Great Ideas in Computing

University of Toronto CSC196
Fall 2021

Week 9: November 15-19 (2021‘)

Announcements

Announcements

- I have posted all of Assignment 3 (A3) on the web page. A3 is due this Friday at 8AM,.
- The next quiz is scheduled for November 26.
- We have our final guest presenter this Wednesday, November 17. Professor Fanny Chevalier will discuss human computer interaction HCI and more specifically visualization.

This weeks agenda

Agenda

- We will continue our discussion of complexity theory, NP-completeness and the $P \neq NP$ conjecture.
- We ended the Friday, November 5 class on slide 13 with an example of a language L_{HC} in the class NP.

  We will begin today by restating that slide, then giving the definition of the class NP and work our way towards the definition of NP-completeness and the $P \neq NP$ conjecture. .
- Then (probably next week) we begin complexity based cryptography.

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 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)

- $SAT = \{F | F$ 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.

NP completeness

We will see that with the exception of FACTOR, the other decision problems are *NP*-complete, a concept we will now motivate and define.

A decison problem (or any problem) L is *NP*-hard if *every* problem $L' \in NP$ can be "efficicently reduced" to L. There are different notions of how to formalize "efficiently reduced" and we will discuss this shortly.

A problem L is *NP*-complete if it is both in *NP* and *NP*-hard.
Here 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 this is in terms of *worst case* complexity.)

Why the religious belief and co-*NP*

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 $f(x) = y$.

What would happen if someone solves the P vs NP question?

I was asked at the end of our last class (on Friday, November 5) what would be the consequences if someone resolves the P vs NP question.

What would happen if someone solves the P vs NP question?

I was asked at the end of our last class (on Friday, November 5) what would be the consequences if someone resolves the P vs NP question.

While the mathematical and scientfiic impact will be enormous, science will not end.

If someoone proves that (as we do not believe) $P = NP$, then the "pracical impact" will depend on how efficiently we can solve NP complete problems; that is, what are the polynomial time bounds.

If someone prove $P \neq NP$, then the "practical impact" will depend on whether or not a given problem can be solved efficiently "in practice" (i.e. for most inputs of for "the inputs we care about"). More later.

Returning to the concept of reduction

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*.

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 $A \leq_T B$ 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 $h(x) \in B$.
It is again 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 $\leq_{Karp}$.

Comparing the reductions $\leq_T^{poly}$ and $\leq_{trans}^{poly}$

Let 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 is NP-hard if for every $A \in NP$, there 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 $A \in NP$.

The importance of NP-completeness

To simplify the notation, lets sometimes use $\leq_{Cook}$ and $\leq_{Karp}$ respectively for genneral poly time

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

It can be shown that $L_{HC}, SAT, Partition, Vertex\text{-}Colour$ and thousands of other problems are NP complete.

So now we know that if we can polynomial time decide any one of these NP-complete problems we can solve them all in polynomial time. Any even if a problem L is in NP but possibly not NP-complete, then $P = NP$ would imply L is polynomial time decidable.

But how do we prove that prove that a decision propblem is NP-complete?

Suppose we know that some problem (for example, SAT) is NP-complete. Then if we can show SAT can be poly time reduced or transformed to (for example) $VC = \textit{vertex-cover}$, then VC must also be NP-complete.

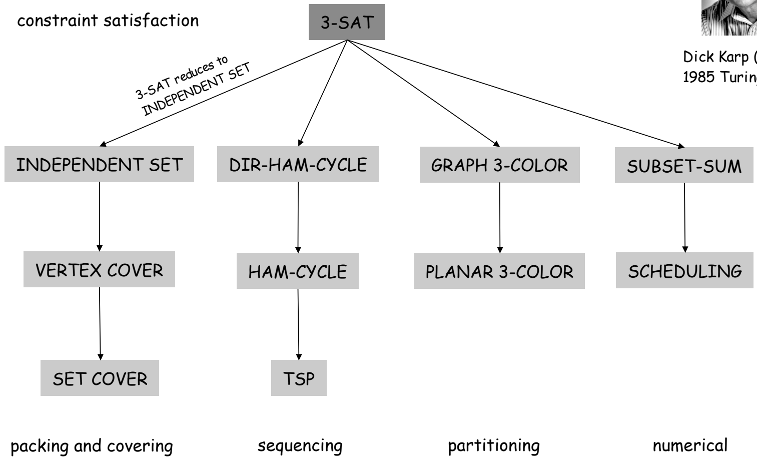
Fact: Polytime reductions and polytime transformations are transitive relations. That is, for example, $A \leq_{Karp} B$ and $B \leq_{Karp} C$ implies $A \leq_{Karp} C$.

In this way, thousands of decision problems L has been created by a tree of polynomial time transformations. (On the next slide, we will show Karp's initial tree.) But we have to start the tree with some NP problem that we prove is NP-complete.

Cook did this for SAT by showing how to efficiently encode any polynomial time Turing machine computation of a verification predicate $R(x, y)$ and a "guess" for a certificate y within propositional logic. (We discuss this in CSC373.)

A tree of reductions/transformations

Polynomial-Time Reductions



Dick Karp (1972)
1985 Turing Award

End of Monday, November 15 class

We ended the class just showing the initial Karp tree of reductions. In next Monday's class I will quickly show some simple reductions and how we can "Cook reduce" (i.e., using $\leq_T^{poly}$) optimization problems and finding certifcates to the corresponing decision problem. Wednesday we have our final guest presentation.

Polynomial time transformations are algorithms and some are"easy'and some can be very difficult to derive and prove that the transformation $h()$ satisfies what is needed for a poly time transformation $\leq_{trans}^{poly}$.

I am posting on quercus a link to a video by David Scot Taylor at San Jose State University that explains very nicely some polynomial time transformatons.

Optimization problems

Each of these problems has an associated optimization 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 then 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 djacent edges and ask if the resulting graph $\tilde{G}$ ihas a vertex cover of size $k-1$. If v cannot be in the minimum vertex cover then we go on to look at another vertex.

Note that while we can "probably" restrict attention to $\leq_{Karp}$ polynomial time transformations for the purpose of showing new problems are NP-complete, we are using the more general $\leq_{Cook}$ 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 $\leq_{Karp}$ 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 $\bar{A} \leq_{Cook} A$ for any language A.
- $A \leq_{Karp} B$ and $B \in NP$ implies $A \in NP$.
- Hence our assumption that $NP \neq co - NP$ implies that we *cannot* have $\bar{A} \leq_{Karp} A$ for any NP-complete A.

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

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

NOTE: The general reduction concept makes sense when reducing say a search or optimization problem to a decision problem (and indeed this is what said about *Vertex-Cover* and we will be doing next for *SAT*). On the other hand, transformations are only about 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 (Cook) 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 the 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 an 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.