Around the **PCP** Theorem

Sanjeev Arora
Princeton University

Technical Report SOCS-97.2
School of Computer Science
McGill University

Foreword

The 1996 McGill Invitational Workshop on Computational Complexity Theory was held at McGill University's Bellairs Research Institute in Barbados, from February 25 to March 2.

This year workshop was devoted to the **PCP** theorem and related material: Sanjeev Arora, from Princeton, was the guest speaker. The participants thus had a unique occasion to hear about what is arguably the most spectacular result in the history of computational complexity from someone who was deeply involved in its development.

Many thanks to the note-takers, and very, very special thank you to Sanjeev who was so generous with his time and energy.

Denis Thérien

Contents

McGill/Barbados Workshop'96	Randomness in Complexity Theory
Lecture 1: Impossibility and Possibility Results	
Lecturer: *Sanjeev Arora*	Notes: *Uwe Schöning*

One of the recurring themes in these lectures is a duality between *possibility* and *impossibility* results in Complexity Theory. The former are algorithmic and thus give *upper bounds* on complexity; the latter give *lower bounds*. Sometimes an upper bound leads (unexpectedly?) to a lower bound. One example is the PCP theorem, which we will talk about later. The result that NP $\subseteq$ PCP$(\log, 1)$ can be considered as possibility result: membership proofs for SAT can be checked much more efficiently than previously believed. On the other hand (assuming P $\neq$ NP) this can also be viewed as an impossibility result: it implies (as we will see) that polynomial-time algorithms cannot compute good approximate solutions to many optimization problems.

Another recurring theme in these lectures is the use of randomness, which is often used to prove results that make no mention of randomness.

With those two remarks in mind, let's start.

1 Random Self-Reducibility

Random self-reducibility (r.s.r.) is a classic example of the duality betweem impossibility and possibility results. We illustrate this concept using the discrete log problem (cf. [BlMi]). Let p be a prime and g be a generator of $\mathbb{Z}_p^*$, that is, an element such that such that $\forall y \in \mathbb{Z}_p^*\ \exists x : g^x \equiv y \pmod{p}$. Then we call x the 'discrete log' of y (usually p and g will be clear from the context). We note that the generator g is easy to find if we know the prime factorization of $p-1$: g is a primitive root iff $a^{(p-1)/q} \not\equiv 1 \pmod{p}$ for all prime factors q of $p-1$.

The discrete log problem is: given a prime p and a generator g, and a point y chosen at random from $\mathbb{Z}_p^*$, find the logarithm x.

Claim: *Suppose some deterministic polynomial-time algorithm A solves discrete log correctly for $1/poly(n)$ fraction of $y \in \mathbb{Z}_p^*$ ($n = |p|$ is the size of the input). Then there is a randomized polynomial-time algorithm A' that w.h.p. solves discrete log at all points in $\mathbb{Z}_p^*$.*

We note that the claim shows that a weak "impossibility" result for the discrete log implies a strong "impossibility" result.

The claim is proved by observing that a worst-case instance discrete log is reducible to a random instance. Given a $y \in \mathbb{Z}_p^*$, pick $r \in \mathbb{Z}_p^*$ at random and compute $yg^r \pmod{p}$. Note that yg^r is uniformly distributed in $\mathbb{Z}_p^*$, so with probability $1/poly(n)$ this point is "easy" for algorithm A. Suppose A gives us a $\hat{x}$ such that $g^{\hat{x}} = yg^r$. Note that we can check whether or not $\hat{x}$ is a correct answer by using the modular exponentiation algorithm [CLR]. Assuming $\hat{x}$ is a correct answer is indeed correct, $y = g^{\hat{x}-r}$, and we output $\hat{x} - r$ as the answer.

Finally, try random strings $r_1, r_2, \ldots, r_{p(n)}$ and run algorithm A on yg^{r_i}. If it ever produces a *right* answer $\hat{x_i}$ with $g^{\hat{x_i}} = yg^{r_i}$ then the desired y is g^{x-r_i}. W.h.p., one of the trials will succeed.

We remark that some *algebraic structure* often underlies random self-reducibility.

2 Randomness in Complexity: The early 1980s

Now we will answer several questions along the lines of: what power does randomness buy us? For example, can randomized polynomial-time algorithms solve SAT? Can we randomly reduce NP to $\overline{SAT}$ or coNP to SAT? As we will see, the answer to these questions is: "Probably not."

One of the pathbreaking results in this area was Sipser's result that BPP lies in the polynomial hierarchy.

First recall the definition of BPP. L is in BPP iff there is a deterministic, polynomial-time machine M such that in case of $x \in L$,

$$Pr_r[M(x,r) = acc] \geq 2/3$$

and in case of $x \notin L$,

$$Pr_r[M(x,r) = acc] \leq 1/3$$

Here Pr_r means the probability when the string r is randomly picked among all strings of size $p(n)$, $n = |x|$, with some suitable polynomial size p.

Actually the constants $2/3$ and $1/3$ can be substituted by, say, $1 - 2^{-(n+1)}$ and $2^{-(n+1)}$. It is not even necessary to use the whole machinery of Chernoff bounds [Che] to prove this. Instead of one random string r now pick t many random strings $r_1, \ldots, r_t$ independently. Accept if and only if the majority of the $M(x, r_i)$ computations would accept. Now the probability that this machine gives a wrong answer is at most

$$\begin{aligned}
\sum_{i=0}^{t/2} \binom{t}{i} (2/3)^i (1/3)^{t-i} &\leq \sum_{i=0}^{t/2} \binom{t}{i} (2/3)^i (1/3)^{t-i} \left(\frac{2/3}{1/3}\right)^{t/2-i} \\
&= \sum_{i=0}^{t/2} \binom{t}{i} (2/3)^{t/2} (1/3)^{t/2} \\
&= (2/9)^{t/2} \cdot \sum_{i=0}^{t/2} \binom{t}{i} \\
&\leq (2/9)^{t/2} \cdot 2^t \; = \; (2\sqrt{2}/3)^t
\end{aligned}$$

Choosing $t = 12n$ gives (for large enough n) that this expression stays below the desired probability $2^{-(n+1)}$.

¿From now on we will assume that BPP is defined using a probability $2^{-(n+1)}$ instead of $1/3$.

Recall that the class P/poly consists of all languages with (nonuniform) polynomial size circuits. Equivalently, it is the class of languages which can be decided in polynomial-time using some "advice string" of polynomial size.

■ (Bennett, Gill [BeGi]) BPP $\subseteq$ P/poly.

Proof: Assume the probabilities in the defn of BPP have been "amplified" as above. We wish to identify an "advice string" for inputs of size n. Let x be an input of size n and suppose the BPP machine uses $r(n)$ random bits on it. Among all $2^{r(n)}$ strings of size $r(n)$, only a fraction $2^{-(n+1)}$ cause the machine to give an incorrect answer on x. Summing this fraction over all inputs of size n, we conclude that at most $2^n \cdot 2^{-(n+1)} = 1/2$ fraction of random strings cause the machine to give an incorrect answer on *any* input of size n. Hence at least $1 - 1/2$ fraction of random strings cause the machine to answer correctly for *all* inputs of size n. Take any one of these strings and use it as the advice string. □

Now we prove a classic result, one of whose implications is that NP is unlikely to be in BPP.

■ (Karp, Lipton [KaLi]) NP $\subseteq$ P/poly $\Rightarrow \forall i : \Sigma_i^p \subseteq$ P/poly.

Proof: By induction. The base case $i = 1$ is the assumption. Now suppose $\Sigma_i^p \subseteq$ P/poly. Then also $\Pi_i^p \subseteq$ P/poly (since P/poly is closed under complement). Let $A \in \Sigma_{i+1}^p$. Then $A = \{x \mid \exists y : (x, y) \in B\}$ for a set $B \in \Pi_i^p \subseteq$ P/poly. That is, there is a sequence of polynomial-length strings (the advices) $w_1, w_2, \ldots$ such that $B = \{(x, y) \mid (x, y, w_{p(n)}) \in C\}$ for a set $C \in$ P and a polynomial p. Put together, $A = \{x \mid (x, w_{p(n)}) \in D\}$ where $D = \{(x, w) \mid \exists y : (x, y, w) \in C\}$ is an NP set. Therefore, by the assumption, $A \in$ NP/poly $\subseteq$ (P/poly)/poly = P/poly. This completes the induction. □

■ (Karp, Lipton [KaLi]) NP $\subseteq$ P/poly $\Rightarrow \forall i : \Sigma_i^p = \Sigma_2^P$.

Proof (somewhat simplified, following [Hop,ArKoScSc]): By elementary results about the polynomial-time hierarchy [BaSe], for showing a collapse of the polynomial-time hierarchy to Σ_2^P, it suffices to prove $\Pi_2^P \subseteq \Sigma_2^P$. Next, observe, under the assumption that SAT is in P/poly, using SAT's self-reducibility structure [Bal], there is a polynomial-time computable function h such that

$$x \in SAT \;\Rightarrow\; h(x, w) = \text{a satisfying assignment for } x$$

Here w is a polynomial-length advice string, depending only on the length of x.

Now let $L \in \Pi_2^P$. Then $L = \{x \mid \forall y : (x, y) \in A\}$ for a set $A \in$ NP. Since SAT is NP-complete, there is a reduction f from A to SAT. Therefore,

$$\begin{aligned} L &= \{x \mid \forall y : (x, y) \in A\} \\ &= \{x \mid \forall y : f(x, y) \in SAT\} \\ &= \{x \mid \exists w \forall y : h(f(x, y), w) \text{ is a sat. assignment for } f(x, y)\} \end{aligned}$$

In this characterization, y is a polynomial-length string. Therefore, A can be expressed by Σ_2^P predicate. It follows $\Pi_2^P \subseteq \Sigma_2^P$. □

By a similar proof the following can be obtained.

■ (Yap [Yap]) co-NP $\subseteq$ NP/poly $\Rightarrow$ PH $= \Sigma_3^P$.

Now we prove Sipser's result that BPP is in the polynomial hierarchy. Similar arguments have been used in other results; these results all have the flavor of "derandomizing" a complexity class at the cost of adding 2 quantifiers.

■ (Sipser [Sip], Gacs[]) BPP $\subseteq \Sigma_2^P \cap \Pi_2^P$.

Proof: (The proof argument given here follows Lautemann [Lau]). Let M be a BPP machine for language L. Let x be an input of size n and suppose the machine uses m random bits for this input. Let S be the set of strings in $\{0,1\}^m$ that makes M accept x. We know that either S is "almost everything" or "almost nothing."

Now we define a "translation" of set S. If $t \in \{0,1\}^m$, we define $t \oplus S$ as the set $\{t \oplus r : r \in S\}$, where $t \oplus r$ is the bitwise xor of the strings t and r.

Observe that for any translation t, $|S| = |t \oplus S|$. This follows from the fact that $r \neq r' \Rightarrow t \oplus r \neq t \oplus r'$. Also observe that if r is fixed and $t \in \{0,1\}^m$ is a random tranlation, then $t \oplus r$ is uniformly distributed over $\{0,1\}^n$.

Now consider the case when S is "big": $\mu(S) = |S|/2^m \geq 1 - 2^{-n}$. We claim that there is a set of transformations $t_0, \ldots, t_{m/n}$ such that

$$\bigcup_{i=0}^{m/n} (t_i \oplus S) = \{0,1\}^m$$

We show this by the probabilistic method. That is, pick the t_i's independently at random. Look at any fixed point $r \in \{0,1\}^m$. Then

$$\Pr[r \notin t_i \oplus S] \leq \frac{1}{2^n}$$

Therefore, by independence,

$$\Pr[r \notin \bigcup_i t_i \oplus S] \leq \left(\frac{1}{2^n}\right)^{1+m/n} = \frac{1}{2^{m+n}}$$

Since there are only 2^m strings rf,

$$\Pr[\bigcup_i t_i \oplus S \neq \{0,1\}^m] \leq \frac{2^m}{2^{m+n}} < 1.$$

Hence there exists a sequence of translations $t_1, t_2, \ldots, t_{m/n}$ such that $\cup_{i=0}^{m/n}(t_i \oplus S) = \{0,1\}^m$.

On the other hand, consider what happens when S is "small": i.e. $\mu(S) \leq 2^{-n}$. Then we get

$$|\bigcup_{i=0}^{m/n} t_i \oplus S| \leq \sum_{i=0}^{m/n} |t_i \oplus S| = \sum_{i=0}^{m/n} |S| = (1 + m/n)|S| \ll 2^m$$

So we can express membership in a BPP-set L as follows

$$x \in L \Leftrightarrow \exists t_0, \ldots, t_{m/n} \, \forall r \, [M(x, t_0 \oplus r)=\text{acc} \bigvee \ldots \bigvee M(x, t_{m/n} \oplus r)=\text{acc}]$$

This is a Σ_2^P characterization of BPP. Since BPP is closed under complement, it must also be included in Π_2^P. □

Now we give an extension of Sipser's result. Recall the class BP·NP. A language L is in this class if there is a $L_1 \in$ NP such that

$$\begin{aligned} x \in L &\Rightarrow Pr_r[(x, r) \in L_1] \geq 2/3 \\ x \notin L &\Rightarrow Pr_r[(x, r) \in L_1] \leq 1/3 \end{aligned}$$

It turns out that BP·NP is the same as Babai's class AM[2] (AM stands for Arthur-Merlin game [Bab]). Reasoning along the same lines as the proof that BPP $\subseteq$ P/poly, we can show that BP·NP $\subseteq$ NP/poly.

Reasoning along the same lines as Sipser, one also gets:

■ (Babai [Bab]) BP·NP $\subseteq \Pi_2^P$.

(Note that BP·NP is not known to be closed under complement).

Now we show that it is unlikely that $\overline{SAT}$ (i.e., co-NP) can be randomly reduced to SAT. Suppose $\overline{SAT}$ reduces to SAT using a randomized reduction. Then coNP $\subseteq$ BP·NP $\subseteq$ NP/poly. Yap's result implies that PH $= \Sigma_3^P$. (Actually this can be improved to a collapse to PH$=\Sigma_2^P=$BP·NP by applying the above techniques directly; see [BoHaZa,Sch,KST].)

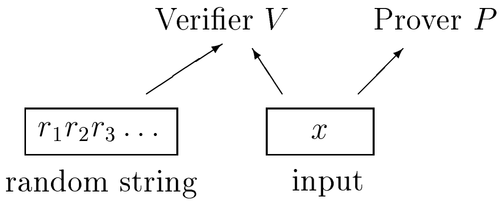
3 Interactive Proofs

Now we move to *interactive proofs*, a notion that gives rise to a rich variety of complexity classes. One of the main results we will prove is that the graph isomorphism problem is probably not NP-complete (i.e., if PH doesn't collapse). Graph isomorphism was one of the 12 problems in Garey and Johnson's book whose status was open in 1979.

