Great Ideas in Computing

University of Toronto CSC196
Fall 2025

Class 13: October 20 (2025)

Announcements and Agenda

Announcements

- Our second guest lecture will be given this Friday, October 24 by Professor Chris Maddison on the subject of maching learning. I strongly advise everyone to attend this talk. It will be highly informative and I plan to ask a question on the assignment or the next quiz relating to this topic.
- I will be gradually adding questions for Assignment 3.

Today's Agenda

- NP, the class of decisions problems verifiable in polynomial time.
- The million $ question: Is $P = NP$?
- The question, the conjecture $P \neq NP$, and a little of its history.
- Efficient (i.e. polynomial time) reductions: $\leq_T^{poly}$ and $\leq_{trans}^{poly}$
- NP-complete decision problems.
- Some examples of N-complete problems.
- An example of a problem *believed to be* in NP but not NP-complete.

NP: the class of languages (decision problems) 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 .

The million $ question: Is $P \neq NP$

Recall that P is the class of language (decision problems) decideable in polynomial time. It is hard to overstate the importance of this question and the impact it has had.
This is literally (and not just figuratively) a million $ question for someone who solves the question. In fact, it is worth much more than just one million $ for a proof that either $P = NP$ or a proof that $P \neq NP$.

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 $P \neq NP$.

The important consequence of NP completeness is that if *any* NP decision problem turns out to be in P, then $P = NP$. Since we strongly believe $P \neq NP$, this means that we strongly believe that no NP complete problem can be 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?

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?

Consider a language L (like $L_{connected}$) that 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?

Consider a language L (like $L_{connected}$) that 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 (independently 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 $P \neq NP$.

Efficient reductions

At the heart of *NP* completeness and more generally algorithm analysis is the concept of (efficient) reduction of problems. When we say that problem A "efficiently" reduces to problem B, we can conclude that an efficient algorithm for B will result in an efficient algorithm for A (and equivalently, the contrapositive states that A not efficiently computable implies that B is not efficiently computable).

There are different definitions for what we mean by an efficient reduction and the precise definition matters in terms of what we want to conclude from the reduction.

One major distinction is between a very general type of reduction (which we will just call *poly time reduction* (i.e., the poly time version of Turing reduction) and the more restricted reduction which we will call *poly time transformation* (i.e., the polynomial time version of a Turing computable transformation).

Two types of reductions continued

The general version of reduction $A \leq_T^{poly} B$ means that there is a poly time algorithm ALG that can call a subroutine for B (as often as it likes) and ALG computes A. Here we count each call to the subroutine as 1 step. It is not difficult to see that if $A \leq_T^{poly} B$ and B is computable in polynomial time, then A is computable in polynomial time.

The $\leq_T^{poly}$ reduction is what Cook used in his seminal 1971 paper.

The more restricted transformation (which we call a polynomial time transformation) $A \leq_{trans}^{poly} B$ means that there is a polynomial time function h (transforming an input instance of A to an input instance of B) such that $x \in A$ if and only if $h(x) \in B$. Note that $|h(x)| \leq p(|x|)$ for some polynomial p. Why?

Two types of reductions continued

The general version of reduction $A \leq_T^{poly} B$ means that there is a poly time algorithm ALG that can call a subroutine for B (as often as it likes) and ALG computes A. Here we count each call to the subroutine as 1 step. It is not difficult to see that if $A \leq_T^{poly} B$ and B is computable in polynomial time, then A is computable in polynomial time.

The $\leq_T^{poly}$ reduction is what Cook used in his seminal 1971 paper.

The more restricted transformation (which we call a polynomial time transformation) $A \leq_{trans}^{poly} B$ means that there is a polynomial time function h (transforming an input instance of A to an input instance of B) such that $x \in A$ if and only if $h(x) \in B$. Note that $|h(x)| \leq p(|x|)$ for some polynomial p. Why?
It is easy to see that $A \leq_{trans}^{poly} B$ and $B \in P$ implies $A \in P$.

Following Cook's paper, Karp provided a list of 21 combinatorial and graph theoretical problems that are NP complete. Karp used the more restrictive $\leq_{trans}^{poly}$. If you like names associated with these reductions then we can denote $\leq_T^{poly}$ as $\leq_{Cook}$ and $\leq_{trans}^{poly}$ as $\leq_{Karp}$.

NP-completeness wrt reductions $\leq_T^{poly}$ and $\leq_{trans}^{poly}$

Let's first explicitly give the definition NP-complete.

