Great Ideas in Computing

University of Toronto CSC196
Fall 2025

Class 12: October 15 (2025)

Announcements and Agenda

Announcements

- Our next guest lecture will be given on October 24 by Professor Chris Madison on the subject of machine learning (ML).
- Our first (of two) quizzes will be held this Friday, October 17 in the tutorial room BA 2139. I will discuss the nature of the quiz in today's class.

Today's Agenda

- Questions about assignment due today and/or upcoming quiz.
- New topic: Complexity Theory. The extended Turing-Church thesis. The class of problems P that are decideable in polynomial time. The class NP of decision problems that are verifiable in polynomial time. The million \$ question: Is $P = NP$?.

Announcements and Agenda

Announcements

- Our next guest lecture will be given on October 24 by Professor Chris Madison on the subject of machine learning (ML).
- Our first (of two) quizzes will be held this Friday, October 17 in the tutorial room BA 2139. I will discuss the nature of the quiz in today's class.

Today's Agenda

- Questions about assignment due today and/or upcoming quiz.
- New topic: Complexity Theory. The extended Turing-Church thesis. The class of problems P that are decideable in polynomial time. The class NP of decision problems that are verifiable in polynomial time. The million \$ question: Is $P = NP$?. Spoiler: "we" strongly believe $P \neq NP$.

Complexity theory; 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 M such that on every input x, M halts and outputs $f(x)$. That is, the Church-Turing thesis equates the informal concept of "computable" with the mathmatically 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 f is efficiently computable if there is a Turing machine M and a polynomial $p(n)$ such that on every input x, 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.

The extended Church Turing thesis continued

In what follows, I will use n to be the length of a an input string; $n = |x|$.

Do we believe the extended Church Turing Thesis?

Why we might 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 bit operands is $O(1)$. This is a robust definition.
- That is, there is a polynomial function $p_2()$ such that if a function f is computable in time $p_1(n)$ on a von Neumann random access machine, then 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 $p(n) = n^6$.
- For problems involving say enormous graphs, we may need sublinear time; for other problems we may need linear or near linear times. But as an abstraction, we are saying that a polynomial time algorithm is "efficient".

Why we should 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 Chruch-Turing thesis.

- An algorithm running in a polynomial time bound like n^{100} is not an efficient algorithm.
- An algorithm running in an exponential time bound like $(1 + \frac{1}{1000})^n$ is an efficient algorithm for reasonably (but not too) large input lengths. Note: $(1 + \frac{1}{k})^k \rightarrow e \approx 2.72$
- 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 it is possible that we will have to change of definition of "efficiently computable" to be polynomial time on a quantum 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 usually encounter polynomial time bounds asymptotically bigger than n^3.
- The robustness of polynomial time (in terms of being closed under composition is 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 a commercial sense.

But what if quantum computers become practical?

Lets assume the quantum computers or other non-classical computers become practical. We are about to discuss the P vs NP issue and the $P \neq NP$ 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 (e.g., quantum) computers?

But what if quantum computers become practical?

Lets assume the quantum computers or other non-classical computers become practical. We are about to discuss the P vs NP issue and the $P \neq NP$ 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 (e.g., quantum) 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. Moroever, our current belief is that there are problems in the class NP that are not computable in polynomial time on a quantum computer.

Polynomial time computable decision problems

We will now restrict attention to decision problems; that is $f : \Sigma^* \rightarrow \{YES, NO\}$. Σ is a finite alphabet and Σ^* is the set of all strings over Σ. We can also identify $\{YES, NO\}$ with say $\{1, 0\}$.

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; that is the languages that are "efficiently decideable".

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 $L_{connected} = \{G = (V, E) | G \text{ is a connected graph}\}$.

It is not difficult to show that $L_{connected}$ is in the class 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* (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* (TSP) is to find an HC of least cost in an edge weighted graph. Have you heard of this problem?.)

But suppose 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* (TSP) is to find an HC of least cost in an edge weighted graph. Have you heard of this problem?.)

But suppose 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 (assuming I know 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* (TSP) is to find an HC of least cost in an edge weighted graph. Have you heard of this problem?.)

But suppose 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 (assuming I know 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 efficiently prove to you that 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* (TSP) is to find an HC of least cost in an edge weighted graph. Have you heard of this problem?.)

But suppose 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 (assuming I know 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 efficiently prove to you that 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 L_{HC}) that satisfies the following conditions: There is a polynomial time decideable 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 that verifies x being in L.

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

For example 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 $y = C$ being a simple cycle containing all the nodes in V .