Another nice result in this section is a "Set Lowerbound" protocol that allows a prover to prove to a polynomial-time verifier that a mutually-agreed upon set is "large."

First we define the class AM[2]: it is a special case of a couple of classes that can be defined in terms of an (interactive) "Prover" and a "Verifier". Using these, we get classes with a new flavor of alternation. The model is this (due to [GoMiRa]):

V is actually nothing else than a polynomial-time computable function q. A protocol between P and V proceeds as follows. Both P and V can read the input x. Additionally V has random strings r_i available on which V can base its computation. First V computes the function value $q(x\#r_1)$ and communicates it to P. P sends



some answer a_1 back (that depends on x and the communicated value of q), then V computes $q(x\#r_1\#a_1\#r_2)$, and sends it to P again, and so on. Finally, V will accept or reject (i.e. the result of the respective evaluation of q is "accept" or "reject").

If the input string is in the language under consideration, then there should exists an answer strategy for P such that V will finally accept. On the other hand, if the input is not in the language, then no matter what P answers V should reject with high probability.

There are several possible variations of the definition. Apart from the requirement that q is computable in polynomial time, P's answers should have polynomial length, and further the number of rounds until V accepts or rejects should be polynomial in the input length. This class is then called IP (for *interactive proof*).

As defined above the random strings r_i are *private* to V, i.e. the r_i's are invisible to P since P can base its answers only on the q-values. If P is allowed to look at the r_i's then this is the *public coin* model (which coincides with Babai's definition of AM [Bab]). ¿From Shamir's proof [Sha] of IP=PSPACE it actually can be seen that AM=IP.

Another variation on the definiton involves restricting the number of rounds to a constant. Denote by AM$[i]$ and IP$[i]$ the class of languages which can be expressed this way in the public coin model (private coin model, resp.) where the number of rounds is at most i. Babai and Moran [BaMo] show that the public coin interactive proof hierarchy (called Arthur-Merlin hierarchy) collapses, i.e. $\bigcup_{i\geq 1} \mathrm{AM}[i] = \mathrm{AM}[2]$.

The following is a simple private-coin 2-round protocol for Graph Non-Isomorphism. Notice that $\overline{ISO}$ is not known to be in NP, but IP[2] might be considered as only a "small" extension of NP.

■ (Goldreich, Micali, Wigderson [GoMiWi]) $\overline{ISO} \in \mathrm{IP}[2]$).

Proof: Given the input graphs G_1, G_2 on n vertices, the Verifier V picks a random permutation $\pi \in S_n$ and a random bit $b \in \{1, 2\}$. V computes the permuted graph $H = \pi(G_b)$ and sends H to the Prover P. The Prover is now supposed to answer with one bit a which should be equal to b. This is the condition for V to accept, otherwise V rejects.

If the two input graphs are not isomorphic, then P can always answer correctly since H can be uniquely identified with one of the two graphs. (Recall that P has infinite computational power.) But if the input graphs are isomorphic, then H could be obtained from G_1 and from G_2. Therefore, without knowledge of b (private coins!) it is impossible for P to find out the correct value of b. The best P can do is to select

a uniformly at random which gives a probability of error (respectively success) of 1/2. $\square$

Private coins seem to be crucial here to make the protocol work. But, surprisingly, it can be shown that both models, private coins and public coins, are equivalent. This is because of a public-coin protocol due to Goldwasser and Sipser, which we will use more than once.

3.1 Goldwasser Sipser Set-lowerbound protocol

First we define universal hash functions. According to [CaWe] a family $\mathcal{H}$ of hash functions $h : U \to W$ is *2-universal* if for any $x, y \in U$, $x \neq y$, and $u, v \in W$, it holds $\Pr_{h\in\mathcal{H}}(h(x) = u \text{ and } h(y) = v) = 1/|W|^2$.

Suppose the verifier is a machine that runs in poly(n) time, and suppose $S \subseteq \{0,1\}^n$ is a set in which membership can be checked in poly(n) time. The GS protocol is a two-round (public-coin) protocol by which the prover can prove to the verifier that the size of S is at least $|S|/2$. (By this we mean that the prover announces a number K which is supposedly $|S|$. The verifier then initiates an interrogation which the prover can pass with high probability if $|S| > K$, and fails with high probability if $|S| \leq K/2$.)

First we replace S by $S' = S \times S \times S \times S$. Note that membership in S' is also decidable in poly(n) time. Now the prover has to show that S' has size at least $|S'|/16$.

Suppose the prover's claim is that $|S'| = K$.
Round 1: The verifier picks a random hash function $h\colon \{0,1\}^{4n} \to \{0,1\}^{\log K-1}$ from a universal family. Then she picks a random $y \in \{0,1\}^{\log K-1}$. She sends h, y to the prover.

Round 2: The prover sends a $x \in \{0,1\}^{4n}$ such that $y \in S'$ and $h(x) = y$.
The verifier accepts if the prover was able to send such an x.

The main observation is that if $|S'| < K/16$, then $|h(S')| < K/16$, so at most $K/16(2^{\log K-1}) = 1/8$ fraction of $y \in \{0,1\}^{\log K-1}$ have preimages in S. Hence the prover has only a probability 1/8 of satisfying the verifier.

Now suppose $|S'| \geq K$. The following claim shows that the prover can satisfy the verifier with probability at least 1/2.

Claim: Let $\mathcal{H}$ be a 2-universal family of hash functions from $\{0,1\}^{4n}$ to $\{0,1\}^m$ and $T \subseteq \{0,1\}^{4n}$ has size 2^m. Then for any $y \in \{0,1\}^m$,

$$\Pr_{h\in\mathcal{H}}[\exists x \in T \;:\; h(x) = y] \geq \frac{1}{2}.$$

Proof of the Claim: By the Inclusion-Exclusion Principle and by the 2-universality of

$\mathcal{H}$ we have

$$\begin{aligned}
\Pr_h(\exists x \in T : h(x) = y) &\geq \sum_{x \in T} \Pr(h(x) = y) \\
&\quad - \sum_{\substack{x, x' \in T \\ x \neq x'}} \Pr_h(h(x) = h(x') = y) \\
&= \frac{|T|}{2^m} - \frac{\binom{|T|}{2}}{2^{2m}} \\
&\geq 1 - 1/2 \; = \; 1/2.
\end{aligned}$$

Finally, we note that the probabilities 1/2 versus 1/8 can be amplified by repeating the protocol in parallel. The verifier picks poly(n) hash functions and points, and sends them all to the prover in Round 1. In Round 2 the prover sends all his responses.

3.2 Private v/s Public Coins

First we change the private-coins protocol for Graph Nonisomorphism to a public-coins protocol. We use the fact that the GS set lowerbound protocol uses public coins.

■ (Goldwasser, Sipser [GoSi]) $\overline{ISO} \in$ AM[2].

Proof: We reduce nonisomorphism to a statement about set-sizes, and use the GS protocol. Specifically, let

$$S = \{H \mid H \equiv G_1 \text{ or } H \equiv G_2\}.$$

Now, if $G_1 \not\equiv G_2$, then $|S| = 2n!$; otherwise, if $G_1 \equiv G_2$, then $|S| = n!$. (Actually this is not quite correct if the graphs have nontrivial automorphisms. It is true only if $|aut(G_1)| = |aut(G_2)| = 1$. But this defect can be corrected by using the set $\{(H, \pi) \mid (H \equiv G_1 \text{ or } H \equiv G_2) \text{ and } \pi \in aut(H)\}$ instead. But for easier presentation, let's stick to our set S). Thus if $G_1 \not\equiv G_2$ the prover can use the GS protocol to prove (in 2 rounds using public coins) that $|S| \geq 2n!$. So $\overline{ISO} \in$ AM[2].

□

Trivially, AM[2] $\subseteq$ BP·NP. Using the above theorem and our earlier observations about collapsing the polynomial-time hierarchy, it is clear that Graph Isomorphism cannot be NP-complete unless the polynomial-time hierarchy collapses. (Otherwise $\overline{ISO}$ is co-NP-complete, and it follows that all of co-NP is included in BP·NP which implies the collapse.)

■ (Boppana, Hastad, Zachos [BoHaZa]) If Graph Isomorphism is NP-complete, then the polynomial-time hierarchy collapses to Σ_2^P=BP·NP.

(See also [Scho] and [KST].)

Actually the above proof method works not only for Graph Isomorphism, more generally it can be shown:

■ (Goldwasser, Sipser [GoSi]) IP[2] $\subseteq$ AM[4].

Proof: In a two-round (private coin) protocol, on input x, the Verifier V picks a random number r, computes $\hat{q} = q(x\#r)$ and sends $\hat{q}$ to the Prover. The Prover selects an answer a such that the probability that the Verifier afterwards accepts, i.e. $q(x\#r\#a)$=accept, is maximized. By some lower bound protocol, using universal hashing, as above, we would like to prove that there are many r's which make the verifier accept. A technical problem here is that the set of r's is divided in equivalence classes where r and r' are considered equivalent (from the Prover's point of view) if $q(x\#r) = q(x\#r')$. What we need to show is, in some sense, that there are "many" equivalence classes, each of which has a "large" number of r's which make V accept. We need to show that "many" $\times$ "large" is sufficiently big, by some lower bound protocol.

Consider some equivalence class, specified by $\hat{q}$ and let a be the optimal Prover's answer to $\hat{q}$. Define the *weight* of the equivalence class $\hat{q}$ to be the number of random strings r such that $q(x\#r) = \hat{q}$ and $q(x\#r\#a)$=accept. Suppose for the moment that all equivalence classes have the same weight, say w. (Notice, this was the case in the Graph Isomorphism example.) Let t be the length of the random strings r so that there are 2^t many random strings. Say, we want to show that there are at least $2^t/2$ many r's which make V accept. Then we could proceed by two lower bound protocols as follows. The first protocol (which takes place in round 1 and 2) makes sure that (with high probability) there are at least $2^t/2w$ equivalence classes. [I.e. the Verifier picks a random hash function h with range size $2^t/2w$ and asks the Prover to present some $\hat{q}$ with $h(\hat{q}) = 1$. Also, at this point the Prover should send a, the best answer to $\hat{q}$ in the original protocol.] The second lower bound protocol (which takes place in round 3 and 4) makes sure that the weight of equivalence class $\hat{q}$ (from the first lower bound protocol) is at least w (with high probability). [I.e. the Verifier picks another random hash function h' with range size w and asks the Prover to present some r with $h'(r) = 1$ and also $q(x\#r) = \hat{q}$ and $q(x\#r\#a)$=accept. If every test is OK, the Verifier accepts.]

How to overcome the problem that the weights of the equivalence classes can be very much different from each other? We do some statistics: each equivalence class with non-zero weight w falls in one of the following categories $S_0, S_1, \ldots, S_t$ according to which S_i contains the weight w.

$$\begin{aligned}
S_0 &= \{1\}, \\
S_1 &= \{2\}, \\
S_2 &= \{3,4\}, \\
S_3 &= \{5,6,7,8\} \\
&\vdots \\
S_t &= \{2^{t-1}+1, \ldots, 2^t\}
\end{aligned}$$

If the sum of all weights is at least $2^t/2$, by the fact that there are $t+1$ categories, there must be some category S_i which collects equivalence classes of weight sum at least $2^t/2(t+1)$. The solution to our problem is to run the first lower bound protocol as described above in parallel for each category S_i. [Some more details: the Verifier selects random hash functions $h_0, h_1, \ldots, h_t$ and sends them to the Prover. The range of h_i has cardinality $2^{t-i}/2(t+1)$. The Prover is then expected to send $(i, \hat{q}, a)$, $i \in \{0, \ldots, t\}$, such that $h_i(\hat{q}) = 1$. In the second lower bound protocol it is verified that, with high probability, the weight of $\hat{q}$ is at least 2^{i-1}, and $q(x\#r) = \hat{q}$ and $q(x\#r\#a)$=accept.]

If the input x is in the language, then by the Claim above, each lower bound protocol can be satisfied with probability 1/2 so that the overall acceptance probability is at least 1/4. On the other hand, if x is not in the language, this means that there is only an exponentially small acceptance probability in the original protocol. Then there is a similar small probability in the new protocol. This can be proved by bounding the probability of the 2 cases "weight$(\hat{q}) \geq 2^{i-n/2}$" and "weight$(\hat{q}) < 2^{i-n/2}$" regarding the Prover's potential answer. The first case is very unlikely to pass the first lower bound protocol, the second case is very unlikely to pass the second lower bound protocol. □

The result can be generalized to IP$[k] \subseteq$ AM$[k+2]$, and together with Babai's result that the Arthur-Merlin hierarchy collapses, we get

$$\bigcup_{i \geq 1} \mathrm{AM}[i] = \bigcup_{i \geq 1} \mathrm{IP}[i] = \mathrm{AM}[2] = \mathrm{IP}[2].$$

References

[1] V. Arvind, J. Köbler, U. Schöning, R. Schuler. If NP has polynomial-size circuits, then MA=AM. *Theoretical Computer Science* 137 (1995) 279–282.

[2] L. Babai. Trading group theory for randomness. *17th ACM Symp. Theory of Computing 1985*, 421–429.

[3] L. Babai, S. Moran. Arthur-Merlin games: a randomized proof system and a hierarchy of complexity classes. *Journal of Computer and System Sciences* 36 (1988) 254–276.

[4] T. Baker, A.L. Selman. A second step toward the polynomial hierarchy. *Theoretical Computer Science* 8 (1979) 177–187.

[5] J. Balcázar. Self-reducibility structures and solutions of NP problems. *Revista Mathematica*, 175–184, Universidad Complutense de Madrid (1989).

[6] C.H. Bennett, J. Gill. Relative to a random oracle A, $P^A \neq NP^A \neq co\text{-}NP^A$ with probability 1. *SIAM Journal on Computing* 10 (1981) 96–113.

[7] M. Blum, S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM J. Comput.* 13 (1984) 850–864.

[8] R.B. Boppana, J. Hastad, and S. Zachos. Does co-NP have short interactive proofs? *Information Processing Letters* 25 (1987) 27–32.

[9] J.L. Carter, M.N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences* 18 (1979) 143–154.

[10] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on a sum of observations. *Annals Mathematical Statistics* 23 (1952) 493–509.

[11] T.H. Cormen, C.E. Leiserson, R.L. Rivest. *Introduction to Algorithms.* MIT Press 1990.

[12] O. Goldreich, S. Micali, A. Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design. *Proc. IEEE Symp. Foundations of Computer Science*, 1986, 174–187. Also: Proofs that release minimum knowledge. *Proc. Math. Foundations of Computer Science*, Lecture Notes in Computer Science 233, Springer 1986, 639–650.

