Great Ideas in Computing

University of Toronto CSC196
Fall 2021

Week 8: November 1-5 (2021)

Slides for Friday, November 5 class

Announcements:

- To keep the dates consistent, we end week 8 with a new topic: complexity theory and the P vs NP question. After reading week, this topic will be resumed in week 9 on Monday, November 15.
- I will soon be posting the remaining questions for assignment 3. Please be sure that your assignments are properly uploaded onto Markus.

This weeks agenda

- Complexity theory and the P vs NP question.

Complexity theory and the road ahead

- As stated, our next topic is complexity theory.
- In particular, we will be discussing the concept of NP and NP-complete decision problems.
- Complexity theory will then lead us to the topic of complexity based cryptography
- Complexity theory will also lead us to the topic of algorithmic game and algorithmic aspects of mechanism design.

What is complexity theory?

The ideal objective of complexity theory is to be able to understand the more of less precise computational complexity of any computational problem in terms of different models of computation and different measures of complexity. We also study how various models and measures relate to each other.

Closely related (or some would say included in complexity theory) are the studies of communication complexity, and proof complexity.

We can study models such as Turing machines (and related classical computational models), quantum computers, Boolean and arithmetic circuits, parallel computers, models for distributed computing, limited models of computation, etc.

In computational complexity, the main complexity measures we tend to study are time, space, and parallel time and analogues of the these measures for circuits.

Our focus and things we won't be discussing

Our focus will be on the classical Turing machine model (as a repesentative of other classical models) and sequential time complexity. Moreover, we will focus our attention on the (literally) million dollar question as to whether or not $P = NP$. I say literally (and not figuratively) since there is a \$1 million dollar reward for solving this problem offered by the Clay Institute.

But there are many other important questions (but arguably not as important) as the P vs NP question. Here are some other important topics where major open problems remain.

- How much can randomization help? For example, can we do more in randomized polynomial time than in deterministic polynomial time? Can we do more in randomized space $O(n)$ than in determninistic $O(n)$ space?
- Can every problem computable in polynomial time say $T(n)$ be computed using only $O(\log n)$ or polylog($T(n)$ space or parallel time? Note: there is a known polynomial relation between sequential space and parallel time.

More on our focus and things we won't be discussing

- Time-space tradeoffs and other tradeoffs (e,g., probability of error vs number of random bits).
- Fine-grained complexity which aims to distinguish between what can be done say in Time $Tn) = O(n)$ vs $O(n^2)$ vs $O(n^3)$

Note: We will follow most of the work regarding the P vs NP question and consider worst case complexity; that is, computations that halt within time $T(|w|)$ for every input string w . We also will only care about asymptotic bounds and usually skip using the "big Oh" notation and note that on Turing machines there is a sense (albeit unreasonable) where the implicit constants in the big-Oh notation do not matter for Turing machine computations (by increasing the size of the tape alphabet).

And now onto sequential time and the P vs NP question.

The extended Church-Turing thesis

We recall the Church-Turing thesis, namely that every computable function f is Turing computable. More precisely, there is a Turing machine $\mathcal{M}$ such that on every input x, $\mathcal{M}$ halts and outputs $f(x)$. That is, the Church-Turing thesis equates the informal concept of "computable" with the mathematically precise concept of "Turing machine computable".

The extended Church-Turing thesis equates the informal concept of "efficiently computable" with the mathematical precise concept of "computable by a Turing machine in polynomial time". More precisely, the extended Church-Turing thesis states that a function is efficiently computable if there is a Turing machine $\mathcal{M}$ and a polynomial $p(n)$ such that on every input x, $\mathcal{M}$ halts in at most $p(|x|)$ steps and outputs $f(x)$.

Here we are assuming $x \in \Sigma^*$ for some finite alphabet Σ and $|x|$ represents the length of the string x. In what follows, I will use n to be the length of an input; $n = |x|$.

The extended Church-Turing thesis continued

Do we believe the extended Church-Turing thesis?

Why should we accept the extended Church-Turing thesis?

