In an optimization problem we seek to find a solution that is the best according to some metric. Many NP search problems have natural optimization variants, for example:

- MAX-MATCH: Given a graph G find a matching of maximum size (with the largest possible number of edges)
- MAX-3SAT: Given a formula ϕ, find an assignment that satisfies as many clauses as possible.
- MAX-CLIQUE: Given a graph G find the largest clique in it.
- MAX-LIN: Given a system of linear equations modulo 2 find a solution that satisfies as many equations as possible.

How hard are these problems? It turns out that MAX-MATCH can be solved efficiently using Edmonds' algorithm from the 1950s. MAX-3SAT, on the other hand, couldn't be solvable efficiently unless P = NP because solving it would allow us to in particular tell if a CNF is satisfiable. MAX-CLIQUE is NP-hard for the same reason. From these examples it looks like optimization problems are as hard or easy as their decision counterparts.

Now think about MAX-LIN. Systems of linear equations can be *solved* efficiently by Gaussian elimination so its decision variant LIN (given a system decide if it has a solution) is in P. However the system might be unsatisfiable (it might have equations that are inconsistent) and it is unclear that Gaussian elimination can find a good solution in that case.

Sometimes finding the best possible solution is not needed; one that is good enough will do. For the MAX-LIN problem, a random solution will satisfy half the equations. Is it possible to do any better?

Complexity theory is about not getting what you want, and to give a negative answer to this type of question it is more natural to talk about a related partial decision or search problem. Recall that a *promise problem* is specified by a pair of disjoint subsets $YES, NO \subseteq \{0,1\}^*$, not necessarily partitioning $\{0,1\}^*$. An algorithm solves the problem if it answers yes and no on the YES and NO instances, respectively; there is no correctness requirement for the remaining ones.

Any candidate solution of a maximization problem has a value: the number of edges in the matching, the number of clauses satisfied by the assignment, and so on. It is common to normalize this value so it is a number between 0 and 1 (for matchings the normalization is the half the number of vertices, for formulas it is the number of clauses, and so on.) In the (c,s)-gapped version of the problem, where $c \geq s$, the YES instances are those that have a solution of value at least c, and the NO instances are those in which every solution has value less than s.

An approximate optimization algorithm is one that given an instance that has a solution of value at least c outputs one of potentially smaller value. If this value is guaranteed to be at least s then in particular the algorithm solves the (c,s)-gapped version of the problem. Thus (α,α)-GAP-MATCH is solvable in polynomial time for every δ, as is, say, $(2/3, 1/2)$-GAP-LIN. On the flip side, to argue that no good approximation algorithm exists for a given problem f, it is enough to show that (c,s)-GAP-f is hard. The key to this is an alternative characterization of the class NP, which we discuss next.

1 Probabilistically checkable proofs

Recall that NP consists of those problems whose solutions can be efficiently described and verified. For this lecture it is helpful to think of NP from the perspective of the verification procedure, i.e., the verifier V. For a fixed input x, the verifier receives a candidate witness or *proof* π and has to check that π is valid.

For instance, the verifier for 3SAT interprets the input π (with n variables and m clauses) as a candidate satisfying assignment for ϕ in $\{0,1\}^n$ and checks that each clause $C_i(x_a, x_b, x_c)$, $1 \leq i \leq m$ of π is satisfied by looking at the corresponding three values π_a, π_b, π_c and checking that $C_i(\pi_a, \pi_b, \pi_c)$ is true.

Consider now a lazy implementation of the verifier V that, instead of looking at all clauses, only checks $C_i(\pi_a, \pi_b, \pi_c)$ is true for a single *random* clause i. If π is a satisfying assignment, this verifier accepts with probability 1. If ϕ is unsatisfiable, however, then no matter what π is there is at least a 1 in m chance that the lazy verifier will detect this:

$$\begin{aligned} &\text{if } x \in \text{SAT}, && \text{then for some } \pi,\ V(x,\pi) = 1 \text{ with probability 1} \\ &\text{if } x \notin \text{SAT}, && \text{then for all } \pi,\ V(x,\pi) = 1 \text{ with probability at most } 1 - 1/m. \end{aligned}$$

On the negative side, this verifier has a very high chance of being fooled (when m is large). On the positive side, V is extremely efficient: It only needs to look at three bits of the proof π and verify that the corresponding constraint is satisfied.

To summarize, there are two ways to verify a solution to 3SAT: One in which the verification is completely sound (the verifier never fails) but requires looking at the whole proof, and another one in which the verifier looks at only three bits of the proof but is rarely sound. Is it possible to get the best of both worlds – a trustworthy verifier that queries a small number of bits in the proof?