Definition: A language (or decision problem) L is NP complete if

1. $L \in NP$.
2. L is NP-hard *with respect to some polynomial time reduction*, for example with respect to either $\leq_T^{poly}$, or $\leq_{trans}$ *poly*. That is, if we are using $\leq_{trans}^{poly}$, then L is NP-hard if for every $A \in NP$, there is a polynomial time computable function h such that $w \in A$ if and only if $h(w) \in L$.

It is not difficult to show :

Fact: $B \in NP$ and $A \leq_{trans}^{poly} B$ implies $A \in NP$.

However, we do not believe that $B \in NP$ and $A \leq_T^{poly} B$ implies $A \in NP$. That is, we do not believe that the class NP is closed under $\leq_T^{poly}$ but is provably closed under $\leq_{trans}^{poly}$.

If I do not say otherwise, when only considering decision problems, I will use the more restrictive $\leq_{trans}^{poly}$.

The importance of NP-completeness

To personalize the notation, we can sometimes use $\leq_{Cook}$ (respectively, $\leq_{Karp}$) for general poly time reductions (respectively, polynomial time transformations).

Basic Fact:

One thing we want in any science (or engineering) is for concepts to be related and organized so as to understand things better. We also wants concepts to have impact and insight.

This is what NP completeness gives us. At some level of generality, we can understand an important class of problems (almost all combinatorial decision and optimization problems, many problems in number theory, and much more) in terms of one NP complete problem. We have:

If L is NP-complete (wrt to either $\leq_{Cook}$ or $\leq_{Karp}$), then $L \in P$ if and only if $P = NP$.

End of Monday, October 20 class

We end at this point, with the definkition of NP-complete decision problems and some motivation for the importance of the $P \neq NP$ conjecture.

I am attaching some additional slides for a little better context. We will continue the discussion in our next class.

Some examples of *NP* complete decision problems

In our examples 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 by 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. Each of these decision problems are easily seen to be in *NP* (i.e. it is easy to provide a verification proedicate and succinct certificate). We will soon define completeness and indicate why each of these problems is *NP* complete.

- L_{HC} as defined previously; i.e., the set of graphs that have a Hamiltonian cycle.
- $SAT = \{F | F$ 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$\}$

A example of a language in NP language that is believed to not be NP complete and believed to not be in P

$FACTOR = \{(N, k) | N$ is an integer that has a proper factor $m \leq k\}$

It is not difficult to see that $FACTOR$ is in NP.

Suppose $FACTOR \in P$. Can you then see how to factor a number N (i.e. provide the prime factorization) in polynomial time? That is, in time $p(n)$ for some polynomial P where $n = \log N$ is the length of the encoding.

We may have mentioned before that it is widely believed that we cannot factor integers in polynomial time and we use that assumption for some cryptographic applications.

The problem of efficiently factoring goes back at least two centuries to Gauss. We will soon see some evidence that $FACTOR$ is not complete.
Note: To determine if an interger is prime or composite is computable in polynomial time while we believe FACTOR is not computable in polynomial time.

NP hardness

A decison problem (or any problem, like say an optimization problem) F is NP-hard if *every* problem $L' \in NP$ (equivalently, any NP complete decision problem) can be "efficicently reduced" to F. While for decidsion problems $\leq^{poly}_{trans}$ is usually sufficent, we will need $\leq^{poly}_{T}$ when L is not a decision problem.

A decision problem L is NP-complete if it is both in NP and NP-hard. **Here again are the immediate consequences of a problem being NP-complete.**

- If L is NP complete, and $L \in P$, then every $L' \in NP$ is in P
- Equivalently, if any $L' \in NP$ is not in P, then every NP-complete problem is not in P.
- There are hundreds (and really thousands) of problems that are NP-complete and since we *"religously"* believe $P \neq NP$, we believe that none of these complete problems can be decided in polynomial time. I emphasize that this is in terms of *worst case* complexity and optmality for optimization problems.

Why the religious belief

Why do we believe so strongly that $P \neq NP$. It is simply that many very talented people over literally centuries have tried to efficiently solve problems that are in NP (and believed to not be in P and especially those that are NP-complete) and failed to do so.

Even so, there have been surprises in complexity theory and one still has to keep in mind that $P \neq NP$ is still a conjecture and not a proven result.

Our confidence in this conjecture is strong enough that modern day cryptography makes this assumption and indeed makes even stronger assumptions. For example, cryptographic protocols usually assume that there exist *one-way functions* f for which it is easy (i.e. poly time) to compute $f(x)$ for any x but given y, it is difficult to find an x such that $f(x) = y$.

For cryptography we also need assurance that a problem is not only hard in a worst case sense but also hard in some "average case" sense.