- We can simulate in polynomial time a random access von Neumann random access machine (if we say, as we should, that the time for basic operations on ℓ bit operands is $O(\ell)$. This is a robust definition.
- That is, if a function f is computable in time $p_1(n)$ on a von Neumann random access machine, then there is some polynomial $p_2(n)$ such that f is computable in polynomial time $p(n) = p_2(p_1(n))$ on a Turing machine. For example, if $p_1(n) = n^3$ and $p_2(n) = n^2$ then $p(n) = n^6$.
- What are some typical asymptotic time bounds for natural problems? I am omitting the big O.

The extended Church-Turing thesis continued

Do we believe the extended Church-Turing thesis?

Why should we accept the extended Church-Turing thesis?

- We can simulate in polynomial time a random access von Neumann random access machine (if we say, as we should, that the time for basic operations on ℓ bit operands is $O(\ell)$. This is a robust definition.
- That is, if a function f is computable in time $p_1(n)$ on a von Neumann random access machine, then there is some polynomial $p_2(n)$ such that f is computable in polynomial time $p(n) = p_2(p_1(n))$ on a Turing machine. For example, if $p_1(n) = n^3$ and $p_2(n) = n^2$ then $p(n) = n^6$.
- What are some typical asymptotic time bounds for natural problems? I am omitting the big O.
  $\log n, \sqrt{n}, n, n \log n, n^2, n^3, \ldots, n^{\log n}, 2^n, n!$
- For problems involving say enormous graphs, we need sublinear time; for other problems we may need linear or near linear times. But as an abstraction, we are saying that polynomial time is the most time we will consider for "efficient computation".

Why should we be less accepting of the extended Church-Turing thesis?

While we are very confidant about the Church-Turing thesis (for defining "computable"), there are various reasons to be a little more skeptical about the extended Church-Turing thesis.

- An algorithm running in a polynomial time bound like n^{100} is not an efficient algorithm. For problems involving say enormous graphs, we need sublinear time; for other problems we may need linear or near linear times. But as an abstraction, we are saying that polynomial time is the most time we will consider for "efficient computation".
- An algorithm running in an exponential time bound like $(1 + \frac{1}{1000})^n$ is an efficient algorithm for reasonably large input lengths.
- While we can simulate classical computers (i.e. von Neumann machines) in polynomial time, we do not know how to simulate non classical computers (e.g., quantum computers) in polynomial time. Factoring is an example of a problem that can be computed in polynomial time by a quantum computer whereas we do not believe factoring is polynomial time computable on a classical computer.

So should we accept the extended Church-Turing thesis?

We can accept the extended Church-Turing thesis, arguing as follows:

- Polynomial time computable functions usually have reasonably small asymptotic polynomial time bounds; that is, $n, n \log n, n^2, n^3$. There are some exceptions (like n^6, but generally speaking we don't encounter asymptotic bounds bigger than n^3.
- The robustness of polynomial time (in terms of being closed under composition and hence not sensitive to the precise model of computing and definition of a time step. This enables us to define our concepts in terms of Turing machines (once we restrict outselves to classical computer models). Linear functions are also closed under composition but linear time computation is very model dependent.
- While non-classical models may contradict the thesis, so far we do not have non-classical computers (e.g., quantum computers that go beyond a small number of quantum bits) that are practical in any commercial sense.

But what if quantum computers become practical?

Lets assume the quantum computers or any other non-classical computers become practical. We are about to discuss the NP vs P issue and the $NP \neq P$ conjecture, the central question in complexity theory.

This conjecture is formulated with respect to the extended Turing thesis. That is, we are accepting the definition that "efficiently computable" means polynomial time computable by a Turing machine. Will everything about this question and conjecture become useless if we someday have available more powerful non-classical computers?

But what if quantum computers become practical?

Lets assume the quantum computers or any other non-classical computers become practical. We are about to discuss the NP vs P issue and the $NP \neq P$ conjecture, the central question in complexity theory.

This conjecture is formulated with respect to the extended Turing thesis. That is, we are accepting the definition that "efficiently computable" means polynomial time computable by a Turing machine. Will everything about this question and conjecture become useless if we someday have available more powerful non-classical computers?

No, the theory we will be developing can be reformulated in terms of a new computational model. We will have new functions (like factoring integers) which will now become efficiently computable (assuming they were not efficiently computable classically). But still there will be an analogous complexity theory based on the (for now hypothetical) new computational model.