To formulate this question, we need to augment the power of an NP verifier with two abilities: Randomness and query (oracle) access to the candidate proof. We can view the verifier for a given input x as a randomized oracle circuit that, given oracle access to a candidate proof π, decides if the proof is convincing or not.

Definition 1. A decision problem (YES, NO) has a *probabilistically checkable proof (PCP)* if for every instance x there exists a randomized oracle circuit V_x computable in time polynomial in x that, when given non-adaptive oracle access to a candidate proof $\pi \in \{0,1\}^*$,

$$\begin{aligned} &\text{If } x \in YES, && \text{then } V_x^\pi \text{ accepts with probability 1 for some } \pi, \\ &\text{If } x \in NO, && \text{then } V_x^\pi \text{ accepts with probability at most } 1/2 \text{ for all } \pi. \end{aligned}$$

By *non-adaptive* we mean that the verifier V makes all its queries into π at once; that is, for every q, the location of the qth query cannot depend on the answer to the previous queries. The following two quantities are particularly relevant in this context:

- **Randomness complexity:** The number of random bits (i.e., the portion of the randomness tape) the verifier uses on an input of length n
- **Query complexity:** The maximum number of locations in the proof the verifier queries on an input of length n (for any setting of the randomness tape)

There is no restriction of the length of π. This is because the length is implicitly determined by the randomness complexity and query complexity. Over all possible choices of the randomness, the verifier can access at most $q(n) \cdot 2^{r(n)}$ bits in π, so this is the effective length of π.

The most useful setting of parameters is to set the randomness complexity $r(n)$ to $O(\log n)$ and the query complexity $q(n)$ to a constant. The class of decision problems that have such proofs is called PCP. In this setting, the effective length of the proof becomes polynomial in the length of the input, and a randomized verifier can be simulated by a determnistic one that tries all possible choices of the randomness, and accepts iff all of them yield accepting computations. The new verifier works just like a regular NP verifier, thus PCP is contained in NP.

What about the opposite direction? Let's look at an example, say SAT. A PCP for SAT would be a procedure that, on input ϕ, gets oracle access to a purported (polynomial-length) proof that ϕ is satisfiable, gets to examine this proof at a *constant* number of places, and based on this has to say whether ϕ is satisfiable or not. We expect it to be correct most of the time. Can this be possible?

Theorem 2 (The PCP Theorem)**.** NP *is contained in* PCP*.*

What this says is that any standard NP proof can be rewritten as a PCP. Formally verifying a mathematical proof is an NP problem: The verifier simply checks that every step is consistent with the rules of logic. If we write this proof in a special PCP form, it is possible to verify its authenticity with confidence, say, 50% just by looking at a constant number of bits inside the proof. Even this constant is very reasonable: examining 4 locations suffices. If we want to have confidence say 99% we just repeat the random verification process 7 times.

2 Hardness of approximation

Probabilistically checkable proofs for NP are closely related to approximation algorithms for optimization problems. For example, consider the problem $(1, 1-\varepsilon)$-GAP-3SAT about finding an assignment that satisfies a $(1-\varepsilon)$-fraction of the clauses in a satisfiable formula:

Input: A boolean formula ϕ.
Yes instances: ϕ is satisfiable.
No instances: Every assignment satisfies fewer than a $(1-\varepsilon)$-fraction of all the clauses in ϕ.

A random assignment will satisfy a 7/8-fraction of the clauses assuming each contains exactly three distinct literals. Even if there are clauses with fewer literals there is a more complicated efficient algorithm with the same approximation guarantee. Is it possible to do better?

Theorem 3. *For $\varepsilon < 1/8$there is a polynomial-time reduction from* SAT *to* $(1, 1-\varepsilon)$-GAP-3SAT*.*

Therefore finding an *approximate* assignment for 3SAT that satisfies 87.6% of the clauses is as hard as finding one that satisfies all the clauses of a satisifable formula in the worst case.

Proving Theorem 3 requires several tools which we won't have time to cover. However, if you don't mind a suboptimal value of ε, we will derive Theorem 3 from the PCP theorem.

2.1 Constrant satisfaction problems

To explain the connection between probabilistically checkable proofs and hardness of approximation, it will be convenient to work with a generalization of qSAT (qCNF satisfiability) called qCSP, for "q-ary constraint satisfaction problem." Let $x_1, \ldots, x_n$ be variables taking values in $\{0,1\}$. A *q-ary constraint satisfaction problem* (over alphabet $\{0,1\}$) is a collection of *constraints* $\phi_i \colon \{0,1\}^q \to \{0,1\}$, each of which is associated with a sequence of q variables $x_{i_1}, \ldots, x_{i_q}$. So qSAT is a special

case of qCSP where all the constraints are disjunctions of literals; in a qCSP, *any* constraint in q variables is allowed. The problem of inverting the candidate one-way function from Lecture 8 is another example of a constraint satisfaction problem.