[13] S. Goldwasser, S. Micali, C. Rackoff. The knowledge complexity of interactive proof systems. *Proc. 17th ACM Symp. Theory of Computing*, 1985, 291–304. Also: *SIAM Journal on Computing* 18 (1989) 186–208.

[14] S. Goldwasser, M. Sipser. Private coins versus public coins in interactive proof systems. *Proc. 18th ACM Symp. Theory of Computing*, 1986, 59–68. Also in: S. Micali (ed.): *Randomness and Computation*, Vol. 5 of *Advances in Computing Research*, JAI Press, 1989, pages 73–90.

[15] J.E. Hopcroft. Recent directions in algorithmic research. *Proc. Theoretical Computer Science*, Lecture Notes in Computer Science 104, Springer-Verlag, 1981, 123–134.

[16] R.M. Karp, R.J. Lipton. Some connections between nonuniform and uniform complexity classes. *Proc. 12th ACM Symp. Theory of Computer Science*, 302–309, 1980. Also: Turing machines that take advice. In: *Logic and Algorithmic*, Monographie No. 30 de l'Enseignement Mathematique, Université de Genève, 1982, 255–274.

[17] J. Köbler, U. Schöning, J. Torán. *The Graph Isomorphism Problem: Its Structural Complexity.* Birkhäuser Verlag, Boston 1993.

[18] C. Lautemann. BPP and the polynomial hierarchy. *Information Processing Letters* 14 (1983) 215–217.

[19] U. Schöning. Graph isomorphism is in the low hierarchy. *Journal of Computer and System Sciences* 37 (1988) 312–323.

[20] U. Schöning. Probabilistic complexity classes and lowness. *Journal of Computer and System Sciences* 39 (1989) 84–100.

[21] A. Shamir. IP = PSPACE. *IEEE Symp. Foundations of Computer Science*, 1990, 11–15.

[22] M. Sipser. A complexity theoretic approach to randomness. *Proc. 15th ACM Symp. Theory of Computer Science*, 1983, 330–335.

[23] C.K. Yap. Some consequences of non-uniform conditions on uniform classes. *Theor. Computer Science* 26 (1977) 23–33.

McGill/Barbados Workshop'96	**Randomness in Complexity Theory**
Lecture 2: Hardness Results for the Permanent	
Lecturer: *Sanjeev Arora*	Notes: *Martin Beaudry and Carlos Zamora-Cura*

Valiant has shown that computing the permanent (whether over integers or over any finite field of characteristic > 2) is $\sharp\mathbf{P}$-complete. Toda's theorem implies that unless the **PH** collapses, the permanent cannot be computed in polynomial time. Note that this is result only talks about worst-case hardness.

In this session Sanjeev presented a very nice result from [FL] which shows that the permanent over finite fields is hard to compute "almost everywhere" (i.e., for almost every matrix). This is done as follows. Suppose there is a heuristic that computes the permanent on "a few" matrices. We show below that this implies that $\mathbf{P}^{\sharp\mathbf{P}} = \mathbf{AM}[2]$. But by Toda's Theorem, $\mathbf{coNP} \subseteq \mathbf{P}^{\sharp\mathbf{P}}$, so we get $\mathbf{coNP} \subseteq \mathbf{AM}[2]$, and hence by the previous lecture, $\mathbf{PH} = \Sigma_3^p$. Since **PH** is believed to not collapse, this provides evidence that the heuristic does not exist.

1 Preliminaries

Throughout this section, p denotes a prime number. All arithmetic is in $\mathbf{F}_p$. We assume that $\{1, \ldots, n\} \subseteq \mathbf{F}_p$. For $i \leq n$, let $L_i^n(x)$ be the degree $n-1$ polynomial

$$L_i^n(x) = \prod_{j \in \{1,\ldots,n\} \setminus \{i\}} \frac{(x-j)}{(i-j)}. \qquad (1)$$

Note that $L_i^n(x) = 1$ at $x = i$ and 0 at $x \in \{1, \ldots, n\} \setminus \{i\}$. Now we state an obvious fact about *polynomial interpolation.*

Fact 1 For any n values $b_1, \ldots, b_n \in \mathbf{F}_p$, there is a unique degree $n-1$ univariate polynomial g such that

$$g(i) = b_i \qquad \text{for } i = 1, \ldots, n.$$

Proof: Just let $g(x) = \sum_{i \leq n} b_i \cdot L_i^n(x)$. QED

Now we define the permanent.

Definition 1 *Let $A = (a_{i,j}) \in \mathbf{F}_p^{n \times n}$. Then*

$$\mathsf{perm}(A) \stackrel{\text{def}}{=} \sum_{\pi \in \mathcal{S}_n} \prod_{i=1}^{n} a_{i,\pi(i)} = \sum_{i=1}^{n} a_{1,i} \cdot A_{1,i},$$

where $A_{1,i}$ is A without its 1^{st} row and i^{th} column (the $(1,i)$ minor of A).

A simple induction shows that $\mathsf{perm}(A)$ is a degree n (multivariate) polynomial in the entries of A.

2 The permanent is random self-reducible

As a warmup, we begin by showing Lipton's result [Lipton] that the permanent is a randomly self-reducible function.

Theorem 1 [Lipton] *If there exists a program Π that computes* $\mathsf{perm}(A)$ *for most of the $A \in \mathbf{F}_p^{n\times n}$, namely for a fraction $1 - \frac{1}{3n}$ of them, then there is a randomized program Π' that computes* $\mathsf{perm}(A)$ *for all matrices (with high probability).*

Remark: An alternative statement is that if the heuristic Π exists for all primes p, then $\mathbf{P}^{\sharp\mathbf{P}} \subseteq \mathbf{BPP}$.

Proof: Let $R \in \mathbf{F}_p^{n\times n}$ be a matrix, and $x \in \mathbf{F}_p$ be a variable. Observe that each entry in $A + x \cdot R$ is a linear function in x. Hence $g_R(x) \stackrel{\text{def}}{=} \mathsf{perm}(A + x \cdot R)$ is a degree-n polynomial in x, with $g_R(0) = \mathsf{perm}(A)$. Observe further that if we fix any nonzero value for x, say $x = i$, and pick R randomly from $\mathbf{F}_p^{n\times n}$, then the matrix $A + iR$ is uniformly distributed on $\mathbf{F}_p^{n\times n}$.

The new program Π' will pick $R \in \mathbf{F}_p^{n\times n}$ randomly and use Π to compute $g_R(x)$. The idea is to just run Π on $A + i \cdot R$ for $i = 1, \ldots, n+1$. Note that since each $A + i \cdot R$ is randomly distributed on $\mathbf{F}_p^{n\times n}$,

$$\Pr_R[\Pi \text{ computes } \mathsf{perm}(A + i \cdot R) \text{ correctly}] \geq 1 - \frac{1}{3n}.$$

Hence with probability at least $1 - \frac{n+1}{3n} \approx \frac{2}{3}$, Π computes $\mathsf{perm}(A + i \cdot R)$ correctly for *each* $i = 1, \ldots, n+1$. But if this happens, then we know the values of $g_R(x)$ at $n+1$ points and so can reconstruct $g_R(x)$ by interpolation. Then we output $g_R(0) = \mathsf{perm}(A)$.

The above procedure may output an incorrect value of $\mathsf{perm}(A)$ with probability $\approx 1/3$. But by repeating the procedure $poly(n)$ times and taking the majority answer this error probability can be made extremely small.

3 LFKN's Interactive Proof system for the permanent

We prove the result of [LFKN] that the language PERM $= \{(A, K) : \mathsf{perm}(A) = K\}$ is in IP. This shows that $\mathbf{P}^{\sharp\mathbf{P}} \subseteq$ IP.

Protocol 1 *The LFKN protocol for PERM.*
Input*: (A, K), where A is an $n \times n$ matrix and $K \in \mathbf{F_p}$.*

$V \to P$*: $D(x) \stackrel{\text{def}}{=} \sum_i L_i^n(x) \cdot A_{1,i}$. (Note that $D(x)$ is an $(n-1) \times (n-1)$ matrix whose each entry is a degree $(n-1)$ polynomial in x. Furthermore, $D(i) = A_{1,i}$ for $i = 1, \ldots, n$.)*

$P \to V$: *A polynomial $g(x)$ of degree $(n-1)^2$ that is supposedly* $\mathsf{perm}(D(x))$. *V computes $g(i)$ for $i = 1, \ldots, n$ and rejects immediately if $K \neq \sum_{i=1}^{n} a_{1,i} \cdot g(i)$.*

$V \to P$: *A random element $a \in_R \mathbf{F}_p$. ¿From now on P has to show that $g(a) =$ $\mathsf{perm}(D(a))$. In other words, all the above actions are repeated with A replaced by the $(n-1) \times (n-1)$ matrix $B(a)$, and K replaced with $g(a)$.*

At the end, the verifier is left with a 2×2 matrix and a claimed value for its permanent. The verifier computes the true permanent and accepts iff it matches the claimed value.

Clearly, if $\mathsf{perm}(A) = K$, then the the prover can answer in a way that makes the verifier accept with probability 1. We claim that if $\mathsf{perm}(A) \neq K$, then the verifier accepts with probability at most n^3/p. We prove this by induction on n. The base case $n = 1, 2$ is obviously true and we prove the inductive step.

Let $D(x)$ be the matrix of polynomials as described in the protocol and $f(x)$ be a degree $(n-1)^2$ polynomial that is $\mathsf{perm}(D(x))$. Note that since $\mathsf{perm}(A) \neq K$, the prover cannot afford to make his polynomial $g = f$. The reason being that $K \neq \mathsf{perm}(A) = \sum_i a_{1,i} \cdot f(i)$, so the verifier would reject immediately if $g = f$. The prover must therefore give a $g \neq f$. But then g, f agree on at most $(n-1)^2$ points in $\mathbf{F}_p$. When V picks a randomly from $\mathbf{F}_p$, the probability that $g(a) = f(a)$ is at most $(n-1)^2/p$. Noting that if $g(a) \neq f(a)$ then $g(a) \neq \mathsf{perm}(D(a))$, we can upperbound $\Pr[V$ accepts on $(A, K)]$ by

$$\begin{aligned} \leq \quad & \Pr_{a \in \mathbf{F_p}}[\mathrm{g(a)} = \mathsf{perm}(D(a))] \\ & + \Pr[\text{verifier accepts in } (n-1) \text{ rounds on } (D(a), g(a)) \mid g(a) \neq \mathsf{perm}(D(a))]. \end{aligned}$$

The first term is at most $(n-1)^2/p$, as already noted, and the second term is at most $(n-1)^3/p$ by the inductive hypothesis. So the overall probability of acceptance is at most n^3/p, and our induction is complete.

4 Feige-Lund Result

A *heuristic* for a computational problem is a deterministic algorithm that computes the correct answer for a fraction ε of the inputs (the set of inputs is infinite, so this "fraction" has to be defined suitably; see the next theorem).

Theorem 1 ([FL]) *If there exists a polynomial-time heuristic $\mathcal{P}$ that for all $n \geq 1$ and primesp computes the permanent correctly on at least a fraction q of $\mathbf{F_p^{n \times n}}$, where $q = \max\{16\sqrt{\frac{n^2}{p}}, \frac{100n^3}{p}\}$, then $\mathbf{P}^{\sharp\mathbf{P}} \subseteq \mathbf{AM}[2]$.*

The proof will use the Goldwasser-Sipser set lowerbound protocol, which was described in the last lecture. At the heart of all this is a very simple observation. Suppose we have n fixed values $b_1, \ldots, b_n \in \mathbf{F_p}$. Suppose we pick two random values $r_1, r_2 \in \mathbf{F_p}$ and construct a degree $n+1$ polynomial g such that $g(i) = b_i$ for $i = 1, \ldots, n$, and $g(n+1) = r_1$ and $g(n+2) = r_2$. In other words,

$$g(x) = \sum_{i=1}^{n} b_i \cdot L_i^{n+2}(x) + r_1 \cdot L_{n+1}^{n+2}(x) + r_2 \cdot L_{n+2}^{n+2}(x).$$

Since r_1, r_2 were randomly picked, it is clear by inspection that for any fixed $a, b \in \mathbf{F_p} \setminus \{1, \ldots, n\}$, the values $g(a), g(b)$ are randomly distributed in $\mathbf{F_p}$, and furthermore they are *pairwise independent* —assuming $a \neq b$. This observation will be crucial.

Feige and Lund use the heuristic $\mathcal{P}$ mentioned in the hypothesis of the Theorem to turn the LFKN protocol into a two round **AM** protocol. They also use the Goldwasser-Sipser set lowerbound protocol described in the previous lecture. For simplicity we first describe an $O(n)$-round protocol, and then indicate why the rounds can be done "in parallel."

Protocol 2 *The Feige-Lund protocol for the permanent; $O(n)$ rounds.*
Input: (A, K) where A is an $n \times n$ matrix and $K \in \mathbf{F_p}$.

$V \to P$: *$B(x) \stackrel{\text{def}}{=} \sum_i A_{1,i} \cdot L_i^{n+2}(x) + R_1 \cdot L_{n+1}^{n+2}(x) + R_2 \cdot L_{n+2}^{n+2}(x)$, where $R_1, R_2 \in_R \mathbf{F}_p^{n \times n}$. Note that $B(x)$ is an $(n-1) \times (n-1)$ matrix whose each entry is a degree $n+1$ polynomial in x.*

$P \to V$: *A degree $(n+1)(n-1)$ polynomial $g(x)$ that is supposedly $\mathsf{perm}(B(x))$. V checks that $K = \sum_{i=1}^{n} a_{1,i} \cdot g(i)$, and rejects immediately if not so.*

$P \leftrightarrow V$: *P and V run the Goldwasser-Sipser lowerbound protocol during which P tries to convince V that*

$$|\{ x \in \mathbf{F}_p \mid g(x) = \mathcal{P}(B(x)) \}| \geq \frac{qp}{2}. \tag{2}$$

(In other words, V sends some random hash functions $h_1, \ldots, h_l$ whose domain is $\mathbf{F}_p$ and whose range U is a set of size $\frac{qp}{4}$. V also sends some random values $u_1, \ldots, u_l \in U$. In response P sends values $x_1, \ldots, x_l \in \mathbf{F}_p$ such that $h(x_i) = u_i$ and "many" x_i are in $\{ x \in \mathbf{F}_p \mid g(x) = \mathcal{P}(B(x)) \}$. Note that V can check in polynomial time whether or not x_i is in the indicated set, since $\mathcal{P}$ runs in polynomial time.)