Polynomial time computable decision problems

We will now restrict attention to decision problems; that is $f : \Sigma^* \to \{NO, YES\}$ As used before, Σ is a finite alphabet and Σ^* is the set of all strings over Σ. We can also identify NO,YES with say $\{0,1\}$ Equivalently, we are considering languages $L \subseteq \Sigma^*$.

The class of languages (decision problems) P is defined as the set of languages L that are decideable in polynomial time on a Turing machine (i.e. languages that are "efficiently decideable"). Formally:

Definition: $L \in P$ if there exists a TM $\mathcal{M}$ and polynomial $p(n)$ such that for every $w \in \Sigma^*, w \in L$ if and only if $\mathcal{M}(w)$ halts in at most $p(|w|)$ steps and accepts w. If $w \notin L, \mathcal{M}$ rejects w.

In what follows, I will assume we have some agreed upon way that we represent graphs $G = (V, E)$ as strings over some finite alphabet Σ. Without refering to the representation, let $L_{connected} = \{G = (V, E) | G$ is connected$\}$

It is not difficult to show that $L_{connected}$ is in P. (For example, we can use breadth first search.)

A language "probably not" in the class P

Consider the following language:
$L_{HC} = \{G = (V, E) | G$ has a simple cycle including all nodes in $V\}$.
It is strongly believed (*but not proven*) that L_{HC} is not polynomial time computable.

A simple cycle containing all the nodes in the graph is called a *Hamiltonian cycle* (HC). (The "well-known" *traveling salesman problem* (or traveling salesperson to be gender neutral) (TSP) is to find an HC of least cost in an edge weighted graph. Have you heard of this problem?.)

A language "probably not" in the class P

Consider the following language:
$L_{HC} = \{G = (V, E) | G$ has a simple cycle including all nodes in $V\}$.
It is strongly believed (*but not proven*) that L_{HC} is not polynomial time computable.

A simple cycle containing all the nodes in the graph is called a *Hamiltonian cycle* (HC). (The "well-known" *traveling salesman problem* (or traveling salesperson to be gender neutral) (TSP) is to find an HC of least cost in an edge weighted graph. Have you heard of this problem?.)

But suppose I know that a given graph G has Hamiltonian cycle.
How can I convince you that G has such a cycle?

A language "probably not" in the class P

Consider the following language:
$L_{HC} = \{G = (V, E) | G$ has a simple cycle including all nodes in $V\}$.
It is strongly believed (*but not proven*) that L_{HC} is not polynomial time computable.

A simple cycle containing all the nodes in the graph is called a *Hamiltonian cycle* (HC). (The "well-known" *traveling salesman problem* (or traveling salesperson to be gender neutral) (TSP) is to find an HC of least cost in an edge weighted graph. Have you heard of this problem?.)

But suppose I know that a given graph G has Hamiltonian cycle.
How can I convince you that G has such a cycle?

I can simply show you a Hamiltonian cycle C and you can easily and efficiently *verify* that C is indeed a HC. That is, I can prove to you that G has a HC.

A language "probably not" in the class P

Consider the following language:
$L_{HC} = \{G = (V, E) | G$ has a simple cycle including all nodes in $V\}$.
It is strongly believed (*but not proven*) that L_{HC} is not polynomial time computable.

A simple cycle containing all the nodes in the graph is called a *Hamiltonian cycle* (HC). (The "well-known" *traveling salesman problem* (or traveling salesperson to be gender neutral) (TSP) is to find an HC of least cost in an edge weighted graph. Have you heard of this problem?.)

But suppose I know that a given graph G has Hamiltonian cycle.
How can I convince you that G has such a cycle?

I can simply show you a Hamiltonian cycle C and you can easily and efficiently *verify* that C is indeed a HC. That is, I can prove to you that G has a HC.

But can I prove to you the G does *not* have a HC?

A language "probably not" in the class P

Consider the following language:
$L_{HC} = \{G = (V, E) | G$ has a simple cycle including all nodes in $V\}$.
It is strongly believed (*but not proven*) that L_{HC} is not polynomial time computable.

