COMPUTATION: DAY 8

BURTON ROSENBERG
UNIVERSITY OF MIAMI

Contents

1. Alternative Proof systems 1
2. Interactive Proofs 3
3. The class IP 4
4. Zero knowledge proof systems 5
4.1. Cheating Verifiers 7

1. Alternative Proof systems

An algorithm is a *proof system* for membership in a set. An NP statement

$$a \in A \iff \exists \pi \, V(\pi, a) = T$$

is interpreted as π providing a proof that $a \in A$, and V verifying that the proof is valid. The requirements are that π must be short as well as convincing.

One has this intuition that a proof π is hard to come by. But there are cases were proofs are plentiful so that choosing a few at random strings, it is exceedingly likely that one constitutes a proof.[1] For instance, Little Fermat, a theorem in number theory, says that if n is a prime integer than for all integers w relatively prime to n,

$$w^{n-1} = 1 \pmod{n}$$

Hence a w that defies this equality proves n not to be a prime. If $1 < w < n$ is not relatively prime to n then n is also not a prime.

Example: In Figure 1 we show the result for each w when $n = 35$. Only 1 through

Date: 28 April 2025.

[1]The notion of proof as taught by Euclid is one of absolutes, however that is not entirely true because some sets are undecidable. In the case of a Σ_1 set S, if the truth is an s is in S, there is a proof of this fact. However there are $s \notin S$ for which there is no proof of this fact.

w	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
$w^{34}(35)$	1	9	4	11	30	1	14	29	16	25	11	9	29	21	15	16	4

FIGURE 1. Witness of 35 being composite

w	1	2	3	4	5	6	7	8	9	10	11	12
$w^{24}(25)$	1	16	6	6	0	21	1	21	11	0	16	11

FIGURE 2. Witness of 25 being composite

17 is shown, since $18 = -17, 19 = -16$ etc. Any w except 1, 6, 29 and 34 are proofs that 35 is not prime.

The result cannot be called a proof in the classical sense, because the argument that an integer is prime is not by a conclusive logical steps, but by a preponderance of evidence. The algorithm might be wrong, and it is not always wrong for any give prime. However, if we modify our notion of proof than many more things can be proven, and the errors can be made practically non-existent.

Definition 1.1. A language $A \subseteq \Sigma^*$ is in *randomized polynomial time* (RP) if there exists a polynomial time algorithm $V(w, a)$ such that,

$$\begin{aligned} a \in A &\iff Pr_{w \in_U W}[V(w, a) = T] > 1/2 \\ a \notin A &\iff \forall w \in W, V(w, a) = F \end{aligned}$$

Where $w \in_U W$ is a uniformly random choice of w from W, and the bound is uniform for all a, according to the case.

The set of composite numbers is RP, and the set of primes is co-RP. Given a composite n, choosing one more numbers at random, will give a witness and the output of the decider will be true. However with a probability that grows exponentially smaller with repeated trials, it might mis-classify n as prime. However, the false side of this set, that n is in fact p a prime, will always yield false.

Note that in use, the algorithm is run k times, and if the number is composite that then probability it is incorrectly classified will be less than $(1/2)^k$. In fact, since this is a polynomial time algorithm the algorithm can be repeated a linear number of times to give an exponentially small error of $(1/2)^n$,

Definition 1.2. The property that a proof system for $a \in A$ proves such, is called *completeness*. If the proof system is certain to prove $a \in A$ for any such a, the proof system has *perfect completeness*.

The property of a proof system such that for $a \notin A$ the system does *prove* falsely $a \in A$ is called *soundness*. If for all $a \notin A$ the system never falsely proves otherwise is called *perfect soundness*.

When there exists an NP proof system for a language, it has perfect completeness and perfect soundness. It is also quick — proofs are The result being a lot more languages had quick proofs that were almost certainly correct. An RP system has perfect soundness and only probabilistic completeness. However in practice there are many more things that can be proved to the satisfaction of the user of the system, with an RP system.

The RP and co-RP classes can be mapped to values T, F and $\perp$. Where RP gives values T and $\perp$, and co-RP give values F and $\perp$. This makes explicit the caveat emptor of the asymmetry. In the case of RP, the conclusion of F from $\perp$ is by preponderance of the evidence (likewise inferring T from $\perp$ for co-RP.

There is a class ZPP which is the intersection of RP and co-RP which uses this three value result explicitly. It provides evidence of true, when it finds such, evidence of false when it finds such, but with a probably less than $1/2$ will decide $\perp$, which signifies no evidence has been found either way.

A further approach, which seems to be the most realistic model of computation in use, as it is sufficiently strong for the intent and sufficiently weak that many things have proofs, is the class BPP.

Definition 1.3. A language $A \subseteq \Sigma^*$ is in *Bounded Probabilistic Polynomial-time* (BPP) if there exists a polynomial time algorithm $V(w, a)$ such that,

$$\begin{aligned} a \in A &\iff Pr_{w \in_U W}[V(w, a) = T] \geq 2/3 \\ a \notin A &\iff Pr_{w \in_U W}[V(w, a) = T] < 1/3 \end{aligned}$$

The choice of w is made uniformly at random from W, and the bound is uniform for all a, according to the case.