The following promise problem GAP-qCSP captures the hardness of finding approximate solutions to q-ary constraint satisfaction problems:

Input: A q-ary constraint satisfaction problem Ψ.
Yes instances: Some assignment satisfies all constraints of Ψ.
No instances: Every assignment satisfies at most half of the constraints in Ψ.

We specialize ε to $1/2$ for convenience. Here is the relation between PCPs and CSPs:

Claim 4. *A decision problem has a PCP with randomness complexity $O(\log n)$ and query complexity q on inputs of size n if and only if it reduces in polynomial time to GAP-qCSP.*

This connection follows more or less directly from the definitions, but it plays a crucial role in many hardness of approximation proofs.

For the forward direction, assume (YES, NO) has a PCP of randomness complexity $O(\log n)$ and query complexity q. Then a given instance z of the decision problem reduces to a CSP Ψ with variables $x_1, \ldots, x_n$ taking value in $\{0, 1\}$, where x_i represents the contents of the ith location in the proof. For each setting of the randomness r of the verifier there is a constraint ψ_r whose variables $x_{r_1}, \ldots, x_{r_q}$ correspond to those places in the proof queried by the verifier. The constraint $\psi(x_{r_1}, \ldots, x_{r_q})$ takes the value 1 only when the verifier would accept if it would have read the bits $x_{r_1}, \ldots, x_{r_q}$ from the proof. There are $2^{O(\log n)} = \text{poly}(n)$ such constraints and each of them can be written down in time polynomial in n: To write down the constraint we run the verifier (with a fixed setting of the randomness) on all possible values of the proof bits it depends on and see when it accepts and when it rejects.

If z is a yes instance, then there is some proof π that makes V accept regardless of the randomness. Then setting x_i to the value of the ith element in π will satisfy all the constraints of Ψ. If z is a no instance, then no matter what π is V rejects at least half the time. Therefore no matter what the variables x_i are set to, at least half of the constraints will not be satisfiable.

For the reverse direction, we are given a polynomial-time reduction from (YES, NO) to GAP-qCSP. Now consider the following PCP verifier V: On input z, run the reduction to produce a CSP Ψ over variables $x_1, \ldots, x_n$. We view the proof π as the sequence of values for $(x_1, \ldots, x_n)$. The verifier chooses a random constraint ψ of Ψ, queries the locations of the proof indexed by the variables in ψ, evaluates ψ on the answers, and accepts iff ψ evaluates to 1. If z is a yes instance then some assignment satisfies all the constraints of Ψ and the corresponding proof will be accepted with probability 1. If z is a no instance then no assignment satisfies more than half the constraints of Ψ, so no matter which proof x^* V is provided it is bound to reject at least half the time.

2.2 Hardness of approximation for 3SAT

Once we have one problem that is hard to approximate – GAP-qCSP – we can get other ones via reductions.

For example, to show that $(1, 1-\varepsilon)$-GAP-3SAT is hard, we can reduce from GAP-qCSP. Given an instance Ψ of GAP-qCSP, we can transform each of its constraints $\psi_i(x_{i_1}, \ldots, x_{i_q})$ into a 3CNF formula ϕ_i by the CSAT to 3SAT reduction from Lecture 7: ϕ_i has variables $x_{i_1}, \ldots, x_{i_q}, y_{i1}, \ldots, y_{is}$, $s = O(2^q)$, such that $(x_{i_1}, \ldots, x_{i_q})$ is a satisfying assignment for ψ_i if and only if there exists a choice

of $y_{i1}, \ldots, y_{is}$ so that $(x_{i_1}, \ldots, x_{i_q}, y_{i1}, \ldots, y_{is})$ satisfies ϕ_i. The resulting formula has at most $32 \cdot 2^q$ clauses.

Now consider the 3CNF instance ϕ obtained by taking a conjunction in all the clauses in all such formulas ϕ_i. Clearly if Ψ has a satisfying assignment, it can be extended to a satisfying assignment for ϕ. We now argue that if no assigmnent satisfies more than half the constraints in Ψ, then no assignment can satisfy more than $1 - 1/64 \cdot 2^q$ fraction of the clauses in ϕ.