A simple cycle containing all the nodes in the graph is called a *Hamiltonian cycle* (HC). (The "well-known" *traveling salesman problem* (or traveling salesperson to be gender neutral) (TSP) is to find an HC of least cost in an edge weighted graph. Have you heard of this problem?.)

But suppose I know that a given graph G has Hamiltonian cycle.
How can I convince you that G has such a cycle?

I can simply show you a Hamiltonian cycle C and you can easily and efficiently *verify* that C is indeed a HC. That is, I can prove to you that G has a HC.

But can I prove to you the G does *not* have a HC?

"Probably not"

NP: the class of languages which are "efficiently verifiable"

Using the HC problem as an example, lets define what it means to be efficiently verifiable.

Let L be a language (like HC) that saitisfies the following conditions: There is a polynomial time decidable relation $R(x, y)$ and a polynomial p such that for every x, $x \in L$ if and only if there exists a y with $|y| \leq p(|x|)$ and $R(x, y) = TRUE$.

$R(x, y)$ is a *verification* relation (or predicate) and y is called a *certificate* with respect to R that verifies x being in L.

Definition: The class NP is the class of languages (decision problems) that have such a verification relation and certificate.

For example L_{HC} is in NP. Namely, given a representation x of a graph $G = (V, E)$, a certificate y is an encoding of a sequence of vertices specifying a Hamiltonian cycle C. $R(x, y)$ checks the conditions for C being a simple cycle containing all the nodes in V.

Many many decision problems are in the class *NP*

First we will note that the class P (decision problems decideable in polynomial time) is a subset of NP; that is, $P \subseteq NP$. Is this obvious?

Many many decision problems are in the class NP

First we will note that the class P (decision problems decideable in polynomial time) is a subset of NP; that is, $P \subseteq NP$. Is this obvious?

Suppose a language L (like $L_{connected}$) is decideable in polynomial time. Then in the definition of NP, we can let let $R(x, y)$ be the relation that is $TRUE$ iff $x \in L$ ignoring y and $R(x, y)$ is polynomial time since we can decide if $x \in L$ in polynomial time by the assumption that $L \in P$.

Many many decision problems are in the class NP

First we will note that the class P (decision problems decideable in polynomial time) is a subset of NP; that is, $P \subseteq NP$. Is this obvious?

Suppose a language L (like $L_{connected}$) is decideable in polynomial time. Then in the definition of NP, we can let let $R(x, y)$ be the relation that is $TRUE$ iff $x \in L$ ignoring y and $R(x, y)$ is polynomial time since we can decide if $x \in L$ in polynomial time by the assumption that $L \in P$.

In saying $P \subseteq NP$, we have left open the possibility that $P = NP$. However, the widely believed assumption (conjecture) is that $P \neq NP$. This question (conjecture) was implicitly asked by (for example) Gauss (early 1800's), von Neumann, Gödel (1950's) , Cobham, and Edmonds (1960s). The conjecture was formalized by Cook in 1971 (indpendently by Levin in the FSU but his work was not known until about 1973).

More specifically Cook defined the concept of NP-completeness and gave a couple of examples of such problems, namely SAT and $CLIQUE$, problems in NP that are believed to *not* be in P. We wil define NP-completeness and the evidence for the conjecture that $P \neq NP$.

Some other examples of decision problems in NP and believed to not be in P

In all of the examples below we always assume some natural way to represent the inputs as strings over some finite alphabet. In particular, integers are represented in say binary or decimal. Polynomial time means time bounded a polynomial $p(n)$ where n is the length of the input string. (I will explain each of the following decision problems as we introduce them. Some problems are naturally decision problems. Others are decision variants of optimization problems and other relations or functions)

- $SAT = \{F | F$ is a propositional formula that is *satisfiable*$\}$
- $PARTITION = \{(a_1, a_2, \ldots, a_n) | \exists S : \sum_{a_i \in S} a_i = \frac{1}{2} \sum_{i=1}^{n} a_i\}$
- $VERTEX\text{-}COLOUR = \{(G, k) | G$ can be vertex coloured with k colours$\}$
- $FACTOR = \{(N, k) | N$ is an integer that has a proper factor $m \leq k\}$

You should be able to provide certificates for the above problems with respect to natural verification predicates.