$V \to P$: *A random value $a \in_R \mathbf{F_p}$. ¿From now on P is supposed to prove that $\mathsf{perm}(B(a)) = g(a)$. In other words, all the above actions are repeated with A replaced by the $(n-1) \times (n-1)$ matrix $B(a)$, and K replaced with $g(a)$.*

Notice that the above protocol is essentially the same as the LFKN protocol but with two changes: (i) $B(x)$ is defined using random matrices R_1, R_2 and (ii) the verifier uses the Goldwasser-Sipser protocol to gain some confidence in the "quality" of the prover's polynomial g: this polynomial is required to agree with the heuristic "reasonably" often (cf. Condition (2)).

Suppose $\mathsf{perm}(A) \neq K$. Then the proof of correctness of LFKN can be trivially modified to show (by just ignoring the G-S part) that the verifier rejects with high probability.

Now suppose $\mathsf{perm}(A) = K$. We show that the honest prover can make the verifier accept with high probability. Specifically, we show that if the prover plays by the rules and returns $g(x) = \mathsf{perm}(B(x))$ then with high probability (over the choice of R_1, R_2) this polynomial satisfies condition (2). We use our observation at the start of this Section. Since $B(x) = \sum_i A_{1,i} \cdot L_i^{n+2}(x) + R_1 \cdot L_{n+1}^{n+2}(x) + R_2 \cdot L_{n+2}^{n+2}(x)$ and R_1, R_2 are random, for each $u \in \mathbf{F_p} \backslash \{1, \ldots, n\}$ the matrix $B(u)$ is randomly distributed in $\mathbf{F}_p^{n \times n}$. Since the heuristic $\mathcal{P}$ computes the permanent correctly for q fraction of inputs, we have $\Pr_{R_1,R_2}[\mathcal{P}(B(u))$ is correct$] \geq q$. Furthermore, if $u \neq v$ then $B(u)$ and $B(v)$ are pairwise independent. Hence Chebyshev's inequality implies that with extremely high probability (something like $1 - 1/\sqrt{qp}$), the fraction of $u \in \mathbf{F_p} \setminus \{1, \ldots, n\}$ such that $\mathcal{P}(B(u))$ is correct is at least $qp/2$. Hence w.h.p., $g(x) = \mathsf{perm}(B(x))$ satisfies (2).

Feige and Lund claim that the $O(n)$ rounds in the above protocol can be carried out in 2 rounds. Note first that the verifier's queries are *nonadaptive*: they do not depend upon the prover's answers but only upon the input and the verifier's random coin-tosses. In particular, note that generating the hash functions used in the G-S protocol requires only that we know the prime p and the fraction $q = \max\{16\sqrt{\frac{n^2}{p}}, \frac{100n^3}{p}\}$.

Protocol 3 *The Feige-Lund protocol for the permanent;* 2 *rounds.*
Input: (A, K), *where* A *is an* $n \times n$ *matrix and* $K \in \mathbf{F_p}$.

$V \to P$: *All the information it would need to send in Protocol 2. This means the matrices* R_1, R_2, *the hash functions sent in doing the Goldwasser-Sipser protocol, and the random value* a. *Also, all the information it will send in the interactive protocol for the* $(n-1) \times (n-1)$ *matrix* $D(a)$.

$P \to V$: *All the information that it sends to* V *in Protocol 2. This means a degree* $(n+2)(n-1)$ *polynomial* $g(x)$, *its response in the Goldwasser-Sipser protocol (in response to the hash functions sent by* V*), and then its responses to* V*'s questions in the interactive proof for* $(D(a), g(a))$.

The 2-round protocol may seem like a bad idea, since the prover gets to see the value a *before* it chooses its polynomial g. Hence it could conceivably choose a polynomial that satisfies $g(a) = \mathsf{perm}(B(a))$ and $K = \sum_i a_{1,i} \cdot g(i)$ (indeed, there are many polynomials that satisfy these two conditions) and go away scot-free from then on.

But recall that the prover's polynomial g also has to satisfy condition (2) (or else the G-S lowerbound protocol rejects w.h.p.). We show that this greatly limits his options: Fact 2 below implies that the number of polynomials g that satisfy condition (2) is quite small.

Fact 2 *Let $\tau > 0$ and let d be an integer such that $\tau > 2\sqrt{d/p}$. Then for any function $f : \mathbf{F_p} \to \mathbf{F_p}$, there are at most $2/\tau$ degree d polynomials g such that*

$$|\{x \in \mathbf{F_p} : f(x) = g(x)\}| \geq \tau \cdot p.$$

Proof: Let $g_1, \ldots, g_N$ be those polynomials, and let $S_i \subseteq \mathbf{F_p}$ be the set

$$\{x \in \mathbf{F_p} : f(x) = g_i(x)\}.$$

Then each $|S_i| \geq \tau p$. For each $i \neq j$, the size of $|S_i \cap S_j|$ is $\leq d$, since two different degree d polynomials can agree in at most d points. Hence we have

$$\begin{aligned} p \geq \left|\bigcup_i S_i\right| &\geq \sum_i |S_i| - \sum_{i \neq j} |S_i \cap S_j| \\ &\geq N \cdot p \cdot \tau - \binom{N}{2} d \end{aligned}$$

Hence if $N \geq 2/\tau$ and $\tau > 2\sqrt{d/p}$, we get $p > p$, which is a contradiction. QED

To show the correctness of the 2-round proof system, we mimic the proof of LFKN. Suppose $K \neq \mathsf{perm}(A)$. We show by induction on n that the probability that the verifier accepts is at most $4n^3/qp$. The base cases $n = 1, 2$ are trivial and we complete the inductive step.

If the prover gives $\mathsf{perm}(B(x))$ as $g(x)$, then the verifier rejects immediately, so assume $g(x) \neq \mathsf{perm}(B(x))$. The prover has two options for what kind of polynomial to give as g.

Option 1: g does not satisfy condition (2).
In this case the GS protocol rejects with high probability, say $1 - 2^{-n}$.
Option 2: g satisfies condition (2).
Fact 2 implies that there are only $4/q$ such polynomials, say $g_1, g_2, \ldots$. Then each g_i agrees with $\mathsf{perm}(B(x))$ in at most $(n-1)(n+1)$ values of x. So when we assign $a \in_R \mathbf{F_p}$ to x,

$$\Pr_a[\exists i \ : \ \text{s.t. } g_i(a) = \mathsf{perm}(B(a))] \leq \frac{4}{q} \cdot \frac{(n-1)(n+1)}{p}.$$

Note the prover's predicament! Even though he knows $a \in \mathbf{F_p}$ before he picks g, the probability (over the choice of a) that there *exists* a g satisfying condition (2) and satisfying $g(a) = \mathsf{perm}(B(a))$ is at most $\frac{4(n-1)(n+1)}{qp}$. Furthermore, if the prover's polynomial g is such that $g(a) \neq \mathsf{perm}(B(a))$, then the inductive hypothesis implies that the prover has at most a chance $4(n-1)^3/qp$ of making the verifier accept during the interactive proof on the input $(B(a), g(a))$. Hence the total chance of making the verifier accept on (A, K) is

$$\begin{aligned}
&\Pr_a[\exists i\ :\ g_i(a) = \mathsf{perm}(B(a))] + \text{error probability of GS protocol} \\
&+ \Pr[\text{verifier accepts on input } (B(a), g(a)) \mid g(a) \neq \mathsf{perm}(B(a))] \\
&\leq \tfrac{4(n-1)(n+1)}{qp} + \tfrac{1}{2^n} + \tfrac{4(n-1)^3}{qp},
\end{aligned}$$

which is at most $4n^3/qp$. This completes our induction.

References

[1] U. Feige and C. Lund, On the Hardness of Computing the Permanent of Random Matrices, *Proc. of the 24th ACM Symp. on the Theory of Computing* (1992), pp. 643-654.

[2] S. Goldwasser, M. Sipser, Private coins versus public coins in interactive proofs systems, *Advances in computing research*, **5** (1989), pp. 73-90.

[3] R. Lipton, New directions in testing, in *Distributed Computing and Cryptography*, J. Feigenbaum and M. Merritt, eds., *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, vol. 2, American Mathematical Society (1991), pp. 191-202.

[4] C. Lund, L. Fortnow, H. Karloff and N. Nisan, Algebraic methods for interactive proof systems, *Proc. of the 31th IEEE Symp. on the Foundations of Computer Science* (1990), pp. 2-10.

McGill/Barbados Workshop'96	Randomness in Complexity Theory
Lecture 3: Hardness of Approximation and the PCP Theorem	
Lecturer: *Sanjeev Arora*	Notes: *David Barrington and Alexis Maciel*

1 Introduction

The problems 3SAT and CLIQUE are NP-complete, and so we believe that they cannot be solved in polynomial time. Clearly, the NP-hardness of these decision problems implies the NP-hardness of the corresponding optimization problems. For example, the optimization version of 3SAT is

MAX-3SAT
input A 3CNF formula φ
output An assignment that maximizes the number of satisfied clauses in φ

In this lecture, we begin a proof that computing even approximations to an optimal assignment is NP-hard. More precisely, let MAX-3SAT(φ) denote the maximum proportion of clauses in φ that can be satisfied by any assignment of the variables. We will prove that there is a number $\beta > 1$ for which the following problem is NP-hard:

computing β-approximations to MAX-3SAT
input A 3CNF formula φ
output An assignment that satisfies at least $(1/\beta)$MAX-3SAT(φ) of the clauses of φ

This hardness result for MAX-3SAT — proved using the PCP Theorem — implies the hardness of approximating many other NP-hard problems. The next few lectures are devoted to these results.

2 Hardness of approximation via the PCP Theorem

To prove the NP-hardness of computing β-approximations to MAX-3SAT for some fixed $\beta > 1$, we will show that the following decision problem is NP-hard for some $\varepsilon > 0$:

MAX-3SAT(ε)

input A 3CNF formula φ such that either MAX-3SAT(φ) $= 1$ or MAX-3SAT(φ) $< 1 - \varepsilon$

output 1 if φ is satisfiable, 0, otherwise

Let us recall the Cook-Levin proof of the NP-hardness of MAX-3SAT to see that it does not suffice. Let L be an NP language and M be an NP Turing machine that recognizes it. Given any input string x, Cook-Levin construct a formula $\varphi_{x,L}$ that represents the computation of M on input x. This formula is satisfiable iff $x \in L$. Hence a satisfying assignment to this formula is a "membership proof." However, this membership proof is not "robust," in the sense that even if $x \notin L$, there are assignments that satisfy *almost all* the clauses in $\varphi_{x,L}$ (i.e., the fraction of unsatisfied clauses $\to 0$ as $|x| \to \infty$) . This is clear by noticing that the Cook-Levin construction represents each step of the computation by a small set of clauses, and so a bad assignment could just satisfy the clauses for all time steps except one at which it "switches" the computation.

Thus we need a better construction such that if $x \notin L$, then the fraction of unsatisfiable clauses is a fixed constant $\epsilon > 0$. In other words, we need a more "robust" type of membership proof. Such proofs of membership will be provided by the PCP Theorem.

A language L is in PCP($\log n, 1$) if membership in L can be probabilistically verified as follows: there is a polynomial-time probabilistic Turing machine V, the verifier, that given an input string x, a "proof" string π and a random string r, satisfies the following conditions:

$$x \in L \Rightarrow \exists \pi \ \text{Prob}[V^{\pi}(x,r) \text{ accepts}] = 1$$

$$x \notin L \Rightarrow \forall \pi \ \text{Prob}[V^{\pi}(x,r) \text{ accepts}] < 1/2$$

Moreover, the random string is of length $O(\log n)$ and the verifier non-adaptively queries $O(1)$ bits of the proof string, i.e., V decides, based on x and r, which positions of π it will examine before it starts examining π.

It is not difficult to see that PCP($\log n, 1$) $\subseteq$ NP. The second half of this lecture and the next two lectures will be devoted to the proof of the converse: NP $\subseteq$ PCP($\log n, 1$). This characterization of NP in terms of probabilistically checkable proofs constitutes the PCP Theorem:

Theorem 2 ([ALMSS],many ideas were in [AS]) NP $=$ PCP($\log n, 1$).

The connection between hardness of approximation and the PCP Theorem is given by the following:

Theorem 3 ([ALMSS]) MAX-3SAT(ε) *is* NP-*hard for some* $\varepsilon > 0$ *if and only if* NP $=$ PCP($\log n, 1$).

Proof: ($\Rightarrow$): Let L be an NP language and x an arbitrary input string. Since MAX-3SAT(ε) is NP-hard, we can reduce L to MAX-3SAT(ε) in polynomial time. Now we show $L \in \text{PCP}(\log n, 1)$. Let x be any input. The verifier first reduces x to an instance φ of MAX-3SAT(ε) that is satisfiable iff $x \in L$. As membership proof for x, the verifier expects a satisfying assignment for φ. Given such a membership proof, it does a probabilistic check on it by randomly choosing a clause in φ and querying the proof for the values of the three variables involved in that clause. It accepts iff the clause is satisfied. Clearly, if $x \in L$, then a satisfying assignment for φ will cause the verifier to accept with probability 1. On the other hand, if $x \notin L$, then every assignment satisfies less than a $1-\varepsilon$ fraction of the clauses and so the verifier accepts with probability less than $1 - \varepsilon$. By repeating the check $O(1/\varepsilon)$ times (which is still some fixed constant), the acceptance probability can be reduced to $1/2$.

($\Leftarrow$): Suppose that $\text{NP} \subseteq \text{PCP}(\log n, 1)$, and hence in particular, $\text{SAT} \in \text{PCP}(\log n, 1)$. We show the NP-hardness of MAX-3SAT(ε) for some fixed $\varepsilon > 0$. Let ψ be an instance of SAT. Given a value for the random string r, let $q_1(\psi, r), \ldots, q_c(\psi, r)$ denote the positions of the proof string queried by the verifier. For each value of r, the computation of the verifier is determined by the value of the proof string at these positions. Therefore, the outcome of the computation is a function of a constant number of bits in the proof and so can be represented by a constant-size 3CNF formula $\varphi_{\psi,r}$, in the sense that $\varphi_{\psi,r}$ is true if and only if $V^\pi(\psi, r)$ accepts. Let φ_ψ be the concatenation of $\varphi_{\psi,r}$ for all possible values of r. If ψ is satisfiable, then there is a proof π such that all $\varphi_{\psi,r}$ are true, so that φ_ψ is satisfiable. On the other hand, if ψ is not satisfiable, then, for every π, at least one clause in each of half of the $\varphi_{\psi,r}$ is not satisfied, which implies that a constant fraction of the clauses in φ_ψ are not satisfied. Since the formula φ_ψ can be constructed in polynomial time, this shows that SAT reduces to MAX-3SAT(ε) for some $\varepsilon > 0$. $\square$

3 Proof of the PCP Theorem

We now proceed to the proof that $\text{NP} \subseteq \text{PCP}(\log n, 1)$. Since SAT is NP-complete, it suffices to show $\text{SAT} \in \text{PCP}(\log n, 1)$. The proof relies upon an algebraic way of looking at SAT. Specifically, it uses the fact that an assignment can be coded as a polynomial over a finite field. The verifier for SAT will expect a proof of satisfiability to be such an encoding of the satisfying assignment, where the polynomial is represented *by value*. We will describe special algebraic procedures that allow the verifier to check such a proof. Next, we give the history of some of these ideas.