We prove the contrapositive. Assume there is an assignment that satisfies a $1 - \varepsilon$-fraction of constraints in ϕ, where $\varepsilon = 1/64 \cdot 2^q$. So at most an ε-fraction of the clauses in ϕ are not satisfied by this assignment z. Since every formula ϕ_i has $32 \cdot 2^q$ clauses, it follows that at most an $2^q \cdot \varepsilon \leq 1/2$ fraction of the formulas ϕ_i are not satisfied by the assignment. Now if a formula ϕ_i is satisfied by z, then the corresponding constraint ψ_i is also satisfied by the x-part of z. So more than half of the ψ_i are simultaneously satisfiable, and therefore Ψ is not a "no" instance of GAP-qCSP.

To summarize, we gave a polynomial-time reduction from a qCSP instance Ψ to a 3CNF ϕ such that

$$\begin{aligned} \text{if } \Psi \text{ is satisfiable,} \quad & \text{then } \phi \text{ is satisfiable} \\ \text{if } \Psi \text{ is } \leq 1/2\text{-satisfiable,} \quad & \text{then } \phi \text{ is } \leq 1 - 1/64 \cdot 2^q\text{-satisfiable.} \end{aligned}$$

Since q is a constant, so is $\varepsilon = 1/64 \cdot 2^q$. By the PCP theorem, we have a polynomial-time reduction from SAT to $(1, 1-\varepsilon)$-GAP-3SAT, and so $(1, 1-\varepsilon)$-GAP-3SAT is NP-hard.

2.3 Hardness of approximation for clique

We now show that assuming the PCP theorem, it is NP-hard to approximate the size of a maximum clique in a graph to within any constant factor. We consider the following promise decision problem $(1, \delta)$-GAP-CLIQUE:

Input: An undirected graph G on n vertices and a number k.
Yes instances: G has a clique of size k.
No instances: G does not have a clique of size δk.

Any potential algorithm that finds a clique of size at least δ times the size of the maximum clique in G in particular solves $(1, \delta)$-GAP-CLIQUE: To distinguish yes from no instances simply test if the size of the clique output by the algorithm has size as least δk.

Theorem 5. *$(1, \delta)$-GAP-CLIQUE is* NP*-hard for every constant $\delta > 0$.*

We first reduce from $(1, 1-\varepsilon)$-GAP-3SAT to $(1, \delta)$-GAP-CLIQUE for $\delta = 1 - \varepsilon$ and then show how to make δ arbitrarily small. This is the same reduction from the proof of Theorem 5 in Lecture 5: This reduction maps satisfiable instances of 3SAT with m clauses into graphs with clique of size m. In the other direction, any clique of size $(1 - \varepsilon)m$ yields partial assignment that satisfies the corresponding clauses of ϕ. You will show how to reduce the constant from $1 - \varepsilon$ to an arbitrarily small constant δ in the homework.

3 A weak PCP theorem

We now prove the following weak version of the PCP theorem:

Theorem 6. *Every* NP *problem has a PCP with randomness complexity polynomial in the input size and constant query complexity.*

This weak PCP theorem is one of the ingredients in the actual proof of the PCP theorem, which we won't provide. To prove Theorem 6 we need to design a PCP with the required parameters for some NP-complete problem. Instead of working with SAT like we usually do, it will be more convenient to choose another NP-complete problem.

We will start by designing a PCP for LIN (which, as we mentioned, is efficiently solvable). Recall that instances of LIN are systems of linear equations modulo 2, like

$$\begin{aligned} x_1 + x_2 + x_5 &= 1 \\ x_1 + x_3 + x_4 + x_7 &= 0 \\ &\vdots \\ x_2 + x_7 &= 0. \end{aligned}$$

The problem is to decide if this system of equations has a solution or not.

Let's start with a standard NP proof for LIN. An NP-proof can consist of the values for all the variables $x_1, \ldots, x_n$. The verifier then checks that these values satisfy all the equations. To read this proof, the verifier needs to observe the values of all n variables, so it must make a total of n queries. The objective is to rewrite the proof in a way that reduces the number of queries to a constant independent of m and n.

Here is the first idea. A system of m equations over n variables modulo 2 can be written as $Ax = b$, where A is an m by n matrix and b is a column vector of length n. (Here the vectors are column vectors.) Now let r be a *random* vector of length m. Consider the products $r^{\mathrm{T}}Ax$ and $r^{\mathrm{T}}b$, which are scalar values in $\{0,1\}$. If $Ax = b$, then $r^{\mathrm{T}}Ax = r^{\mathrm{T}}b$. An easy but important observation is that if *at least one* of the equations fails to hold, namely $Ax \neq b$, then $\Pr[r^{\mathrm{T}}Ax = r^{\mathrm{T}}b] = 1/2$. To see this, notice that $\Pr[r^{\mathrm{T}}Ax = r^{\mathrm{T}}b] = \Pr[r^{\mathrm{T}}(Ax - b) = 0]$. If $Ax - b$ is nonzero, then the product $r^{\mathrm{T}}(Ax - b)$ has equal chance of being 0 and 1, so the probability is exactly $1/2$.

