Great Ideas in Computing

University of Toronto CSC196
Fall 2023

Week 9: November 13 - November 17 (2023)

Week 9 slides

Announcements:

- Assignment 3 is due this Friday ar 9 AM. I am willing to delaly the due date until Monday, November 20 (if everyone agrees) to allow more time to discuss complexity theory.
- Second and final quiz on Friday, Novemeber 24.
- Final guest presentation Monday, Nov 27 by Kyros Kutalakos. He will be discussing computer vision.

This weeks agenda

- Finish discussion of complexity theory:
  1. Polynomial time reducibility and polynomial time transformations
  2. Definition of *NP*-completeness
  3. The Karp tree of transformations
  4. Some concluding comments on complexity theory
- Complexity based cryptography. (This will probably be next week)

Question: Why does the Arts and Science Faculty have these first year foundational courses?

Week 9 slides

Announcements:

- Assignment 3 is due this Friday ar 9 AM. I am willing to delaly the due date until Monday, November 20 (if everyone agrees) to allow more time to discuss complexity theory.
- Second and final quiz on Friday, Novemeber 24.
- Final guest presentation Monday, Nov 27 by Kyros Kutalakos. He will be discussing computer vision.

This weeks agenda

- Finish discussion of complexity theory:
  1. Polynomial time reducibility and polynomial time transformations
  2. Definition of *NP*-completeness
  3. The Karp tree of transformations
  4. Some concluding comments on complexity theory
- Complexity based cryptography. (This will probably be next week)

Question: Why does the Arts and Science Faculty have these first year foundational courses? To engage in discussions with different perspectives, and learning to express your thoughts. We have four more weeks of classes. Please participate.

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 problem, I will use the more restrictive $\leq_{trans}^{poly}$.

The importance of NP-completeness

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

There are literally thousands of NP complete decision problems.

The basic fact says that in some sense all NP problems are "equivalent" in terms of what is efficiently computable computable.

But our polynomial time reduction is not sufficiently refined to characterize what is computable in linear time, or quadratic time, etc

And when we consider optimization problems like the TSP problem, $\leq_T^{poly}$ is not sufficiently refined in terms of how well we can efficiently approximate an optimal solution.

End of Monday, November 13 class

Today was a review and clarification of material concerning the class NP, polynomial time reductions and transformations, and NP-completeness.

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 predicate and succinct certificate). We will soon 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 in CNF that is *satisfiable*$\}$
- $PARTITION = \{(a_1, a_2, \ldots, a_n) | \exists S : \sum_{a_i \in S} a_i = \frac{1}{2} \sum_{i=1}^{n} a_i\}$
- $VERTEX\text{-}COLOUR = \{(G, k) | G = (V, E)$ can be vertex coloured (by χ) with k colours such that $\chi(u) \neq \chi(v)$ if $(u, v) \in E$ $\}$

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 easy 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?

We 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 NP-complete.

NP completeness

A decison problem (or any problem, like say an optimization problem) L is NP-hard if *every* problem $L' \in NP$ can be "efficiently reduced" to L. While for decision problems $\leq_{trans}^{poly}$ is sufficent, we will need the more general $\leq_{T}^{poly}$ when L is not a decision problem.

A decision 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 that this is in terms of *worst case* complexity.)

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 (and one-way permutations) 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.

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

A frequent question that is asked is the following: What would be the consequences if someone resolves the P vs NP question

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

A frequent question that is asked is the following: What would be the consequences if someone resolves the P vs NP question

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

If someoone proves that (as we do not believe) $P = NP$, then the "practical 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 or for "the inputs we care about"). More about worst case vs "practical application" later.

But how do we prove that a decision problem 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* = *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 $A \leq_{Karp} C$.