2. Interactive Proofs

The NP paradigm has made use of a polynomial time verifier V such that for an $A \subseteq \Sigma^*$, then $a \in A$ if and only if there exists a y and $V(y, a) = T$. This y has various interpretations. It can be a hint, or the answer; it guides the computation to prove that a is in A. The concept of NP is that one does not specify how y is found. It is sufficient that it exists.

In an *interactive proof* two Turing machines converse using a shared tape. One machine is called the Prover, and the other is called the Verifier. The Verifier is a PPT; but the Prover can be any Turing Machine. This generalizes the notion of a hint or proof string, in that the Prover can try to convince the Verifier by providing this string. In fact, the Prover and Verifier can converse in multiple rounds.

Definition 2.1. An *interative turing machine* is a pair of Turing machines, a probabalistic polynomial time (PPT) verifier V and a possibly computational unbounded prover P. They communicate by a communication tape. Given an input a the prover can help the verifier come to decision, denoted

$$\langle P, V \rangle(\omega, a) \in \{ T, F \}$$

where ω is the randomness of V chosen over some probability space Ω. The time is polynomial for V, but the time of P is not generally considered.

Definition 2.2. The class IP is the class of languages $A \subseteq \Sigma^*$ such that there exists a pair of interactive TM's $\langle P, V \rangle$ such that,

(1) For $a \in A$,
$$Pr_\Omega\big(\langle P, V \rangle(\omega, a) = T\big) \geq 2/3$$
where the probability is over $\omega \in \Omega$.

(2) For $a \notin A$, and for any prover P^*,
$$Pr_\Omega\big(\langle P^*, V \rangle(\omega, a) = T\big) \leq 1/3$$
where the probability is over $\omega \in \Omega$.

Note the asymmetry. The verifier must verify absolutely. So nothing a "cheating prover" can do will make verifier mistake an $a \notin A$ for one in A, except with limited probability.

Also note, that because of the power of the prover, the prover is generally not probabilistic. The ω is only used by the verifier. The reason is, since the Prover is so powerful, rather than randomize its actions, it will take the one action that maximizes its advantage, whether that is to convince the verifier to accept or to convince the verifier to reject.

3. The class IP

Definition 3.1. Graphs $G_1 = \langle V_1, E_1 \rangle$ and $G_2 = \langle V_2, E_2 \rangle$ are isomorphic if,

(1) There is a bijection on the vertex sets, $\phi : V_1 \to V_2$, and
(2) The bijection respects the edge structure if the graph:
$$(v, v') \in E_1 \;\Leftrightarrow\; (\phi(v), \phi(v')) \in E_2$$

We write $G_1 \cong G_2$.

Theorem 3.1. The language $ISO = \{ (G_1, G_2) \,|\, G_1 \cong G_2 \}$ is NP complete.

Proof: It is in NP because given a permutation on the vertices ϕ, it is checkable in polynomial time that $\phi(G_1) \cong G_2$.

To establish that it is NP complete, a reduction such as 3-SAT$\leq_P$ISO is required. The proof is omitted. □

Theorem 3.2. The language $NISO = \{ (G_1, G_2) \mid G_1 \ncong G_2 \}$ is in IP.

Proof: We give an $\langle P, V \rangle$ system that recognizes *NISO*.

(1) The Verifier V choses a random bit b and a random permutation ϕ. The verifier V sends the Prover P the graph $H = \phi(G_b)$.
(2) The Prover figures out for which $\tilde{b}$ does $G_{\tilde{b}} \cong H$, and sends the verifier $\tilde{b}$. If both or neither, it sends a random $\tilde{b}$.
(3) The Verifier accepts if $b = \tilde{b}$.

If the graphs are not isomorphic, then $(G_1, G_2) \in NISO$. Then the Prover can determine for which $\tilde{b}$ that $H \cong G_{\tilde{b}}$, and the verifier accepts with probability 1. If they are isomorphic then $H \cong G_1 \cong G_2$, and it was the verifier's choice, unknown to the prover which of these two isomorphisms the verifier is expecting. A random guess by the prover means the verifier accepts with probability 1/2. This is greater than the allowed error probability, so two independent trials are made, the the only accepts if both times the prover has guessed its coin. So the probability the verifier accepts is 1/4. □

4. Zero knowledge proof systems

What is it about the isomorphism between two graphs that is knowledge? Given a graph G_0, it is not knowledge to create a random permutation ϕ of the vertices of G_0 to create $G_1 = \phi(G_0)$. However given a (G_0, G_1) promised to be isomorphic, it is knowledge find the permutation ϕ on the vertices of G_0 such that $G_1 = \phi(G_0)$.

- *Polynomial time calculations do not increase knowledge. They reveal aspects of the same knowledge.*

The creation of ϕ does have a strange requirement — that the computer has access to randomness. However, how is it that the draw of random numbers is knowledge? Two machines each drawing random numbers, to the learn the same things?

