1 Zero-knowledge proofs

In this class of interactive proofs, prover wants to convince verifier that the input is a yes-instance without giving out the solution. For example, if the prover wants to convince the verifier that two graphs are isomorphic, then the following protocol achieves this: 1) prover permutes one of the graphs and sends verifier the permuted graph H
2) verifier sends a number i which can be 1 or 2
3) prover sends the permutation σ of G_i that maps G_i into H.
4) Verifier checks that $\sigma(G_i) = H$ and accepts if so.

Note that in this protocol the verifier never learns the permutation that maps G_1 to G_2. In particular, the interaction between the prover and the verifier always looks like the following: a random permutation sent by the prover, a random bit by the verifier, another random permutation sent back by the prover. The point here is that it is statistically impossible to differentiate between the distribution of any two random permutations and a bit and the real interaction protocols. This is called *perfect statistical zero-knowledge*. It is "perfect" because two distributions are the same (just "statistical" says that they are the same up to some ϵ factor).

An alternative to statistical zero-knowledge is *computational* zero-knowledge. In that case, the distributions are not exactly the same, but no polynomially-bounded statistical test can tell the difference. An example of a computational zero-knowledge protocol would be the following protocol for 3-colourability:
1) prover encodes a colouring of a graph by a hash function and sends it to the verifier.
2) verifier asks the prover for a specific edge
3) prover sends the keys that open exactly that edge in the encoding he sent earlier.

2 Probabilistically Checkable Proofs

3 PCP theorem

To talk about PCPs, we need the definition of a probabilistic polytime verifier V that is $(r(n), q(n))$-*restricted*, i.e., on input of size n,

1. V uses at most $O(r(n))$ random bits, and
2. V queries at most $O(q(n))$ bits of the proof string.

We say that $L \in \mathsf{PCP}[r(n), q(n)]$ if there is a probabilistic polytime $(r(n), q(n))$-restricted verifier V such that

[1] This lecture is a modification of notes by Valentine Kabanets

1. (**completeness**) $x \in L \Rightarrow \exists \pi_x \ \mathbf{Pr}[V^{\pi_x}(x) = 1] = 1$,

2. (**soundness**) $x \notin L \Rightarrow \forall \pi \ \mathbf{Pr}[V^{\pi}(x) = 1] \leq 1/2$.

Here, the probabilities are taken over the random choices of the verifier, and the notation V^{π} means that the verifier V has random access to the string π, i.e., V may request to see the ith bit of the string π, for any i.

We now state the famous PCP theorem, which is one of the deepest results in theoretical computer science in the last decade.

Theorem 1 (PCP Theorem)**.** $\mathsf{NP} = \mathsf{PCP}[\log n, 1]$.

The PCP Theorem was proved by Arora, Safra, and Arora, Lund, Motwani, Sudan, Szegedy in two papers that in turn built upon rich body of work on interactive proof systems done earlier by many researchers. Recently, the authors of these two papers have been honored with the Gödel prize for their contribution to computer science.

4 (Non)-approximability

5 Hardness of Approximation

Recall that a Vertex Cover Minimization problem is: Given a graph $G = (V, E)$ find a smallest-size set $S \subseteq V$ that covers all the edges of G (i.e., every edge in G shares at least one of its endpoints with the set S). We know that the decision version of Vertex Cover is NP-complete. Hence, the Minimization version is NP-hard. Assuming that $\mathsf{P} \neq \mathsf{NP}$, this means that there is no polytime algorithm that finds an optimum-size vertex cover in any given graph.

If we can't hope to find a smallest vertex cover, perhaps we can efficiently find a set cover which is "almost" optimal. We say that the Vertex Cover Minimization problem can be efficiently *c-approximated*, for some $c \geq 1$, if there is a polytime algorithm that, on any input graph G, returns a vertex cover S of G such that $|S_{opt}| \leq |S| \leq c|S_{opt}|$, where S_{opt} is a smallest-size vertex cover for G.

For some c, such an approximation algorithm exists.

Theorem 2. *VC is efficiently 2-approximable.*

Proof. Let $G = (V, E)$ be any input graph. Here's the algorithm for finding a vertex cover S of G.

Initially set $S = \emptyset$. Until there are no edges left, repeat the following: Take any edge $e = \{u, v\}$. Add vertices u and v to S. Remove e from G, as well as remove all edges that touch e (at u or v).

Clearly, the described algorithm runs in polytime, and produces a vertex cover S. Next we argue that $|S| \leq 2|S_{opt}|$. To this end, observe that the set S corresponds to (the endpoints of) a set of mutually disjoint edges of G. Any vertex cover of G (including an optimal S_{opt})

must cover each of these $|S|/2$ edges; so every vertex cover must contain at least one vertex for each of these edges. This means that $|S_{opt}| \geq |S|/2$, as promised. □

Now that we have a 2-approximating algorithm for VC, we may ask for a 3/2-approximating algorithm, or more generally, for a $(1+\epsilon)$-approximating algorithm for any $\epsilon > 0$. If such an algorithm exists, then it's almost as good as being able to solve VC optimally. Does it exists???

It turns out that the answer is "No, unless $\mathsf{P} = \mathsf{NP}$". In other words, it is NP-hard to approximate VC to within some constant $1 < \alpha \leq 2$. Note that it is hopeless to try to show *unconditionally* that VC cannot be γ-approximated. This is because if $\mathsf{P} = \mathsf{NP}$, then certainly VC can be 1-approximated. Thus, proving that VC is NP-hard to approximate is the next best thing.

Remark This can be viewed as a generalization of NP-hardness from exact optimization problems to approximation problems.

The proof of this result follows from the PCP theorem.