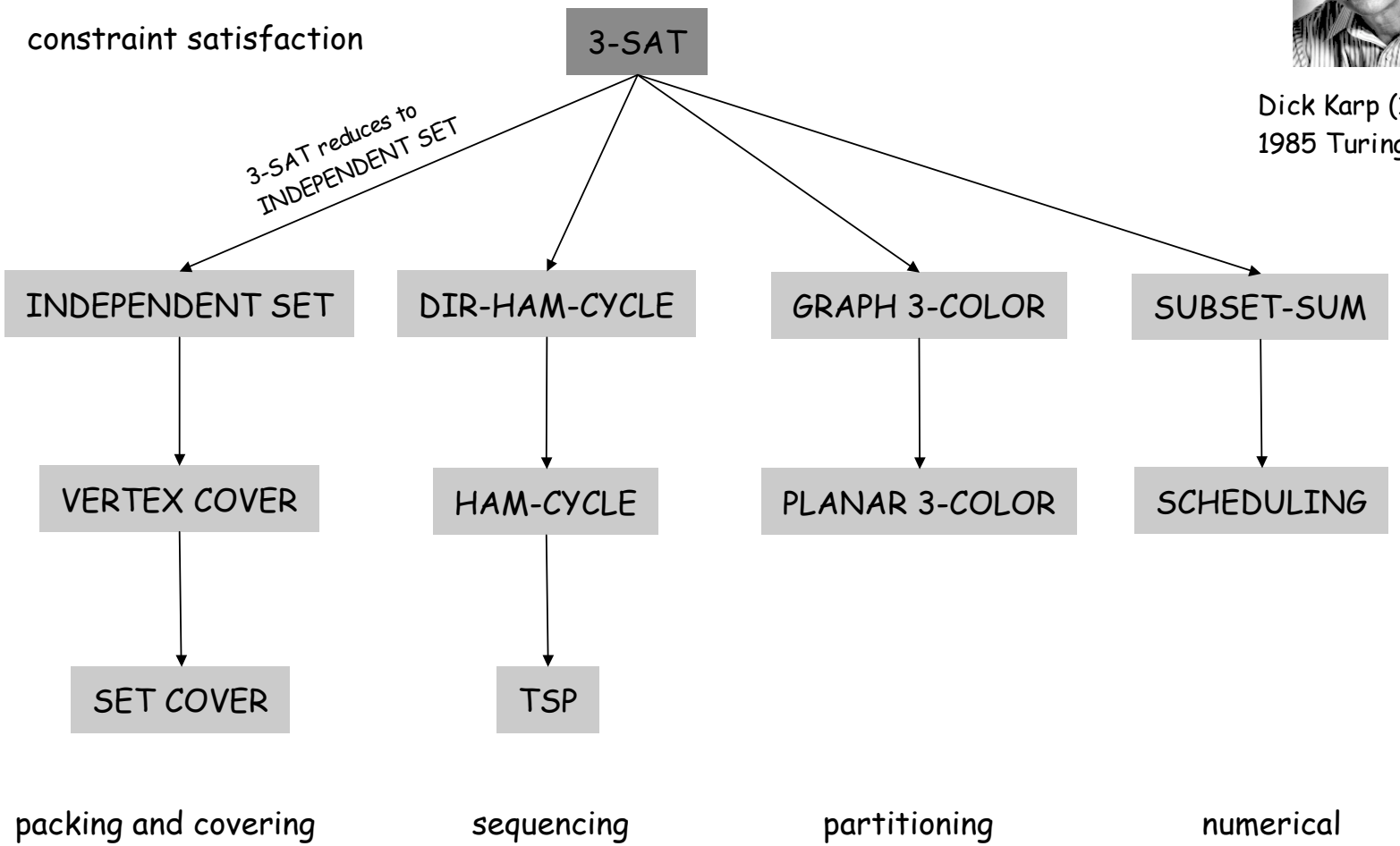
In this way, thousands of decision problems *L* have been created by a tree of polynomial time transformations. (On the next slide, we will show Karp's initial tree.) But we have to start the tree with some *NP* problem that we prove is *NP*-complete.

Cook did this for 3*SAT* (the restriction of SAT to formulas with at most 3 literals per clause.