Once we fix r, the expression $r^{\mathrm{T}}Ax$ is just some linear combination of the x_is; so checking that $r^{\mathrm{T}}Ax = r^{\mathrm{T}}b$ amounts to verifying that some linear combination of the x_is adds to the desired value $r^{\mathrm{T}}b$. In the worst case, however, this expression may involve all the variables x_i, which take n queries to read, so it seems like we have not made any progress towards verifying the satisfiability of the LIN system with a constant number of queries.

Now comes the second idea. A classical NP-proof for a LIN instance consists of the values for all the variables $x_1, \ldots, x_n$. In the PCP proof, we will ask not only for the values of the variables, but also for the value of every *linear combination* of them. This is called the *Hadamard encoding* of x: for every $a \in \{0,1\}^n$, there is a bit in the proof (indexed by the string a) which is supposed to give the value of the linear combination $\langle a, x\rangle = a_1x_1 + \cdots + a_nx_n$. Then, to find out the value $r^{\mathrm{T}}Ax$, it is sufficient to query this proof at the location $a = r^{\mathrm{T}}A$ (which is an n bit vector).

So here is the suggested PCP for LIN: On input a LIN system $Ax = b$, the verifier expects a proof $\pi \in \{0,1\}^{2^n}$, where for every $a \in \{0,1\}^n$, $\pi(a)$ is supposed to equal the value $\langle a, x\rangle$. The verifier chooses a random $r \in \{0,1\}^m$, computes $a = r^{\mathrm{T}}A$ and checks that $\pi(a) = r^{\mathrm{T}}b$.

If the system $Ax = b$ is satisfiable and x is a satisfying assignment, then the proof π where $\pi(a) = \langle a, x\rangle$ clearly makes the verifier accept with probability 1. Now we want to claim that if the system is not satisfiable, then the verifier rejects with probability at least $1/2$.

We can try to reason along the following lines: If $Ax \neq b$ for every x, then no matter what x is, we

have that $r^{\mathrm{T}}Ax \neq r^{\mathrm{T}}b$ with probability 1/2. Since $\pi(a) = \langle a, x\rangle = r^{\mathrm{T}}Ax$, the verifier rejects with probability 1/2. However, this reasoning is incomplete. What is the problem?

The issue is that the verifier cannot be sure that $\pi(a) = \langle a, x\rangle$ for every proof location a; the proof may easily claim, say, that $x_1 = 0$, $x_2 = 0$, but $x_1 + x_2 = 1$. If the verifier queries this proof at $x_1 + x_2$, he will not be aware of this inconsistency. So the verifier can be fooled by a proof that "tells it whatever it wants to hear", namely always give a value $\pi(r^{\mathrm{T}}A)$ that equals $r^{\mathrm{T}}b$.

To take care of this issue, the verifier wants to "force" the proof to be of a particular form: Namely, he wants to be sure that for every a, $\pi(a) = \langle a, x\rangle$ for some assignment x. But by our proof of Claim 1 in Lecture 8, to check this condition the verifier needs to query a constant fraction of the whole proof. Our aim is to have the verifier issue a constant *number* of queries.

What the verifier will be able to do instead is ensure that with high probability, $\pi(a) = \langle a, x\rangle$ for *most* (say 90%) values of a. Once this is established, it turns out that the verifier can recover the other (10%) of the values, and thus have (with high probability) *virtual access* to a correct proof of the form $\pi(a) = \langle a, x\rangle$.

3.1 The linearity test

Let us reformulate the problem we have to deal with: Given access to a proof π, which we think of as a string of length 2^n, we want to guarantee (with high probability) that π is of the form $\pi(a) = \langle a, x\rangle$ for some x while making only a constant number of (random) queries into π.

To do this it will be useful to think of π not as a *proof* but as a *function* of a: This is a function f that maps $\{0,1\}^n$ to $\{0,1\}$. We want to check that f is of the form $f(a) = \langle a, x\rangle$ for some x; in other words, f should be a *linear* function over $\mathbb{F}_2$, the field of two elements. In this language, we just argued that a verifier of constant query complexity cannot tell apart linear functions from non-linear functions. What the verifier will be able to do, however, is detect if the function is "far" from linear.

Definition 7. A function $f\colon \{0,1\}^n \to \{0,1\}$ is *δ-close to linear* if $\Pr_{x\sim\{0,1\}^n}[f(x) \neq \langle a, x\rangle] \leq \delta$ for some $a \in \{0,1\}^n$. f is *δ-far from linear* if it is not δ-close to linear.