Coding theory. The idea of encoding information using polynomials goes back to coding theory. Suppose that a sender wants to transmit a message m to a receiver in a such a way that even if some bits get changed during transmission, the receiver can recover the message. To accomplish this, the sender will send an encoding of the message m. For example, let F be a finite field and t an integer such that $|F| \geq 3t$. Fix $3t$ elements $a_1, \ldots, a_{3t}$ of F. Given a message m of length $t \log |F|$, the sender breaks m into t pieces of length $\log |F|$ and views those pieces as elements of F in some canonical way. Let $b_1, \ldots, b_t$ denote these elements. The sender then calculates the unique degree-$(t-1)$ interpolating polynomial $g(x)$ of $(a_1, b_1), \ldots, (a_t, b_t)$ and transmits $g(a_1), \ldots, g(a_{3t})$. Now suppose that no more than t errors occur during transmission. Distinct polynomials of degree $t-1$ agree on at most $t-1$ points. Therefore, there is a unique polynomial of degree $t-1$ that agrees on at least $2t$ points with the function defined by the values received. This polynomial must be $g(x)$. The receiver can therefore find $g(x)$ and recover the message from $g(a_1), \ldots, g(a_t)$. (Note that we avoided an algorithmic issue: Can such a g be reconstructed efficiently? The answer is yes. Sanjeev later described a decoding algorithm.)

Secret sharing. A related fact noticed by Shamir is that if in the above encoding, we evaluate the polynomial at random points, then its values look random. Effectively, this causes the original message to be "dispersed" into random-looking bits. Specifically, Shamir designed a cryptographic primitive called *secret sharing*. Assuming the same set-up as in the previous example, suppose that the pieces of the message are to be distributed among n players so that no subset of $\leq k$ of them can reconstruct the message (or even one bit of information about the message), but every subset of $\geq t+k+1$ of them can. (Thus the secret neccessarily requires a "quorum" to be revealed.) The scheme is as follows: pick $k+1$ random elements $b_{t+1}, \ldots, b_{t+k+1}$ of F, and compute the unique degree-$(t+k)$ interpolating polynomial $g(x)$ of $(a_1, b_1), \ldots, (a_{t+k+1}, b_{t+k+1})$. Then evaluate $g(x)$ at any n points known to all players, and distribute these values to the n players. The key property is that these values are k-wise independent, meaning that for any subset of k players, all k-tuples of values are equally likely, no matter what the message m was. (Recall that in Lecture 2 the Feige-Lund result on the permanent used a similar construction, but with pairwise independence.) Therefore, *no* subset of k players can reconcstruct even one bit of information about m. Nevertheless, *every* subset of $t+k+1$ players can compute the interpolating polynomial and thus reconstruct m. These ideas of robust encoding and dispersal of information by means of polynomials will play a key role in what follows.

We now proceed to the proof that $\mathrm{NP} \subseteq \mathrm{PCP}(\log n, 1)$.

The encoding. We describe how to encode assignments as polynomials. (We call the resulting polynomial the *polynomial extension* of the assignment.) Let F be a finite field, let h be an integer less than $|F|$ and let m be an integer such that $(h+1)^m = n$. Fix an embedding of $[0, h]$ into F and a bijection between $\{1, \ldots, n\}$ and $[0, h]^m \subseteq F^m$. An assignment $s \in \{0, 1\}^n$ can then be viewed as a function $s : [0, h]^m \rightarrow \{0, 1\} \subseteq F$. The encoding $\hat{s}$ of s will be a degree-mh polynomial in m

variables such that

$$\hat{s}(u_1, \dots, u_m) = s(u_1, \dots, u_m),$$

for every $(u_1, \dots, u_m) \in [0, h]^m$. (Multivariate interpolation shows that such a polynomial exists.) This polynomial will be represented by all its values on F^m. This representation is not too large since it will be the case that $|F|$ is roughly h^3 and so $|F|^m$ is roughly n^3.

The procedures The verifier will use a procedure that as its input is given a function $f : F^m \to F$ and has to determine whether f is close to a polynomial of appropriate (small) degree. "Close" here means that the function agrees with such a polynomial on most values. This procedure will be probabilistic and will examine only very few values of f — eventually only a constant number of them. This is the *low-degree test*, the technically most difficult part of the proof of the PCP theorem.

A second procedure will be used by the verifier to compute, given f and $x \in F^m$, the value $\tilde{f}(x)$, where $\tilde{f}$ is the polynomial that f is close to. This too should examine only very few values of f.

3.1 The low-degree test

We first review some of the history of program checking, a field that gave rise to the low degree test. Recall our earlier example (Lecture 1) of the discrete log function. A black box able to calculate the discrete log on a few instances sufficed to calculate it for all instances. (To find $\log y$, we keep choosing random $r < p - 1$ until yg^r happens to be one of the instances on which the box is correct. We can tell when this has happened because the correct log is easily verified.) Another way to look at this is that we have a large table (consisting of all the correct values of the discrete log function) and we are using *random self-reducibility* to repair a partially correct version of the table. We will now see that polynomials over a field F provide another good example of random self-reducibility.

Suppose you are given a set of values, alleged to be close to (i.e., equal to for most inputs) a low-degree multivariate polynomial. Using the definitions above, let $\mathbf{u}$ and $\mathbf{v}$ be uniformly-chosen random elements of F^m. Then there is a line through F^m consisting of the point $\mathbf{u} + t(\mathbf{v} - \mathbf{u})$ for each $t \in F$. If the table were exactly equal to some polynomial f of degree mh, say, then each of these lines would itself be a univariate polynomial (in t) of degree mh. It is clear that if most of the points in the table agree with a fixed low-degree polynomial, then most points on most lines will agree with particular low-degree univariate polynomials, one for each line.

What is less clear, but still true as we shall see, is the converse: that *if* most points on most lines agree with low-degree polynomials, *then* there is a single multivariate low-degree polynomial which agrees with the entire table on most points. (Specifically, if a $1-\epsilon$ fraction of the points on a $1-\epsilon$ fraction of the lines agree, then so do a $1-\delta$

fraction of the points overall, where δ depends only on ϵ and not, for example, on m.) This result is proved in [ALMSS] by combining earlier results of Rubinfeld and Sudan [RS] and Arora and Safra [AS]. The basic thread of progress in all the "PCP" papers is an improvement of this *low-degree test*.

The key advantage of working with a line rather than a polynomial is that the number of function values along a line is poly$(\log n)$, rather than poly(n) as is the entire table. How do we verify a claimed behavior along a line? The verifier can demand a table, for every line (there are $|F|^{2m} = n^6$ of them) which gives a univariate polynomial describing the function values on the line. This polynomial has $mh + 1$ coefficients.

For our verifier, $h, |F|$ are poly$(\log n)$ and $m = \Theta(\log n / \log\log n)$. The low degree test operates by taking (say) 100 lines, reading their coefficients, and querying 100 points on each line. Thus the number of bits read by the verifier is $O(mh \log |F|)$, which is poly$(\log n)$. We will later reduce this number to be $O(1)$. For now, we note something that will later be important: at least the number of *pieces* of information (i.e., function values, line polynomials) is $O(1)$.

3.2 Reconstructing some values

Once the low degree test has operated, the verifier believes that the table f is mostly equal to a low-degree polynomial $\tilde{f}$. The next problem is for the verifier to obtain the value of $\tilde{f}$ at a number of points. (Of course, *a priori* the particular points the verifier wants might be among those for which f and $\tilde{f}$ have different values.)

We will need the concept of a *degree-k curve* which is a set of $|F|$ points in F^m with a parametric representation of the form $C(t) = \{(g_1(t), \dots, g_m(t)) : t \in F\}$, where each $g_i \in F[t]$ is a degree k univariate polynomial. If $s\colon F^m \to F$ is any function, we denote by s_C the restriction of s to curve C. Note that if $s \in F[x_1, \dots, x_m]$ is a degree d polynomial, then $s_C = s(g_1(t), \dots, g_m(t))$, which is a degree dk univariate polynomial (in the curve parameter t).

To get the values of $\tilde{f}$ for k designated points $z_1, \dots, z_k$, where k is at most poly$(\log n)$, we proceed as follows. We ask the prover —ahead of time— to place in the proof the polynomial that (supposedly) describes the restriction of $\tilde{f}$ on each degree-k curve passing through $z_1, \dots, z_k$. For each curve the degree of this univariate polynomial is about mhk, because $\tilde{f}$ has degree mh. Thus it can be represented using about $mhk \log |F|$ bits, which is poly$(\log n)$. We pick a random point $z_{k+1} \in_R F^m$, and construct a degree-k curve $C(t)$through the $k + 1$ points $z_1, \dots, z_{k+1}$. (We can construct such a curve by doing a Lagrange interpolation in each of the m coordinates.) We find the polynomial p corresponding to this curve in the table. Then we pick a random point u on the curve and verify that the value produced by p at u is equal to $f(x)$. Thus reassured about the "quality" of p, we use the values produced by p at $z_1, \dots, z_k$ as the values of $\tilde{f}(z_1), \dots, \tilde{f}(z_k)$.

We claim that the probability that the above procedure produces incorrect values of $\tilde{f}$ is very small. Note that since z_{k+1} is randomly picked, each point on the curve is randomly distributed in F^m (though not independently distributed). This is similar to the fact about secret sharing we noted earlier. Since f agrees with $\tilde{f}$ on some $1-\varepsilon$ fraction of the points, then on the average degree-k curve the number of points on which f and $\tilde{f}$ agree is $\approx 1-\varepsilon$. We conclude by averaging that on *most* (say, a fraction $1-\sqrt{\varepsilon}$) degree-k curves passing through $z_1, \ldots, z_k$, (that is, for most choices of z_{k+1}), the restrictions f_C and $\tilde{f}_C$ will agree on *most* values of t (say, a fraction $1-\sqrt{\varepsilon}$ points on C).

Let $p(C)$ denote the prover's polynomial for curve C. The only way the prover can cheat on this curve is by giving some polynomial $p(C) \neq \tilde{f}_C$ as the curve polynomial. We say a curve C is *good* if f_C, $\tilde{f}_C$ agree on $\geq 1-\sqrt{\varepsilon}$ fraction of its points. If C is a good curve and $p(C) \neq \tilde{f}_C$, then $p(C)$ must differ with $\tilde{f}_C$ on $1 - mhk/|F|$ fraction of points of C. So triangle inequality implies that $p(C)$ and f_C must differ on $1-\sqrt{\varepsilon}-mhk/|F|$ fraction of points. Thus we will expose such cheating with high probability, since we will compare the value of p and f_C at a random point u on the curve.

The probability that the prover successfully cheats is upperbounded by

$$\begin{aligned} \Pr_C[C \text{ is not good}] \quad &+ \Pr_{u\in C}[f \text{ and } p(C) \text{ agree on } u \mid C \text{ is good}] \\ \leq \sqrt{\varepsilon} \qquad\qquad &+ \sqrt{\varepsilon} + \frac{mhk}{|F|} \end{aligned}$$

Finally, we note an interesting fact. In order to reconstruct the value of $\tilde{f}$ at k points in F^m, we needed to to get only *two* pieces of information from the provided tables (namely, the polynomial corresponding to some degree-k curve and the value of f at some random point on the curve). This will be important later when we try to *aggregate* queries.

3.3 SAT $\in$ PCP$(\log n, \mathrm{polylog}(n))$

Let's review our progress towards a PCP system for SAT. Assume that the low degree test works as claimed. Given a formula ψ, a function f (given as a table of values), and the extra information outlined above, the verifier can convince herself (by looking at polylog many table entries) that the low-degree polynomial $\tilde{f}$ exists and represents a string. We'll later see the *sum check* procedure, by which the verifier may be convinced that this string is a satisfying assignment for ψ. (Here the main problem will be that she must not be fooled even by an assignment that satisfies all but one clause.) The verifier is doing this with only $O(\log n)$ random bits, as she is picking $O(1)$ points in F^m on which to query f. So modulo these two gaps, we have a proof that SAT is in the class PCP$(\log n, \mathrm{polylog}(n))$, which is more or less the central result of [BFLS]. Note that our verifier is *non-adaptive* in that future queries do not depend on the results of earlier queries but only on the random bits.

3.4 Towards SAT $\in$ PCP$(\log n, 1)$: aggregating queries

What remain are a number of refinements of this proof system which will eventually give us that SAT is in PCP$(\log n, 1)$. Here is the first. Currently the verifier needs to read polylog many "pieces" of information, of polylog bits each. We'd like to alter the protocol so she reads $O(1)$ pieces of polylog bits each. This trick is called *aggregating queries*, and it will use the procedure of Section 3.2.

Abstracting the situation, assume only that on input x the verifier V uses poly$(logn)$ bits from a proof Π, uses $O(\log n)$ random bits, and is non-adaptive. We design a verifier V_{new} which expects a proof Π_{new} consisting of poly(n) entries each of poly$(logn)$ bits. V_{new} will read $O(1)$ of these entries.

The main idea is that Π_{new} is supposed to be an encoding (using a low-degree polynomial) of a proof Π which the old verifier V would have accepted with probability 1. The verifier checks using the low degree test that Π_{new} is close to a low-degree polynomial, and gets k values from it by the method in Section 3.2, where k is the polylog number of values demanded of Π by V. That is, Π_{new} contains whatever information is needed for the low-degree test, together with the coefficients of the line polynomial $\tilde{\Pi}(c(t))$ for each polynomial c derived from a random point z_{k+1}. So V_{new} reads one of these line polynomials (one "piece" of information), looks up f at $O(1)$ points to check its validity ($O(1)$ more "pieces"), and then uses it to construct the desired k entries of Π. There are still only polynomially many entries in $\tilde{\Pi}$, because each corresponds to a possible value of the random bits chosen by V and V_{new}, and there are still only $O(\log n)$ of these.

3.5 Preview of the proof: Verifier Composition

The rest of the argument to put SAT in PCP$(\log n, 1)$ relies on a technique called *verifier composition* [AS]. This involves composing two verifiers and getting a verifier that is (when the parameters are right) more efficient than either.