A tree of reductions/transformations

Polynomial-Time Reductions

Dick Karp (1972)
1985 Turing Award



$3SAT$ is NP-complete

Assuming that $3SAT$ is NP-complete, it follows that all the problems in the tree are NP-complete. (Actually the reductions show that all the problems in thre Karp tree are NP-hard but they are also NP-complete since they are all easily seen to be in NP.)
An example in the Karp tree of a not so obvious transformation: $3SAT \leq^{poly}_{trans}$ Independent Set. (In Cook's paper, it was stated as 3-SAT $\leq^{poly}_{T}$ Clique.)

The problems in the Karp tree are a very small sample of the thousands of NP-complete problems.
But how do we show that $3SAT$ is NP-complete? You do not have to worry about that but here is the idea.
Suppose we have a TM $\mathcal{M}$ that we assume is executing in polynomial time (poly time in terms of the length $|w|$ of the input w to the TM. Fixing $\mathcal{M}$, the idea is to show how to encode the computation of $\mathcal{M}$ on an input w by a $3CNF$ formuala F_w. More precisely we can show:

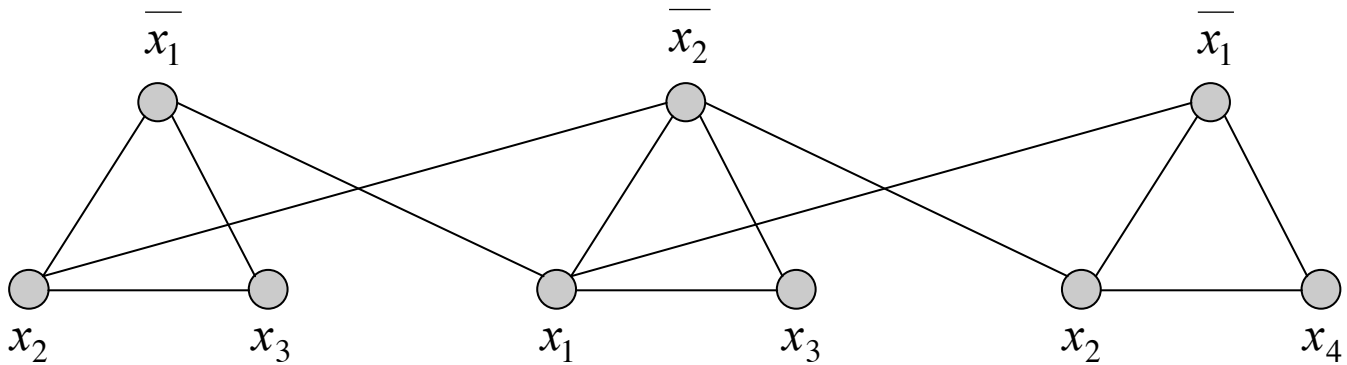
There is a polynomial time transformation h : $\mathcal{M}$ accepts w if and only if $h(w) = F_w$ is satsifiable.

3SAT reduces to Independent Set

Claim

3SAT $\leq_\tau$ Independent Set

- Given an instance F of 3SAT with k clauses, we construct an instance (G, k) of Independent Set that has an independent set of size k iff F is satisfiable.
- G contains 3 vertices for each clause; i.e. one for each literal.
- Connect 3 literals in a clause in a triangle.
- Connect literal to each of its negations.



$$\Phi = \left(\overline{x_1} \vee x_2 \vee x_3\right) \wedge \left(x_1 \vee \overline{x_2} \vee x_3\right) \wedge \left(\overline{x_1} \vee x_2 \vee x_4\right)$$

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 djacent 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 resticted 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*-complerte 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 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.

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 mentioned 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.

End of Week 9 slides

We ended at the previous slide. There are a few more slides following which discusses randomization and in particular randomized polynomial time.

While the power of randomization is not well understood (theoretically), there is an important sense in which it is provably necessary for cryptography.

Nexy week we begin cryptography.

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* 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 BP

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 $\frac{1}{2} - \frac{1}{n^k}$).

Like RP, 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.