In the $(1, 1-\delta)$-GAP-LIN promise problem, the YES instances are linear functions and the NO-instances are functions that are δ-far from linear.

Theorem 8. *For $\delta < 1/2$, the one-sided $(1-\delta)$-error randomized query complexity of $(1, 1-\delta)$-GAP-LIN is 3.*

By one-sided we mean that the algorithm functions that are linear with probability 1. The idea behind this "linearity test" is the following dual characterization of linearity. A function $f\colon \{0,1\}^n \to \{0,1\}$ is linear if and only if

$$f(x) + f(y) = f(x + y)$$

for every pair $x, y \in \{0,1\}^n$. The linearity checks that this relation is satisfied for a *random* pair x, y.

The linearity test: On input 1^n, choose $x, y \in \{0,1\}^n$ independently at random, query the oracle at locations x, y and $x + y$ and accept iff $f(x) + f(y) = f(x + y)$.

Clearly, if f is linear, the test accepts with probability 1. We need to show that if f is δ-far from linear, then the test rejects with probability at least δ. We will prove the contrapositive, namely

Claim 9. *If* $\Pr[f(x)+f(y)=f(x+y)] \geq 1-\delta$ *then* f *is* δ*-close to a linear function.*

Proof. Writing $F(x) = (-1)^{f(x)}$, $f(x)+f(y)=f(x+y)$ means $F(x)F(y)F(x+y)=1$ and

$$\mathrm{E}_{x,y\sim\{0,1\}^n}[F(x)F(y)F(x+y)] = \Pr[F(x)F(y)F(x+y)=1] - \Pr[F(x)F(y)F(x+y)=-1] = 1-2\delta.$$

Fourier analysis is often helpful with expressions that look like the one on the left. Let's see what we get:

$$\begin{aligned}
\mathrm{E}[F(x)F(y)F(x+y)] &= \mathrm{E}_{x,y\sim\{0,1\}^n}\Big(\sum_{a\in\{0,1\}^n}\hat{F}_a\chi_a(x)\Big)\Big(\sum_{b\in\{0,1\}^n}\hat{F}_b\chi_b(y)\Big)\Big(\sum_{c\in\{0,1\}^n}\hat{F}_c\chi_c(x+y)\Big) \\
&= \mathrm{E}_{x,y\sim\{0,1\}^n}\sum_{a,b,c\in\{0,1\}^n}\hat{F}_a\hat{F}_b\hat{F}_c\chi_a(x)\chi_b(y)\chi_c(x+y) \\
&= \sum_{a,b,c\in\{0,1\}^n}\hat{F}_a\hat{F}_b\hat{F}_c\cdot\mathrm{E}_{x,y\sim\{0,1\}^n}[\chi_a(x)\chi_b(y)\chi_c(x+y)].
\end{aligned}$$

Recalling that $\chi_a(x) = (-1)^{\langle a,x\rangle}$, we can write

$$\mathrm{E}_{x,y\sim\{0,1\}^n}[\chi_a(x)\chi_b(y)\chi_c(x+y)] = \mathrm{E}_{x,y\sim\{0,1\}^n}[\chi_{a+c}(x)\chi_{b+c}(y)] = \mathrm{E}_{x\sim\{0,1\}^n}[\chi_{a+c}(x)]\,\mathrm{E}_{y\sim\{0,1\}^n}[\chi_{b+c}(y)].$$

This expression equals zero unless $a=c$ and $b=c$, and so

$$\mathrm{E}[F(x)F(y)F(x+y)] = \sum_{\substack{a,b,c\in\{0,1\}^n:\\ a=c \text{ and } b=c}}\hat{F}_a\hat{F}_b\hat{F}_c = \sum_{a\in\{0,1\}^n}\hat{F}_a^3.$$

Since $\sum\hat{F}_a^3 \leq (\max\hat{F}_a)\sum\hat{F}_a^2 = \max\hat{F}_a$, there must exist $a\in\mathbb{F}^n$ such that $\hat{F}_a \geq 1-2\delta$. But $\hat{F}_a = \mathrm{E}[F(x)\chi_a(x)] = \Pr[F(x)=\chi_a(x)] - \Pr[F(x)\neq\chi_a(x)]$, so $\Pr[F(x)=\chi_a(x)] \geq 1-\delta$. □

3.2 A PCP for LIN

We are now almost ready to describe the PCP for LIN. Let's recall what we did so far: Given a system of equations $Ax=b$, we expect a PCP proof of the form $\pi(a) = \langle a,x\rangle$ for all $a\in\{0,1\}^n$. The linearity test (repeated a constant number of times) allows us to verify, with probability at least say 9/10, that π is 1/10-*close* to a proof of this form. If we could tell that π was exactly of this form, then we can finish by choosing a random linear combination r of the equations in the system, query the proof at location $r^{\mathrm{T}}A$, and accept iff the queried value equals $r^{\mathrm{T}}b$. By our previous reasoning, if π encodes a satisfying assignment for the system then the verifier accepts with probability 1 and if it doesn't, it accepts with probability 1/2.