We've seen verifiers V and V_{new}. After we perform the query aggregation of part (11), the verifier V_{new} can be seen to operate in *poly-log time*. This is because all it needs to do is to (a) look up $O(1)$ univariate polynomials each of poly-log degree, and (b) check the equality of $O(1)$ pairs of function values, each of poly-log bits. Now consider the tableau of V_{new}'s computation, as in the proof of the Cook-Levin theorem — it is a poly-log by poly-log table.

Composing V_{new} with itself will give us a third verifier V_{newnew}. The main idea in composition is that the proof for the new verifier Π_{newnew} is supposed to contain (for each of poly(n) choices of random string of V_{new}) an encoding of the entire tableau of V_{new}'s computation. This encoding is our standard one using low degree polynomials. The verifier V_{newnew} uses V_{new} in a recursive way to check that this encoding represents a tableau of an accepting computation of V_{new}! For reasons that Sanjeev did not specify in this lecture, V_{newnew} uses table entries of size only $O(\log\log n)$.

Then we construct a new, different verifier V_{diff}, which uses polynomially many random bits but reads only $O(1)$ bits of the proof. Composing V_{newnew} with V_{diff} gives a new verifier that will be a PCP$(poly, 1)$ procedure. During composition, V_{diff} is used to check that V_{newnew} would have accepted its proof. Even though V_{diff} needs "polynomially many" random bits this all works out OK since the running time of V_{newnew}'s computation was only poly$(\log\log n)$. So the input to the V_{diff} has size poly$(\log\log n)$, for which it needs poly(poly$(\log\log n)) = o(\log n)$ random bits.

References

[1] Arora, S., *Probabilistic Checking of Proofs and Hardness of Approximation*, Ph.D. Thesis, Computer Science Division, University of California, Berkeley, CA, USA. Revised version available as Technical Report CS-TR-476-94, Dept. of Computer Science, Princeton University, Princeton, NJ, USA. Anonymous ftp: ftp.cs.princeton.edu.

[2] Arora, S., Lund, C., Motwani, R., Sudan, M. and Szegedy, M., Proof verification and intractability of approximation problems. In *Proc. 33rd IEEE Symp. on Foundations of Computer Science*, 1992, 13–22.

[3] Arora, S. and Safra, S., Probabilistic checking of proofs. In *Proc. 33rd IEEE Symp. on Foundations of Computer Science*, 1992, 2–13.

[4] Babai, L., Fortnow, L., Levin, L. and Szegedy, M., Checking computations in polylogarithmic time. In *Proc. 23rd ACM Symp. on Theory of Computing*, 1991, 21–31.

[5] Rubinfeld, R. and Sudan, M., Testing polynomial functions efficiently and over rational domains. In *Proc. 3rd Annual ACM-SIAM Symp. on Discrete Algorithms*, 1992, 23–32.

[6] Shamir, A., How to Share a Secret, *Communications of the ACM*, 22 (1979) 612–613.

McGill/Barbados Workshop'96	Randomness in Complexity Theory
Lecture 4: The PCP Theorem, Continued.	
Lecturer: *Sanjeev Arora*	Notes: *Ricard Gavaldà and Denis Thérien*

1 Normal Form Verifiers

In the last lecture we said that the verifier for SAT is actually the composition of three verifiers, each one checking the computation of the previous one. We give now the details of this composition.

To be composed, verifiers have to be in a certain *Normal Form,* satisfying the following properties (note that n is the number of variables in the 3CNF formula whose satisfiability is being proved):

(i). The verifier is *nonadaptive*: the queries it makes to the proof depend only upon the input and the random string. (By contrast if the verifier were *adaptive*, then its second query would depend upon the response received to the first query, and so on.)

(ii). The verifier expects the input proof Π to be over some alphabet Σ. The size of Σ may depend on the input size. (If you wish, think of Π as a table, where each letter of Σ is a table entry containing $\log|\Sigma|$ bits which are read all at once.)

(iii). Given any integer $p \geq 1$ the verifier can check *split-encoded assignments* that are split into $p+1$ parts. This mode of operation is described below.

Prover[1] and verifier agree on an error-correcting code, i.e., a set of valid codewords $C \subseteq \Sigma^k$ with the property that if x and y are in C, then $d(x,y) \geq 0.5$. Here, k may depend on n.

The verifier also defines a mapping to encode binary strings into codewords:

$$\sigma: \ \{0,1\}^{n/p} \rightarrow C.$$

Then an assignment s is viewed as $s = s_1 s_2 \ldots s_p$, where each $s_i \in \{0,1\}^{n/p}$, and encoded as $z_1 \cdot z_2 \cdots z_p$, where each $z_i = \sigma(s_i)$.

The "proof" received by the verifier is of the form $z_1 \cdot z_2 \cdot \cdots \cdot z_p \cdot \pi$, where each z_i and π is a word in $\Sigma^\star$. (Note that the proof is actually just a string of bits—the verifier, by breaking it into substrings of $\log|\Sigma|$ bits can view it as a string

[1] By "prover" we mean the person constructing the proof of satisfiability.

over Σ.) If the prover is playing faithfully, each z_i is a word in Σ^k and is a codeword in C.

The verifier satisfies two conditions:

(C1) If each z_i is a codeword and $\sigma^{-1}(z_1)\ldots\sigma^{-1}(z_p)$ is indeed a satisfying assignment, then there is a π such that

$$\Pr[\ V \text{ accepts } z_1 \cdot \cdots \cdot z_p \cdot \pi\ ] = 1.$$

(C2) If either of the following is true, then the verifier rejects $z_1 \cdot \cdots \cdot z_p \cdot \pi$ with probability at most $1/2$, no matter what π is:

(C2.1) any one of $z_1 \ldots z_p$ is not δ-close to a codeword in C, for $\delta = 0.1$, or

(C2.2) all of $z_1, \ldots, z_p$ are δ-close to a codeword $\tilde{z}_i$ in C, but $\sigma^{-1}(\tilde{z}_1) \cdot \cdots \cdot \sigma^{-1}(\tilde{z}_p)$ is not a satisfying assignment.

In the verifiers we will actually construct, the code is just the set of degreee d polynomials (for an appropriate d) and ondition (C2.1) is essentially the Low Degree Test, checking that the z_i's are close to a polynomial.

2 The Composition Lemma

Let R, Q, and T be integer-valued functions.

Definition 1 *A normal form verifier is (R, Q, T)-restricted if on an input of n bits and given a a split-encoded assignment in p parts, it behaves as follows*

- *it reads the input and uses at most $R(n)$ random bits,*
- *performs a (possibly long) computation, deciding upon a sequence of $O(p \cdot Q(n))$ positions of Π to query,*
- *makes the queries and reads the corresponding $O(p \cdot Q(n))$ letters from Π, which it writes on the work tape,*
- *and finally, uses these letters and poly$(T(n))$ time to reach a decision (this is the time we called decision time before).*

Note that the number of bits actually read from Π is $O(p \cdot Q(n) \cdot \log |\Sigma|)$, and that this quantity must be less than $\text{poly}(T(n))$, since the verifier must read all this information in its decision time.

Lemma 4 (The Composition Lemma) *If verifier V_1 is (R, Q, T)-restricted and verifier V_2 is (R', Q', T')-restricted, then there is a verifier $V_1 \circ V_2$ that is (R'', Q'', T'')-restricted, where*
$R''(n) = R(n) + R'(poly(T(n)))$,
$Q''(n) = Q(n) \cdot Q'(poly(T(n)))$, *and*
$T''(n) = T'(poly(T(n)))$.

Proof. The basic idea of the proof is as follows. Let us fix the input of size n. (From now on, we systematically drop the argument n to all functions.) Recall that V_1 uses a random string to decide which bits of the proof to look at, reads those bits, and then decides in its decision time whether to accept or reject. Let $\Pi[i]$ denote the ith letter of the proof. Consider the action of verifier V_1 using a random string, say $w \in \{0,1\}^R$. The verifier uses w to compute a series of Q locations to read, $i_1(w)$, $i_2(w)$, ..., $i_Q(w)$, and then reads letters $\Pi[i_1(w)]$, ..., $\Pi[i_Q(w)]$ from Π onto its worktape. Then it performs a computation in poly(T) time to determine whether or not to accept.

The Cook-Levin theorem gives us a formula ψ_w such that: (A). It has one variable for each bit to be read from the proof, plus many other variables to describe V_1's computation. Thus we denote the formula as $\psi_w(y_1, \ldots, y_Q, y_0)$ where each y_i and y_0 is a block of variables. (B). Its size is poly(T) (C). The verifier accepts proof Π using random string w iff there is a[2] bit string ρ_w such that

$$\Pi[i_1(w)] \cdot \cdots \cdot \Pi[i_Q(w)] \cdot \rho_w \text{ is a satisfying assignment for } \psi_w. \tag{3}$$

The new verifier $V_1 \circ V_2$ is obtained by modifying V_1 as follows. Instead of reading $\Pi[i_1(w)], \ldots, \Pi[i_{Q+1}(w)]$, the new verifier uses the ability of verifier V_2 to check split-encoded assignments. Namely, the prover is asked to provide a "probabilistically checkable subproof" (in a split-encoded form using the code of verifier V_2) that condition (3) holds. Then the verifier uses V_2 as a subroutine (specifically, its ability to check split-encoded assignments in $Q+1$ parts) to check this subproof. We emphasize that the formula ψ_w (hence, the proof of its satisfiability) depends on the random bits w, but the prover does not know w ahead of time. So the prover has to provide a subproof for every choice of w.

Now we describe $V_1 \circ V_2$ more formally.

First we change V_1 and V_2 in a number of ways to make the discussion easier.

1. We assume *w.l.o.g.* that V_1 explicitly computes the boolean formula ψ_w. Furthermore, we assume it expects proof Π to contain an additional 2^R letters, where for $w \in \{0,1\}^R$ the wth letter is supposed to play the role of ρ_w. Verifier V_1,

[2] In case this is too mustifying, just think of ψ_w as a circuit representing the computation of the verifier on $\Pi[i_1(w)]$, ..., $\Pi[i_Q(w)]$. And ρ_w gives the settings of the internal gates of the circuit.

after reading $\Pi[i_1(w)], \ldots, \Pi[i_Q(w)]$ as before, reads the wth letter among these additional 2^R letters. read this letter ρ_w, and accepts at the end iff

$$\Pi[i_1(w)] \cdot \cdots \cdot \Pi[i_Q(w)] \cdot \rho_w \text{ is a satisfying assignment for } \psi_w. \tag{4}$$

Changing V_1 in this way increases the proof length by $2^R \cdot \text{poly}(T)$, but we do not care about proof size so far. Furthermore, the number of queries has increased to $Q+1$ from Q. We will denote the address of ρ_w by $i_{Q+1}(w)$, and let

$$a_w = \Pi[i_1(w)] \cdot \cdots \cdot \Pi[i_Q(w)] \cdot \Pi[i_{Q+1}(w)].$$

2. Padding if necessary, we assume that all locations of Π have the same bit-size.

We also change V_2 slightly. We assume that it accepts proofs for unsatisfiable formulas with probability below a small ϵ, instead of only $1/2$ as in condition (C2) on verifiers. To do this, execute V_2 about $O(\log(1/\epsilon))$ times and reject if any one of the runs fails. This only increases the resources used by a constant factor.

From now on, V_1 and V_2 always stand for these modified versions.

Now we can describe the proof format for $V_1 \circ V_2$. Let Y be the number of letters that V_1 expects in the proof, and let m be the number of variables in the formula ψ_w it will construct using any random string r. Then $V_1 \circ V_2$ will use V_2 to check a split-encoded assignment to ψ_w with $(Q+1)$ parts. Let $\sigma\colon\ \{0,1\}^{m/(Q+1)} \to \Sigma'$ be the encoding scheme of V_2.

Then $V_1 \circ V_2$ expects a proof Π_{new} of the form

$$\Pi_{new} = \sigma(\Pi[i]) \cdot \sigma(\Pi[2]), \ldots, \sigma(\Pi(Y)) \cdot \pi_1 \cdot \pi_2 \cdots \pi_{2^R},$$

where Π is some proof that would have been presented to V_1 and each π_i is also a word over the alphabet Σ'. Note that this new proof has $Y + 2^R$ words over the alphabet Σ'.

(Of course, that is what the new verifier *expects*; the prover could present it with garbage. But we show below that any garbage will be summarily rejected.)

How $V_1 \circ V_2$ works. $V_1 \circ V_2$ operates as follows. First it uses a random string $w \in \{0,1\}^R$ to simulate the actions of V_1 up to the point where it constructs formula ψ_w and computes its queries $i_1(w), \ldots, i_{Q+1}(w)$. Next, it uses the verifier V_2 to check that the following split-encoded assignment in $Q+1$ parts constitutes a proof of satisfiability for ψ_w:

$$\Pi_{new}[i_1(w)] \cdot \Pi_{new}[i_2(w)] \cdots \Pi_{new}[i_{Q+1}(w)] \cdot \pi_w.$$

Note that $V_1 \circ V_2$ does not read this entire split-encoded assignment — in fact, doing that would make $V_1 \circ V_2$ at least as inefficient as V_1! Instead, the verifier uses a random string w' (consisting of as many random bits as required by V_2) and lets V_2 decide upon a "few" queries in this split-encoded assignment. It reads only the corresponding bits in the proof. (See figures 1(a)-(b).)

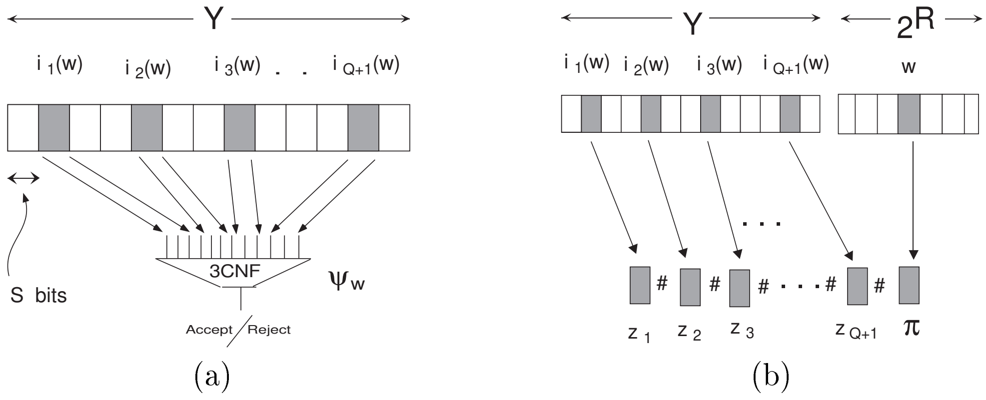


Figure 1: (a) Verifier V_1 expects a proof with Y symbols. Using $w \in \{0,1\}^R$ as a random string it queries locations $i_1(w), \ldots, i_{Q+1}(w)$. (b) Verifier $V_1 \circ V_2$ expects a proof with $Y + 2^R$ words. Using $w \in \{0,1\}^R$ it selects $Q+2$ words; these are shaded in the figure. These words are viewed as a $Q+1$-part split-encoded assignment $z_1 \sharp z_2 \sharp \cdots \sharp z_{Q+1} \sharp \pi$, which $V_1 \circ V_2$ checks using the normal-form verifier V_2.

