Great Ideas in Computing

University of Toronto CSC196
Fall 2025

Class 17: November 5 (2025)

Announcements and Agenda

Announcements

- Our third guest presentation will be given on Monday, November 10 by Professor Niv Nayan on the subject of Data Structures.
- Assignment 3 was due this Wednesday, November 5 at 3PM. I added one thought question to the two questions previously listed. I have extended the due date until Friday, November 7 a 3PM.
- Our fourth and final guest presenter will be Jonathan Panuelos who will discuss "computational mechanics" which relates to the simulation of physical systems. His presentation will be on Wednesday, November 19.

Today's Agenda

- Finish complexity theory. Reducing optimization and search problems to the underlying decision problem. Some other issues in complexity theory.

Search and Optimization problems

Each of the problems in the Karp tree has an associated optimization problem or search problem. For example, the *Vertex-Cover* problem is usually expressed as the following optimization problem:
Given a graph $G = (V, E)$, find a minimum size vertex cover for G; that is, a subset $V' \subset V$ such that for every edge $e = (u, v) \in E$, either $u \in V'$ or $v \in V'$. This is the inculusive "or" so that it is possible that both u, v are in V'.

If we can solve the optimization problem efficiently, we can immediately solve the decision problem. Does everyone understand this?

What is not as immediate, is the fact that if we can solve the *Vertex-Cover* decision problem then we can solve the *Vertex-Cover* optimization problem.

We would do this by first determining (using the decision problem) the size of the minimum vertex cover. Does everyone see how to do this?

The vertex-cover optimization problem continued

Suppose k is the size of the minimum vertex cover. We want to create a vertex cover V' one vertex at a time starting with $V' = \varnothing$. We iteratively decide for each vertex v, whether or not we can include $v \in V'$. That is, we determine if we can remove v and all its adjacent edges and if the resulting graph $\tilde{G}$ has a vertex cover of size $k-1$ the we add v to the cover V' that we are creating. We then continue with the graph $\tilde{G}$ trying to create a cover if size $k-1$. If $\tilde{G}$ does not have a vertex cover of size $k-1$, we go back to graph G and try another node u to see if $\tilde{G}$ has a vertex cover of size $k-1$ if we remove u and its adjacent edges. We keep searching for a vertex x that we can remove from G and add to V'.

Note that while we usually restrict attention to $\leq^{poly}_{trans}$ (polynomial time transformations) for the purpose of showing new problems are NP-complete, we are using the more general $\leq^{poly}_{T}$ (polynomial time reductions) to reduce the optimization problem to the decision problem.

Another conjecture: $NP \neq$ co-NP

FACT: If L is NP-complete wrt $\leq^{poly}_{trans}$ then $\bar{L} \in NP$ if and only if $NP =$ co-NP

There is another widely believed conjecture again based on the inability of experts to show that $\bar{L} \in NP$ for any NP-complete problem which states that $NP \neq$ co-NP. For example, as stated before, we do not believe there is a "short" certificate for showing that a graph does *not* have a Hamiltonian cycle.

As I mentioned before, we believe factoring intergers is not polynomial time computable. In fact, there is a sense in which we believe it is not polynomial time computable "on average" (whereas the basic theory of NP completeness is founded on worst case analysis).

Surprisingly, co-$FACTOR$ is in NP. That is, given an input (N, k), we can provide a certificate verifying that N does *not* have a proper factor $m \leq k$.

Since co-$FACTOR$ is in NP, and we conjecture that $NP \neq$ co-NP, this leads us then to believe that $FACTOR$ is in $NP \setminus P$ but *not* NP-complete.

Returning to the two different reductions

As far as I know, there is no proof that the two reductions are different but there is good reason to believe that they are different in general.

- Clearly $\bar{A} \leq_T^{poly} A$ for any language A.
- $A \leq_{trans}^{poly} B$ and $B \in NP$ implies $A \in NP$.
- Hence our assumption that $NP \neq co - NP$ implies that we *cannot* have $\bar{A} \leq_{trans}^{poly} A$ for any NP-complete A.

On the other hand as far as I know all known NP complete problems have been shown to be complete using transformations $\leq_{Karp}$.