In the following diagram,

$$G_1 \xrightarrow{\phi_1} \tilde{G} \xleftarrow{\phi_2} G_2$$

The graph $\tilde{G}$ is chosen randomly among graphs isomorphic to G_1. We assume G_2 is isomorphic to G_1. The two permutations ϕ_1 and ϕ_2 are equally likely, and the composition $\phi_2^{-1}\phi_1$ is the isomorphism between G_1 and G_2.

The presentation of just ϕ_1 or just ϕ_2 is not knowledge. As $\tilde{G}$ can be selected by first selecting b and the ϕ_b. Such an activity is polynomial time. However the chance that $\tilde{G}$ happens twice, once as $\phi_1(G_1)$ and again as $\phi_2(G_2)$ is exponentially small.

An interactive proof has the prover propose a random $\tilde{G}$, and the verifier selects b. The prover then provides ϕ_b. That the verifier is free to challenge either $b = 1$ or

$b = 2$ means that if the prover consistently returns ϕ_b it must also know the "other" $\phi_{\bar{b}}$, and hence the isomorphism between the graphs.

Theorem 4.1. Graph isomorphism can be proved in zero-knowledge.

Proof: An interactive proof system $\langle P, V \rangle$ decides *ISO* by this algorithm,

(1) The prover chooses a random permutation ϕ of the vertices of G_1 and sends the verifier $H = \phi(G_1)$.
(2) The verifier chooses are random bit b and sends it to the prover.
(3) The prover sends the isomorphism ϕ' such that $H = \phi'(G_b)$.

This is an interactive proof system. If the graphs are isomorphic, the prover can always provide ϕ' and the verifier accepts with probability 1. If the graphs are not isomorphic, no prover can convince the verifier with probability greater than $1/2$, because half the time the H it provided is not isomorphic to the G_b the verifier choose.

To show it is zero-knowledge, we need to show that for the case of isomorphic graphs nothing the verifier experiences in the course of the computation is anything it could not have computed itself. It if does receive knowledge, it does so from the communication with the prover. In this algorithm that communication is a triple drawn from a probability space of triples,

$$X = \{ (H, b, \phi) \mid H = \phi(G_b) \},$$

where H is a graph, G_1 and G_2 are the given graphs, b chosen randomly, and ϕ is a random permutation of vertices on G_b.

We think of the random sampler that delivers one of this triples. In the algorithm, it is drawn by first the Prover providng H, then the Verifier providing b and the the Prover responding with the only correct ϕ that satisfies the constraint for the triples. Because of this solving an instance of ISO, this sampler is NP hard.

However, a sampler with the same distribution could start with a random ϕ and b and set H to satisfy the constraint. This sampler, called *the simulator* can be a PPT.

A simple case is where the verifier flips a fair coin for b. In which case without changing any computation it can let the simulator flip the coin. In this case, that of *honest verifier zero-knowledge*, we have our proof. The Prover-Verifier combination is replaced with this Simulator-Verifier combination for a PPT computation.

The prover might also bias the choice of H (equivalently, ϕ). Since we prove zero-knowledge in the positive case of accepting the input, we can choose our prover, and we choose one that does not do this. The verifier does not have the flip a fair coin. It can react to H is it cares to, thereby shaping the distribution X. The simulator must match this distribution. The formal definition of zero-knowledge does require that the proof hold for any verifier.

The formal definition of zero-knowledge only asks for a proof in the accepting case, and in this case the simulator is working fine. Fortunately, nothing has to be proven in the rejecting case, since in this case the simulator cannot simulate the prover. The prover will recognize that in has been given an impossible task and modify its answers. The simulator however must continue to produce answers the same as always, where $H = \phi(G_b)$, and therefore incapable of rejecting an input. The algorithm will become completely unsound.

4.1. **Cheating Verifiers.** The simulation of X when the protocol is instantiated with an arbitrary V must by necessity use V to mold the distribution. A copy of V is made, that references the same randomness string, and the $x = (H, b, \phi(G_b))$ is tested against the V. Specifically, V is run up to its decision on b, and if it agrees with that in the drawn x, the simulation outputs that x. Else it retries. If at last it never succeeds, it marks its failure outputing the value $\perp$. The repetition limit is set to satisfy an error bound to be determined later.

We claim that this stochastic process of generating a uniform distribution underlying X, then selecting, gives the distribution on X of the original protocol. However, what we mean is when conditioned on $x \neq \perp$. Left to consider is how the case of indeterminacy $x = \perp$ is to be handled.

We can choose our prover in the case of accept, and therefore a verifier must reject on $\perp$. Else a cheating prover can send $\perp$ in place of valid arguments and cause a false accept. This will mean that completeness will not be perfect. That is, with an (exponentially) small probability, isomorphic graphs will be mis-classified.

The non-zero knowledge protocol had perfect completeness. This approach to zero-knowledge sacrifices perfect completeness. And with that I leave you to ponder.