How efficient is $V_1 \circ V_2$? By chugging through the numbers, we see that the various parameters for $V_1 \circ V_2$ are as claimed.

Proof of correctness. Now we show that $V_1 \circ V_2$ is a correct verifier for SAT.

First, suppose the input formula is satisfiable.

Then there is some proof Π which V_1 accepts with probability 1. Hence for each w, the bitstring $\Pi[i_1(w)] \cdot \Pi[i_2(w)] \cdots \Pi[i_{Q+1}(w)]$ is a satisfying assignment for ψ_w. By the definition of *checking split-encoded assignments*, for each w there is a string π_w such that V_2 accepts

$$\sigma(\Pi[i_1(w)]) \cdot \sigma(\Pi[i_2(w)]) \cdots \sigma(\Pi[i_{Q+1}(w)]) \cdot \pi_w$$

with probability 1. Hence by encoding each letter of Π with σ and adding 2^R new words to the proof consisting of these π_w's, we get a proof Π_{new} that $V_1 \circ V_2$ accepts with probability 1.

Now suppose the input formula is unsatisfiable.
Suppose Π_{new} is some proof consisting of $Y + 2^R$ words. Let p be the probability with which $V_1 \circ V_2$ accepts Π_{new}. We show $p \leq 1/2 + \epsilon/2$.

Note that the random string used by $V_1 \circ V_2$ is of the form (w, w') such that w is fed to V_1 and w' is fed to V_2. For every w let

$$r_w = \Pr_{w'}[V_1 \circ V_2 \text{ accepts using random string } (w, w')].$$

Note that if $r_w \geq \epsilon$ then the split-encoded assignment

$$\Pi_{new}[i_1(w)] \cdot \Pi_{new}[i_2(w)] \cdots \Pi_{new}[i_{Q+1}(w)] \cdot \pi_w$$

must consists of a proof of satisfiability for ψ_w. (This means in particular that $\Pi_{new}[i_1(w)], \ldots, \Pi_{new}[i_{Q+1}(w)]$ must be δ-close to some codewords.)

Now imagine constructing a proof Π for V_1 by decoding the first Y words of Π_{new}. In other words, if $\Pi_{new}[i]$ is δ-close to the encoding of some bitstring, say $\sigma(x)$, then make $\Pi[i] = x$. (If no such x exists, define $\Pi[i]$ arbitrarily.) Note that V_1 cannot accept such a proof Π with probability more than $1/2$. For any w, if V_1 rejects Π using w then $r_w \leq \epsilon$.

Hence the probability with which $V_1 \circ V_2$ accepts Π_{new} is at most

$$\frac{1}{2} + \frac{1}{2}\epsilon.$$

□

3 Verifiers Used for the PCP Theorem

Theorem 2 *There is a normal form verifier that is* $(\log n, 1, \mathit{poly}\log n)$*-restricted.*

Proof. This is the verifier that we described in the previous lecture, with some parts yet to be described. □

By the Composition Lemma we have:

Corollary 3 *There is a normal form verifier that is* $(\log n, 1, \mathit{poly}(\log\log n))$*-restricted.*

Proof. Just compose the verifier in Theorem 2 with itself. This affects the various parameters as follows.

1. *Random bits;* These go from $O(\log n)$ to $O(\log n) + \log(\log n) = O(\log n)$.

2. *Number of Queries:* These go from $O(1)$ to $O(1) \cdot O(1) = O(1)$.

3. *Decision time:* This goes from $\mathrm{poly}(\log n)$ to $\mathrm{poly}(\log(\mathrm{poly}(\log n))) = \mathrm{poly}(\log\log n)$.

Thus the corollary is proved. □

With a decision time as small as $\mathrm{poly}(\log\log n)$, it is possible to use a verifier using many random bits with respect to its input size:

Theorem 4 *There is a normal form verifier that is* $(n^3, 1, 1)$*-restricted.*

Proof. We only sketch the proof. It is complete in Sanjeev's thesis ([AArora:thesis], Section 4.3) and also in Sudan's thesis [sudan:thesis]

We claim the following: Given a 3SAT formula F with n variables and n clauses, we can build a set of n cubic functions in GF(2), $\{ \; p_i(X_{i_1}, X_{i_2}, X_{i_3}) \mid 1 \le i \le n \; \}$ such that $a = a_1 \ldots a_n \in \{0,1\}^n$ is a satisfying assignment for F if and only if

$$p_i(a_{i_1}, a_{i_2}, a_{i_3}) = 0 \quad \text{for all } 1 \le i \le n$$

So this is the condition we want to check reading $O(1)$ bits of the proof.

Use now the following fact:

> If $p_1, \ldots, p_n \in \mathrm{GF}(2)$ are not all zero, and $r_1, \ldots, r_n$ are picked uniformly at random from GF(2), then
>
> $$\Pr_{r_1 \ldots r_n}\left[\sum_{i=1}^{n} p_i \cdot r_i \neq 0 \right] = 1/2.$$

So the problem now is to compute $\sum_{i=1}^{n} p_i(a_{i_1}, a_{i_2}, a_{i_3}) \cdot r_i$. To do this, we ask the prover to provide as part of the proof the functions (on n, n^2, and n^3 variables, respectively):

$$\begin{aligned} f(x_1, x_2, \ldots, x_n) &= \sum_{i=1}^{n} a_i x_i \\ g(\{y_{ij}\}) &= \sum_{i,j=1}^{n} a_i a_j y_{ij} \\ h(\{z_{ijk}\}) &= \sum_{i,jk=1}^{n} a_i a_j a_k z_{ijk} \end{aligned}$$

Here $\{y_{ij}\}$ is a vector with n^2 coordinates $(y_{11}, y_{12}, \ldots, y_{1n}, y_{21}, y_{22}, \ldots, y_{2n}, \ldots,)$. (Note: The prover gives the functions in table form, so they are huge, about 2^{n^3}. However, we only count the decision time used by the verifier, after $O(1)$ entries have been read.)

Why do these functions help? Recall that each p_i is cubic. Then each term $p_i(a_{i_1}, a_{i_2}, a_{i_3}) \cdot r_i$, and in fact their sum, can be obtained as linear combination of the r_i's with coefficients of the form a_i, or $a_i a_j$, or $a_i a_j a_k$. The value of this linear combination is available as the sum of just three values, one from f, one from g, and one from h. So only three table entries in GF(2) have to be read.

Of course, we have to make sure with high probability that the tables given by the prover do correspond to the functions f, g, and h defined by the assignment a. We omit the description of this test in these notes. (The main idea is to do a low degree test over $GF(2)$.) Note only that the need for $O(n^3)$ random bits shows up when verifying h. $\square$

And composing the verifiers in Corollary 3 and Theorem 4, by the Composition Lemma again we have:

Corollary 5 *There is a normal form verifier that is* $(\log n, 1, 1)$*-restricted.*

And this immediately implies the PCP theorem.

4 The Sumcheck

In the last lecture, we argued that the verifier V, given a table of values describing a function $f : F^m \to F$, can be convinced that f is suitably close to a low-degree polynomial that represents a string of n bits. We now outline how V can further be convinced that the represented string satisfies some 3SAT instance ψ (wlog we assume ψ has n clauses and is over n variables). The basic idea is to reduce this question to determining if a certain multivariate polynomial sums to 0 on a domain. We will first outline how the reduction proceeds and then explain the work of the sumchecker.

Recall how we encoded n-bit boolean assignments by multivariate polynomials over fields. Given a field F, an integer $h < |F|$ and m such that $(h+1)^m = n$, we fix an embedding of $H = \{0, \ldots, h\}$ into F and a bijection between H^m and the set $\{1, \ldots, n\}$. We encode an assignment $a = (a_1, \ldots, a_n) \in \{0,1\}^n$ by a degree-mh polynomial A, in m variables, such that for every $u \in H^m$, $A(u) = a_u$. We will say that an m-variable polynomial of degree mh is a *satisfying polynomial* if the sequence of values $(A(v) : v \in H^m)$, ordered according to the bijection, is 0-1 valued and makes the 3CNF instance ψ true.

In our context, the satisfying polynomial will be provided by the prover. Given a candidate polynomial A, we can construct a new polynomial g that represents whether or not A is a satisfying polynomial. The main idea is that boolean operators can be replaced by addition and multiplication over the field. For example, if y_1, y_2, y_3 are $0/1$ valued field elements, then they satisfy the clause $y_i \vee \neg y_j \vee y_k$ iff $(1-y_i)y_j(1-y_k) = 0$. We will construct g by writing a single polynomial that implies the satisfiability of all the clauses.

For $j = 1, 2, 3$, let $\chi_j : H^m \times H^m \to \{0,1\}$ be defined by $\chi_j(c, v) = 1$ iff y_v is the j^{th} variable in clause c, and let $s_j : H^m \to \{0,1\}$ be defined by $s_j(c) = 1$ iff the j^{th} variable in clause c is unnegated. Note that A satisfies the formula iff

$$\prod_{j=1}^{3} \chi_j(z, w_j)(s_j(z) - A(w_j)) = 0 \qquad \forall z, w_1, w_2, w_3 \in H^m.$$

Now to obtain the desired polynomial g, we construct the polynomial extensions of χ_j's and s_j's. Denote these extensions by $\hat{\chi}_j : F^{2m} \to F$ and $\hat{s}_j : F^m \to F$. Note these polynomials have degree $2mh$ and mh respectively. Let

$$g(z, w_1, w_2, w_3) = \prod_{j=1}^{3} \hat{\chi}_j(z, w_j)(\hat{s}_j(z) - A(w_j)).$$

By construction, g is 0 at every point in H^{4m} iff A is a satisfying polynomial for ψ. Note that this is a polynomial of degree $9mh$ in $4m$ variables.

We now refer the reader to Sanjeev's thesis for the following. (The lemma is first proved in [BFLS-New].)

Lemma 5 ("Zero Tester Family of Polynomials") *Let F be a field of cardinality q. There exists a family of $q^{O(m)}$ polynomials $\{R_1, R_2, \dots\}$ in $F_{4mh}[x_1, \dots, x_{4m}]$ such that if $f : H^{4m} \to F$ is any function not identically 0, then if R is chosen randomly from this family,*

$$Pr[\sum_{y \in H^{4m}} R(y) f(y) = 0] \le \frac{1}{100}$$

This family is constructible in $q^{O(4m)}$ time.

Hence V can "see" that the above polynomial g is 0 on H^{4m} by selecting at random R_i in the family, and testing if $R_i \cdot g$ sums to 0 on H^{4m}. If g is identically 0, the sum will always be 0, if not, the sum will be non-zero with very high probability. Note that evaluating $R_i \cdot g$ at one point requires evaluating A at three points only.

It remains to solve the following problem: given a table of values of a degree-d polynomial B in l variables, a set $H \subseteq F$ and a value $c \in F$, to determine (by reading only 1 value of B !) if $\sum_{u \in H^l} B(u) = c$ or not. To do this, the verifier will receive from the prover a table containing for every $1 \le i \le l$ and every $u \in H^{i-1}$, a univariate polynomial $g_u(y_i)$ of degree d. In a good proof, it will be the case that $g_u(y_i) = \beta_u(y_i)$, where $\beta_u(y_i) = \sum_{v \in H^{l-i}} B(u, y_i, v)$.

In particular, for every a in F, $g_\epsilon(a)$ is the sum of the values $B(a, v)$ over all v in H^{l-1}. Note that if $f(y_1)$ is an "incorrect" degree-d polynomial, then $f(a)$ will differ from $\beta_\epsilon(a)$ on at least $q - d$ values of a, since two distint degree-d polynomials cannot agree on more than d points.

Consider the following procedure:

$current = c$
pick random $a_1, \dots, a_l$ in F, and let $u_i = (a_1, \dots, a_{i-1})$
For $i = 1$ to l do
  if $current \neq \sum_{y_i \in H} g_{u_i}(y_i)$
    output REJECT; exit
  else
    $current = g_{u_i}(a_i)$
If $g_{u_l}(a_l) \neq B(a_1, \dots, a_l)$

output REJECT
else
output ACCEPT

Obviously, if the sum $\sum_{u\in H^l} B(u) = c$, then the verifier, given the correct proof, will accept for every choice of $a_1, \dots, a_l$. Suppose now the sum is not equal to c: we argue that, for any table sent by the prover, the procedure will reject with high probability. If $l = 1$, either $g_\epsilon = B$ and g_ϵ does not sum to c (and is rejected with probability 1), or $g_\epsilon \neq B$ and the two polynomials disagree on at least $q - d$ points. Therefore $Pr_{a_1}[g_\epsilon(a_1) \neq B(a_1)] \geq 1 - d/q$. A similar argument proceeds inductively for $l > 1$. If $g_\epsilon = \beta_\epsilon$ then $\sum_{y_1\in H} g_\epsilon(y_1) \neq c$ and the input is rejected with probability 1. Otherwise, for $q - d$ values of a, we have $g_\epsilon(a) \neq B_\epsilon(a)$. Let a_1 be such a value: by the induction hypothesis, no information given by the prover can convince the procedure with probability more than $d(l-1)/q$ that $g_\epsilon(a_1)$ is the sum of $B(a_1, v)$ over all $v \in H^{l-1}$. Altogether the probability of rejection is thus at least $(1 - d/q)(1 - d(l-1)/q) \geq 1 - dl/q$. It is straightforward to see that the procedure runs in polynomial time, using $l \log q$ random bits.

5 Checking Split-Encoded Assignments

Let us see how a verifier can check assignments that are split into p parts for any integer p. Assume the 3SAT instance ψ has np variables split into p equal blocks. Recall from section 1 that prover and verifier have agreed on an alphabet Σ, an error-correcting code $C \subseteq \Sigma^k$ and a mapping $\sigma : \{0,1\}^n \to C$. An assignment $s \in \{0,1\}^{np}$ is then viewed as $s = s_1 \dots s_p$, each s_i in $\{0,1\}^n$, and encoded as $\sigma(s_1)\dots\sigma(s_p)$.