I know of no compelling evidence that general reductions and transformations are different when restricted to the class NP.

NOTE: The general reduction concept is needed when reducing say a search or optimization problem to a decision problem (and indeed this is what we described for *Vertex-Cover* and we will be doing next for *SAT*). On the other hand, transformations are what we use for decision problems (i.e., languages).

Finding a certificate for an *NP*-complete problem

One might wonder if we can always efficiently *find* a certificate if we can decide whether or not a certicifcate exists. In fact, for *NP*-complete problems we can polynomial time reduce finding a certificate to deciding if a certificate exists.

Fact Let L be a *NP*-complete problem. We can prove that for every YES input instance x (where we know that a certificate exists wrt some verification predicate) that a certificate can be computed in polynomial time assuming we can solve the decision problem in polynomial time.

Of course, we do not believe that an *NP*-complete decision problem can be solved in polynomial time so this is just a claim that it is sufficient to just focus on the decision problem.

As another example, consider *SAT* and suppose F is satisfiable. That means we can set each propositional variable (to TRUE or FALSE) so that the formula evaluates to TRUE. So how do we find a satsifying truth assignment for F?

Finding a satisfying assignment for a formula F assuming $P = NP$

Once we assume $P = NP$, we would know that the decision problem for SAT is satisfiable. So we would first test if the given formula F is satisfiable. If so, we can construct a satisfying assignment one variable at a time. Consider the following example:

$$F = (\bar{x}_1 \vee x_2) \wedge (\bar{x}_2 \vee x_3) \wedge (\bar{x}_3 \vee \bar{x}_1) \equiv (x_1 \rightarrow x_2) \wedge (x_2 \rightarrow x_3) \wedge (x_3 \rightarrow \bar{x}_1)$$

Now since F is satisfiable, there must be some way to set (say) x_1 to either TRUE or FALSE so that the resulting formula still is satisfiable.

If we set x_1 to TRUE, then the resulting formula $F' = F|_{x_1 = TRUE}$ will become FALSE so it must be that x_1 is FALSE in any satisfying assignment.

How would we know that $F' = F|_{x_1 = FLASE}$ is satisfiable?

Finding a satisfying assignment for a formula F assuming $P = NP$

Once we assume $P = NP$, we would know that the decision problem for SAT is satisfiable. So we would first test if the given formula F is satisfiable. If so, we can construct a satisfying assignment one variable at a time. Consider the following example:

$$F = (\bar{x}_1 \vee x_2) \wedge (\bar{x}_2 \vee x_3) \wedge (\bar{x}_3 \vee \bar{x}_1) \equiv (x_1 \rightarrow x_2) \wedge (x_2 \rightarrow x_3) \wedge (x_3 \rightarrow \bar{x}_1)$$

Now since F is satisfiable, there must be some way to set (say) x_1 to either TRUE or FALSE so that the resulting formula still is satisfiable.

If we set x_1 to TRUE, then the resulting formula $F' = F|_{x_1 = TRUE}$ will become FALSE so it must be that x_1 is FALSE in any satisfying assignment.

How would we know that $F' = F|_{x_1 = FLASE}$ is satisfiable? We would again use the decision procedure SAT applied to F'. We would continue this way to see how to set x_2, x_3. In this example, x_2 can be set TRUE or FALSE and we would just choose one value. In general, a formula can have many satisfying assignments.

NP completeness is a great idea

I know some (many?) students may find this to be difficult material as you would not have seen it before. Please ask questions

I do think this material is fundamental to computer science (as a discipline) and computing (in terms of its impact).

Some ideas are great ideas even when we are not that aware of them. I argued that this was the case with respect to Turing's work and the von Neumann model.

The concept of *NP* completeness is something that algorithm designers may or may not think of routinely but at some level of understanding we do need to know that common (say optimization) problems cannot be solved effciciently for all input instances.

I note that there have been many surprises in complexity theory so I again emphasize that a conjecture may guide our thinking but we always have to be aware of what has and has not been proven.

Can randomization help?

We should note that there are many other fundamental questions in complexity theory (in addition to the P vs NP question). One such question is can randomization help.