There is one problem left to resolve: The linearity test does not allow us to conclude that π is *exactly* linear, but merely that it is δ-close to linear for a small constant δ of our choice.

However, it turns out that once we know that π is 1/20-close to a linear function ℓ, we can simulate a query into ℓ by making *two* queries into π. The idea is simple: To find the value of $\ell(y)$, we choose a random u and output $\pi(y+u)+\pi(u)$. Since both $y+u$ and u are random, with probability $1-2/10 = 4/5$ we will have $\pi(y+u)=\ell(y+u)$ and $\pi(u)=\ell(u)$, and therefore $\pi(y+u)+\pi(u) = \ell(y+u)+\ell(u) = \ell(y)$. This procedure which extracts the value of $\ell(y)$ out of π (which is a faulty version of ℓ) is called *local correction*, as it allows us to correct the faults in y locally (i.e. with very few queries).

We can now give the full PCP for LIN: On input $Ax+b$ and given oracle access to a proof π, first run the linearity test a constant number of times to ensure that with probability at least 9/10, π is

1/20-close to a linear function ℓ. Then choose a random r and compute the value $a = r^{\mathrm{T}}A$. Now to find $\ell(a)$, choose a random u and query π at locations $a+u$ and u. Accept iff $\pi(a+u)+\pi(u) = r^{\mathrm{T}}b$.

If $Ax = b$ has a solution, then the proof $\pi(a) = \langle a, x\rangle$ will pass the linearity test with probability 1, and $\pi(a+u) + \pi(u) = \langle a, x\rangle$ with probability 1. Since $\langle a, x\rangle = r^{\mathrm{T}}Ax = r^{\mathrm{T}}b$, the verifier accepts with probability one.

If $Ax = b$ does not have a solution, let $\ell(a) = \langle a, x\rangle$ be the closest linear function to the proof π provided to the verifier. By the analysis of the linearity test, with probability 9/10, π and ℓ are 1/10-close. Assume this is the case. Then $r^{\mathrm{T}}Ax \neq r^{\mathrm{T}}b$ with probability 1/2. On the other hand, $\ell(a) = \pi(a+u) + \pi(u)$ with probability at least 4/5. All these events hold with probability $1 - 1/10 - 1/2 - 1/5 \geq 1/10$; if this is the case, then

$$\pi(a+u) + \pi(u) = \ell(a) = \langle a, x\rangle = r^{\mathrm{T}}Ax \neq r^{\mathrm{T}}b$$

so the verifier rejects with probability at least 1/10.

To summarize, if $Ax = b$ is satisfiable, there exists π which makes the verifier accept with probability 1, and if not, then every π makes the verifier reject with probability at least 1/10. We can repeat the PCP a constant number of times to increase the rejection probability to 1/2.

3.3 A PCP for all problems in NP

We now generalize the PCP for LIN into one that works for all of NP. To do so we need an NP-complete problem which looks like LIN. This problem is called QE (for quadratic equations modulo 2). An instance of QE is a system of homogeneous quadratic equations modulo 2 like

$$\begin{aligned} x_1x_3 + x_2x_2 + x_5x_7 &= 1 \\ x_1x_1 + x_1x_3 + x_4x_4 + x_2x_7 &= 0 \\ &\vdots \\ x_2x_2 + x_3x_3 &= 0. \end{aligned}$$

You can show that this problem is NP-complete by reduction from CSAT, in much the same way we proved that SAT is NP-complete.

Now QE looks a lot like LIN, except that the equations are quadratic. Each of these quadratic equations can be viewed as a linear equation over the n^2 variables $y_{ij} = x_ix_j$. The PCP for LIN proves that this system has a solution y_{ij}. It remains to verify that the solution is indeed of the form x_ix_j. (For example the solution $y_{11} = y_{22} = 0$, $y_{12} = y_{21} = 1$ is not of this form.)

To do so, we augment the proof π, which represents the Hadamard encoding of y with another part ρ which gives the Hadamard encoding of x. Namely, for yes instances, the proof (π, rho) will be

$$\begin{aligned} \pi(a) &= \sum_{i,j=1}^{n} a_{ij}y_{ij}, \quad \text{where } y_{ij} = x_ix_j \\ \rho(b) &= \sum_{i=1}^{n} b_ix_i. \end{aligned}$$

Given access to such a ρ, the verifier can check that it is, say, 1/100-close to linear. Once it knows that both π and ρ are close to linear, it can treat them both as linear by using local decoding, so let's assume that they are both linear. It remains to check that they are consistent.

