The Father of Computer Science") published
  "On Computable Numbers with an Application to the Entscheidnungsproblem"

- He introduced a mathematical model of a computer (before computers had been invented), which we call a "Turing Machine".
- All present day computers can be simulated by Turing machines. The hardware gets faster, but they are not fundamentally different.
- There may be fundamental limitations on how fast and large computers can be.
- Say the limit is one bit per electron, with the computer clock limited by the speed of light between atoms.
- With such limits, even with the whole earth as a computer, a brute force search through all 300-bit strings would take universe lifetimes.

What does it mean to be efficiently computable?

- Let n denote a measure of the input/output "size" of the problem. One can also use diagonalization to show that

For any time bound $T(n)$, there are computable functions not computable within time T(n) for problems of size n.

- Following Cobham and Edmonds (circa 1965), we will equate the intuitive concept of "efficiently computable" with computable in polynomial time (i.e. time bounded by a polynomial function of the encoded length of the inputs and outputs).
  - This has sometimes been called the Extended Church-Turing Thesis.
  - This hypothesis is not literally believed in contrast to the Church-Turing Thesis. **Why?**
  - But it is an abstraction that has led to great progress in computing.
- **Informal claim:** Any function (polynomial time) computable is (polynomial time) computable by a Turing machine.
- For the time being we will not define a precise computational model such as the Turing machine model.

Encoding of inputs and outputs

- We are always assuming that inputs and outputs are encoded as strings over some finite alphabet S with at least 2 symbols.
  - S^n denotes the set of all finite strings of length n over the alphabet S.
  - The empty string is the only string of length 0.
  - $S^* = \bigcup_{n \geq 0} S^n$ denotes the set of all finite strings over the alphabet S
- We can use as many symbols as we want but 2 suffices for our purpose. (**Note:** finitely many symbols on a keyboard.)
- We can also encode in unary, but that causes an exponential blowup in representation.
- We can encode a set of inputs or outputs $w_1, \ldots, w_n$ by having a special symbol (say $\#$) to separate the inputs but again this can all be encoded back into 2 symbols.
- If we need to distinguish components of an input, we can denote such an encoding of many inputs (or outputs) as $\langle w_1, \ldots, w_n \rangle$.
  - We will usually not need to be so careful about the distinction between an object G and its encoding $\langle G \rangle$.

What are the objects of study?

- Since it suffices to encode all inputs as finite strings over the alphabet $S = \{0, 1\} = \{\text{false}, \text{true}\}$, we often refer to the complexity issues of such computations as Boolean complexity theory.
  - This is in contrast to, say, arithmetic complexity theory, or the theory of real valued computation.
- Our general objects of study (for Boolean complexity theory) are the computation of:
  1. Functions $f : S^* \to S^*$
  2. Search problems
     - ★ Let $R(x, y) \subseteq S^* \times S^*$ be a relation.
     - ★ Given x, need to find a y satisfying $R(x, y)$ or say that no such y exists.
  3. Optimization problems
     - ★ Let $R(x, y) \subseteq S^* \times S^*$ be a relation, and $c : S^* \times S^* \to \Re$ be an objective function.
     - ★ Given x, we need to find a y satsifying $R(x, y)$ so as to minimize (or maximize) $c(x, y)$ or say that no such y exists.
  4. Decision problems
     - ★ Let $R(x, y) \subseteq S^* \times S^*$ be a relation.
     - ★ Given x, determine if there exists a y satisfying $R(x, y)$.

Polynomial time computable functions

- We say that a function $f : S^* \rightarrow S^*$ is computable in time $T(\cdot)$ if:
  1. There is an algorithm (to be precise a Turing machine or an idealized RAM program with an appropriate instruction set)
  2. For all inputs $w \in S^*$, the algorithm halts using at most $T(n)$ "time" where $n = |w| + |f(w)|$.
- We can define polynomial time search problems, optimization problems and decision problems similarly.
- We will generally not be dealing with functions where $|f(w)| > |w|$. So n will be the length of the input.
- In particular, for decision problems, n will always refer to the length of the encoded input.
- We let P denote the class of decision problems that are solvable in polynomial time. We sometimes abuse notation and also use P to denote any problem computable in polynomial time.

Computational Complexity Theory

- Classifies problems according to their computational difficulty.
- The class P (Polynomial Time) [Cobham, Edmonds, 1965]
- P consists of all problems that have an efficient (e.g. $n, n^2 \ldots$) algorithm. (n is the input length)

Examples in P

- Addition, Multiplication, Square Roots
- Shortest Path (Google Maps)
- Network flows (Internet Routing)
- Pattern matching (Spell Checking, Text Processing)
- Fast Fourier Transform (Audio and Image Processing, Oil Exploration)
- Recognizing Prime Numbers [Agrawal-Kayal-Saxena 2002]
- . . .

Polynomial time continued

- Until we have a precise computation model, we cannot define time.
- But we can proceed informally by assuming that we all have a good intuition as to our concept of computational time.
- One of the nice properties of a Turing machine model is that the concept of a computational step and hence time is well defined.
- **Warning:** If say a RAM has a multiplication operation, then by repeatedly squaring we can compute 2^{2^n} in n operations.
  - We argue that we cannot assume such an operation only costs 1 time unit since the operands will have 2^n bits and hence we might be encoding an exponential time computation within such operations.
- In particular, it can be shown that if we do not account properly for the cost of RAM operations, then we can factor integers (and thus be able to break encryption schemes such as RSA) in polynomial time but such a computational algorithm would not be considered realistic.
- One way to deal with such RAM models is to say that any operation on operands of length m requires time m.

Classical vs quantum computation

- Quantum computation takes advantage of principles of quantum mechanics (such as “entangled states”) which allow a quantum state involving n ‘qubits’ to be a “superposition” of up to 2^n possible bit strings.
- In 1994 Peter Shor proved that a quantum computer can factor numbers into primes in polynomial time.
- So if large quantum computers can be built, they could crack the RSA encryption scheme.
- **The Catch:** Despite substantial effort, physicists have so far been unable to build an actual quantum computer large enough to process more than a dozen bits of information.
- **Caveat:** Quantum cryptography provides a promising approach to secure communication in which security (rather than complexity) depends on principles of quantum mechanics.

Note

Quantum computation does NOT change the Church-Turing thesis, that is, what is computable. But it does seem to change what is computable in polynomial time.