Consider the following problem: We are *implicitly given* two multivariate polynomials $p(x_1, \ldots, x_n)$ and $q(x_1, \ldots, x_n)$. For example, the polynomials might be the result of a polynomial time computation using the arithmetic operations $+, -, *$. Or p and q might be the determinants of $n \times n$ matrices with entries that are linear functions of the $\{x_i\}$.

The *polynomial equivalence question* asks whether or not $p \equiv q$ as polynomials; that is, does $p(x_1, \ldots, x_n) = q(x_1, \ldots, x_n)$ for all values of the $\{x_i\}$. Lets say that the x_i are all integers or rationals.
Note that this is the same as asking whether or not $p - q \equiv \mathbf{0}$ where $\mathbf{0}$ is the zero polynomial.

How would you solve the *identically zero* question for a univariate polynomial (again given implicitly)?

Polynomial equivalence problem continued

Fact: A non zero univariate polynomial $p(x)$ of degree d has at most d distinct zeros. This means that if we evaluate $p(x)$ at say $t > d$ random points $r_1, \ldots r_t$, the probability that $p(r_i) = 0$ is at most $\frac{d}{t}$.

Schwartz-Zipple Lemma: This lemma extends the above fact to multivariate polynomials. That is,
If $p(x_1, \ldots, x_n)$ is a non zero polynomial of total degree d (with coeficients in a ring or field F like the integers or rationals) then
$Prob[p(r_1, \ldots r_n) = 0] \leq \frac{d}{|S|}$ when the r_i are chosen randomly in a finite subset $S \subseteq F$.

Polynomial equaivalence and the class *RP*

So to test if p is identically zero, we take $|S|$ sufficienlty large (or do repeated independent trials with say $|S| = 2d$), and see if the evaluation returns a non-zero value. If $p(r_1, \ldots, r_n) = 0$, we will claim that $p \equiv \mathbf{0}$. The error in this claim will be at most $\frac{d}{|S|}$ and we will only make an error if $p \not\equiv \mathbf{0}$.

This is an example of a polynomial time randomized algorithm with 1-sided error (with say error at most $\frac{1}{2}$) and RP is the class of languages that have such an algorithm.

In fact the error can be as big as $1 - \frac{1}{n^k}$ for any fixed k as we can do polynomially many repeated trials to reduce the error probability using the fact that $(1 - 1/t)^t \to \frac{1}{e}$ as $t \to \infty$.

Open question: Is $RP = P$? As a specific example, is the polynomial equivalence problem in P?

RP and *BPP*

Surprisingly, some prominent complexity theorists (but not everyone) believe $P = RP$. More generally, they believe $BPP = P$ where BPP is the class of languages that can be solved by a polynomial time randomized algorithm with 2-sided error (with probability of error at most $\frac{1}{2} - \frac{1}{n^k}$).

For BPP, we can amplify the probability of a correct answer by running a polynomial number of trials and taking the "majority vote" amongst the outcomes of the individual trials.

A langauge in RP can be formulated so that there are many certificates and hence $RP \subseteq NP$.

One final comment about the conjecture $P \neq NP$. While we strongly believe $P \neq NP$, all is not lost if $P \neq NP$. For example, for an optimization problem, while it may be NP-hard to compute an *optimal solution*, for many NP-hard problems there are efficient *approximately optimal* algorithms. And many natural problems have efficient algorithms when considering restricted classes of (or distributions over) instances that tend to occur naturally.

Are there other issues in complexity theory?

Arguably, the central question in complexity theory is "what is and what isn't computable in deterministic and randomized polynomial time". But there are many other interesting issues.

Here are some other issues.

- For problems computable in polynomial times, what is the best time bound? This topic is often referred to as "fine grained complexity".
- What can and can't be done in polynomial space ($PSPACE$). Note $NP \subseteq PSPACE$ but we believe it is a much bigger class?
- What can be said about time complexity when input instances are coming from some known or unkown distribution (rather than worst case instances)? In our next topic *complexity based cryptography*, we base the security of cryptographic protocols on "average case behaviour".
- Can we prove that P contains decision problems not computable in log space. Here we have to define a Turing machine model that allows us to define $\log(n)$ space where n is the length of the input.