If π and ρ are indeed consistent, then it must be the case that for every $b, c \in \{0,1\}^n$:

$$\pi(b_1c_1, b_1c_2, \ldots, b_nc_n) = \rho(b_1, \ldots, b_n)\rho(c_1, \ldots, c_n) \tag{1}$$

which suggests the following consistency test: Choose b, c at random from $\{0,1\}^n$ and check that (1) is satisfied.

Clearly if ρ and π are consistent linear functions this test will accept with probability 1. We now argue that if ρ and π are linear but inconsistent then the test will reject with probability at least $1/4$. Any two linear functions ρ and π have the form

$$\rho(b_1, \ldots, b_n) = b_1x_1 + \cdots + b_nx_n$$
$$\pi(a_{11}, \ldots, a_{nn}) = a_{11}y_{11} + a_{12}y_{12} + \cdots + a_{nn}y_{nn}$$

for some $x \in \{0,1\}^n, y \in \{0,1\}^{n^2}$. Now suppose that $y_{ij} \neq x_ix_j$ for at least one pair (i,j). Then we claim that with probability at least $1/4$,

$$\rho(b_1, \ldots, b_n)\rho(c_1, \ldots, c_n) \neq \pi(b_1c_1, b_1c_2, \ldots, b_nc_n).$$

To see this, we evaluate both sides:

$$\rho(b_1, \ldots, b_n)\rho(c_1, \ldots, c_n) = \sum_{i,j=1}^{n} b_ic_jx_ix_j$$
$$\pi(b_1c_1, \ldots, b_nc_n) = \sum_{i,j=1}^{n} b_ic_jy_{ij}.$$

Let's think of the difference of the two: For fixed x and y, this is a quadratic polynomial in the variables b_i and c_j. Our assumption is that at least one of the terms x_ix_j and y_{ij} are different, so this polynomial is not identically zero. But when we evaluate a non-zero quadratic polynomial over $\mathbb{F}_2$ at a random input, the probability that it vanishes is at most $1/4$. This follows from a more general form of the DeMillo-Lipton-Schwarz-Zippel Theorem from Lecture 7: Write the quadratic polynomial as

$$\sum_{i=1}^{n} b_i \cdot (\text{some linear function of } c_1, \ldots, c_n).$$

Since the polynomial is non-zero, at least one of these linear functions is non-zero, so with probability $1/2$ it won't vanish after $c_1, \ldots, c_n$ are replaced by random values. After this replacement is done, we just get a non-zero linear function in $b_1, \ldots, b_n$, so with another $1/2$ probability this won't vanish after $b_1, \ldots, b_n$ are replaced by random values.

To summarize, we have the following PCP for QE. The proof is of the form (π, ρ), where π has length 2^{n^2} and ρ has length 2^n. We expect ρ and π to encode some satisfying assignment x to the system of equations. To check this is the case, we run the following tests say 10 times:

1. **Linearity test:** Run the linearity test a constant number of times on π and ρ. This ensures that with probability 99%, both π and ρ are 1%-close to linear functions.
2. **Consistency test:** Choose random $b, c \in \{0,1\}^n$ and check that

   $$\tilde{\rho}(b_1, \ldots, b_n)\tilde{\rho}(c_1, \ldots, c_n) = \tilde{\pi}(b_1c_1, b_1c_2, \ldots, b_nc_n)$$

   where $\tilde{\rho}(b) = \rho(b+u) + \rho(u)$ for a random u, and similarly for $\tilde{\pi}$.

3. **Satisfiability test:** Choose random $r \in \{0,1\}^m$. Take a linear combination of the equations as indexed by r and let a denote the vector of coefficients. Accept if $\tilde{\pi}(a) = r^{\mathrm{T}} b$.

If the QE instance is satisfiable, then π and ρ that encode a satisfying assignment make the verifier accept with probability 1. If not, then no matter what π and ρ are, the verifier rejects with constant probability by an extension of the analysis we did for LIN.

References

The PCP Theorem was proved in 1993 by Arora, Lund, Motwani, Sudan, and Szegedy, who built upon a breakthrough of Arora and Safra from 1992. In 2005 Irit Dinur came up with a completely different proof. Theorem 3 was proved by Håstad. The constant 1/8 in this theorem is optimal; Zwick showed that 1/8-GAP-3SAT can be solved in polynomial time. Much stronger inapproximability results for clique are known; the best one was also obtained by Håstad.