The verifier V thus expects the proof to be of the form $\sigma(s_1)\dots\sigma(s_p)\pi$,where $s = s_1 \dots s_p$ is a satisfying assignment, and π is some information allowing V to efficiently check that indeed s satisfies the 3SAT instance ψ, using the Sumcheck procedure of last section. The exact details will depend on the restriction imposed on the verifier. We will discuss here the case where V is $(\log n, poly(\log n), \log n)$-restricted.

In this situation, the verifier uses the encoding σ that maps strings in $\{0,1\}^n$ to their degree-mh polynomial extensions, as described in the last section. In other words the proof is required to contain, in table format, p functions $f_1, \dots, f_p : F^m \to F$, followed by some additional information. V first checks that each f_i is close to a low-degree polynomial. It then uses the same arithmetization of 3SAT instances that was described in the previous section, except that $A(w_j)$ is replaced by $\sum_{i=1}^{p} \lambda_{ij}(w_j)A_i(w_j)$, where $\lambda_{ij} : H^m \times H^m \to \{0,1\}$ is defined by $\lambda_{ij}(c,v) = 1$ iff the j^{th} variable in clause c is $y_{(i-1)n+v}$, and the A_i's are potential satisfying polynomials for the various blocks of variables.

References

[1] Arora, S., *Probabilistic Checking of Proofs and Hardness of Approximation*, Ph.D. Thesis, Computer Science Division, University of California, Berkeley, CA, USA. Revised version available as Technical Report CS-TR-476-94, Dept. of Computer Science, Princeton University, Princeton, NJ, USA. Anonymous ftp: ftp.cs.princeton.edu.

[2] Arora, S., Lund, C., Motwani, R., Sudan, M. and Szegedy, M., Proof verification and intractability of approximation problems. In *Proc. 33rd IEEE Symp. on Foundations of Computer Science*, 1992, 13–22.

[3] Arora, S. and Safra, S., Probabilistic checking of proofs. In *Proc. 33rd IEEE Symp. on Foundations of Computer Science*, 1992, 2–13.

[4] Babai, L., Fortnow, L., Levin, L. and Szegedy, M., Checking computations in polylogarithmic time. In *Proc. 23rd ACM Symp. on Theory of Computing*, 1991, 21–31.

[5] Rubinfeld, R. and Sudan, M., Testing polynomial functions efficiently and over rational domains. In *Proc. 3rd Annual ACM-SIAM Symp. on Discrete Algorithms*, 1992, 23–32.

[6] Sudan, M. *Checking of polynomials and proofs and Hardness of Approximation Problems.* Springer Verlag, 1996.

McGill/Barbados Workshop'96 **Randomness in Complexity Theory**

Lecture 5: Hardness of Approximations

Lecturer: *Sanjeev Arora* Notes: *Uwe Schöning and Carlos Zamora-Cura*

An *approximation* algorithm for an optimization problem is an algorithm that computes solutions whose cost is within a factor β of the optimal (where $\beta > 1$ is the *approximation ratio* of the algorithm). This definition is for *minimization* problems; for *maximization* problems the algorithm computes a solution of value at least $1/\beta$ times the value of the optimum solution. To give an example, achieving an approximation ratio β for *MAX-CLIQUE* means that we are able to compute, given any graph G, a clique of size at least $\omega(G)/\beta$, where $\omega(G)$ is the size of the largest clique in G (i.e., the *clique number* of G).

We already saw in Lecture 3 that the **PCP** theorem implies the hardness of approximating MAX-$3SAT$. In this lecture we will derive further consequences of the PCP theorem. We will use the hardness result for MAX-$3SAT$ proved in Lecture 3 to prove hardness results for other approximation problems, notably MAX-$3SAT(3)$ (each variable appears in at most 3 clauses), *CLIQUE*, and *CHROMATIC NUMBER*.

Further reading on this topic may be found in [AL96]. That paper also classifies known inapproximability results into four classes, based upon the approximation ratio that figures in the result.

1 Hardness of *MAX-3SAT*

In Lecture 3 the following theorem was proved.

Theorem 2 **NP** $=$ **PCP**$(\log n, 1)$ *if and only if the following polynomial-time reduction* $\tau : SAT \to MAX\text{-}3SAT$*, together with some* $\varepsilon > 0$*, exists:*

$$\forall\varphi : \quad \begin{array}{lcl} \varphi \in SAT & \Rightarrow & MAX\text{-}3SAT(\tau(\varphi)) = 1, \\ \varphi \notin SAT & \Rightarrow & MAX\text{-}3SAT(\tau(\varphi)) < \frac{1}{1+\epsilon}. \end{array}$$

(Here, MAX-$3SAT(\phi)$ *means the maximum fraction of clauses that can be satisfied by some assignment to the formula* ϕ *in 3-CNF.)*

The proof for the forward direction (which is the nontrivial direction here) goes as follows. From the assumption we have a polynomial-time verifier V for SAT with the following properties. On input a formula φ of size n it uses $O(\log n)$ many random bits, and depending on these random bits it specifies a constant k number of positions in the polynomially long proof $b_1 b_2 \dots b_{p(n)}$ that it wants to look at. After having seen these bits, the verifier accepts or rejects. Now, if φ is satisfiable, then there is a proof such that no matter what random bits V uses, it will accept. If φ is not satisfiable, then for no proof whatsoever the verifier will accept with higher probability than, say 1/2. Each of the polynomially many potential random choices r_i, $i = 1, \ldots, q(n)$, determines a constant number of bits in the proof where V wants to look at, say $b_{i1}, \ldots, b_{ik}$. For each $i = 1, \ldots, q(n)$ we can set up a truth-table T_i that represents the functional relation between the bits $b_{i1}, \ldots, b_{ik}$ and the acceptance behavior of the verifier V (having seen these bits). Each truth-table has $2^k = O(1)$ many lines and can be represented by a CNF formula F_i in the (respective constant size subset of the) variables $b_1, b_2, \ldots, b_p(n)$. (By a standard construction we may assume that F_i is in 3CNF). Now consider the conjunction $F_1 \wedge \cdots \wedge F_{q(n)}$ as the result of the desired reduction τ on input φ. If φ is satisfiable, then clearly MAX-3SAT$(F_1 \wedge \cdots \wedge F_{q(n)}) = 1$. If φ is not satisfiable, then under any assignment at least half of the F_i's will be false, that is, there is at least one clause in every such F_i that becomes false. Hence in this case, MAX-SAT$(F_1 \wedge \cdots \wedge F_{q(n)}) < \frac{1}{1+\varepsilon}$, for some $\varepsilon > 0$.

Therefore, by the fact that $\mathbf{NP} = \mathbf{PCP}(\log n, 1)$ is proved, we have the following

Corollary 1 *There is some $\beta > 1$ such that MAX-3SAT is hard to approximate within a factor of β. (The meaning of "hard" is "unless $\mathbf{P} = \mathbf{NP}$").*

PROOF. Suppose there is a polynomial-time approximation algorithm for *MAX-3SAT* that gives very good estimates for the maximum number of satisfiable clauses. On input φ apply this algorithm to the formula $\tau(\varphi)$ and get a very good estimate for the maximum number of clauses that can be satisfied. If this estimate is very close to 1, then φ must be satisfiable, otherwise φ is unsatisfiable. From the existence of this polynomial-time algorithm for *SAT*, it follows $\mathbf{P} = \mathbf{NP}$. ■

Next we will give a "gap-preserving" reduction $\tau : MAX\text{-}3SAT \to MAX\text{-}3SAT(3)$. This means that we transform the formula φ to a formula $\tau(\varphi)$ such that each variable has at most 3 occurrences. The property of this reduction is

$$\forall \varphi :$$
$$\begin{array}{rcl} MAX\text{-}3SAT(\varphi) = 1 & \Rightarrow & MAX\text{-}3SAT(\tau(\varphi)) = 1 \\ MAX\text{-}3SAT(\varphi) < \frac{1}{1+\varepsilon} & \Rightarrow & MAX\text{-}3SAT(\tau(\varphi)) < \frac{1}{1+\varepsilon'} \end{array}$$

for some $\varepsilon' > 0$.

Remark 1 *Such a restriction to a constant number of occurrences of each variable will later turn out to be useful, for other reductions.*

The reduction essentially goes back to [PY88] where completeness for the class MAXSNP is proved. Such MAXSNP-complete problems are all equivalent with respect to (non)existence of polynomial-time approximation schemes. See [AL96] for an introduction to this topic and [PY88] for further details.

PROOF. Suppose there are n variables in the given formula φ. Let m_i be the number of occurrences of the ith variable. Since φ is a 3-CNF formula, this means that it has $m = \sum_{i=1}^{n} m_i/3$ many clauses.

Classically one would introduce new variables for each occurrence of variable x_i, say $x_i^1, x_i^2, \ldots, x_i^{m_i}$, and write down clauses for the "circle of implications" $(x_i^1 \to x_i^2)$, $(x_i^2 \to x_i^3)$, ..., $(x_i^{m_i} \to x_i^1)$. But this construction is not 'robust' enough here because one might assign half of these variables the values true, and the other half the value false. This "consistency violation" might have the effect that many original clauses are set to true, but it might be only one of the $(x_i^j \to x_i^{j+1})$ clauses that becomes false. Therefore, even if φ was unsatisfiable, the fraction of simultaneously satisfiable clauses in the new clause set might become almost 1.

Instead, we need a construction such that it does not pay for an assignment to be inconsistent. The solution to the problem is to use a (version of) *expander graph*. This is an undirected graph $G = (V, E)$ based on the nodes $V = \{x_i^1, \ldots, x_i^{m_i}\}$ such that for every subset $S \subseteq V$, $|S| \le m_i/2$, we have $|E(S, \overline{S})| > |S|$. (Here, $E(S, \overline{S})$ is the set of edges that connect a node in S with a node in $\overline{S} = V - S$.) For each edge (x_i^j, x_i^k) in the expander graph G we introduce 2 clauses which are necessary to express the formula "$(x_i^j \leftrightarrow x_i^k)$".

We use the fact that there are efficient constructions of such expander graphs for any number n of vertices such that every node has bounded degree, namely 14 [LPS88,GG81,C91]. Therefore, our formula contains the m "original" clauses (with the jth occurrence of x_i replaced by x_i^j). Further, it contains $\sum_{i=1}^{n} 14m_i = 14 \cdot 3m = 42m$ clauses, based on the expander graph construction. The total number of clauses is therefore $43m$.

Suppose an optimal assignment (one that makes as many clauses true as possible) assigns some of the variables x_i^j (for fixed i) the value true (call this set S), and others the value false (call this set $\overline{S}$). Without loss of generality assume $|S| \le |\overline{S}|$, i.e. $|S| \le m_i/2$. We consider in this case false as the "correct" assignment for variable the x_i because false has the majority against true. The $|S|$ many "incorrect" assignments might turn $|S|$ many original clauses to true. Hence, from the inconsistency we have a gain of $|S|$ more clauses that become true. On the other hand, from the property of the expander graph, there are more than $|S|$ edges that connect S and $\overline{S}$. Therefore there are more than $|S|$ clauses that become false, by the inconsistency in the assignment. We conclude that it does not pay for an assignment to be inconsistent: the optimal assignment either assigns all x_i^j the value true, or all of them get the value false.

If φ is satisfiable, then clearly MAX-3SAT$(\tau(\varphi)) = 1$. If MAX-3SAT$(\varphi) < 1/(1+\varepsilon)$, then we get

$$\text{MAX-3SAT}(\tau(\varphi)) < \frac{42m + (\frac{1}{1+\varepsilon})m}{43m} \le \frac{1}{1+\varepsilon'}$$

for some $\varepsilon' > 0$.

We are not done yet. Each variable in $\tau(\varphi)$ occurs once in the original clauses and 28 times in the expander graph clauses. These 29 occurrences of each variable can be reduced to just 3 occurrences by adding the "classical" construction mentioned above. That is, let y_i denote a variable in $\tau(\varphi)$. It occurs in the formula exactly 29 times. Introduce the new variables $y_i^1, \ldots, y_i^{29}$ for each occurrence. Add the clauses $(\neg y_i^1 \vee y_i^2)$, $(\neg y_i^2 \vee y_i^3)$, ... ,$(\neg y_i^{29} \vee y_i^1)$. The constant ε' reduces to $\varepsilon'/29$, but each variable occurs now exactly 3 times. ■

2 Hardness of approximating the clique number

In this section we will prove that approximating the clique number of a graph, $\omega(G)$, is **NP**-hard. We will need the following lemma.

Lemma 1 *There is a polynomial reduction τ from* MAX3-SAT *to* CLIQUE *such that for all $\epsilon > 0$ and $c \in [0, 1]$,*

$$\forall\varphi :$$
$$\begin{aligned} MAX\text{-}3SAT(\varphi) = c \quad &\Rightarrow \quad \omega(\tau(\varphi)) = c\tfrac{N}{3} \\ MAX\text{-}3SAT(\varphi) < \tfrac{c}{1+\epsilon} \quad &\Rightarrow \quad \omega(\tau(\varphi)) < \tfrac{c\frac{N}{3}}{1+\epsilon} \end{aligned}$$

where N is the number of vertices in $\tau(\varphi)$.

PROOF. Let $\varphi = C_1 \wedge \cdots \wedge C_m$ be a *3CNF* with variables $x_1, \ldots, x_n$. Construct $\tau(\varphi)$ in the following way; represent each C_i by $G_i = (V_i, E_i)$ such that $V_i = \{x_{i1}, x_{i2}, x_{i3}\}$ and $E_i = \emptyset$. The nodes x_{i1}, x_{i2}, x_{i3} stand for the literals in C_i. We will join $x_{ik} \in V_i$ and $x_{jl} \in V_j$ (for $i \neq j$) if and only if the literals they represent are not negations of each other. The vertices of $\tau(\varphi)$ are the union of the V_i's and their edges are all the edges among the G_i's as defined above. Notice that a clique in this graph cannot contain two vertices representing negations of each other. With these observations we can conclude that any partial assignment of $x_1, \ldots, x_n$ satisfies as many clauses as there are vertices in the clique, thus for $\tau(\varphi)$ the conditions are true. ■

The hardness of *CLIQUE* relies on a *self-improvement* property which is defined by means of a graph product (this means that we present a reduction from *CLIQUE* to *CLIQUE* which increases the gap.) In this graph product we assume that any graph has all its self-loops, i.e., if $G = (V, E)$ let denote $\hat{E} = E \cup \{(u, u) \mid u \in V\}$ and consider then instead of G the graph $\hat{G} = (V, \hat{E})$.

Definition 2 *For two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ their* product $G_1 \times G_2$ *is the graph whose vertex set is $V_1 \times V_2$ and edge set is*

$$\{((u_1, v_1), (u_2, v_2)) \,:\, (u_1, u_2) \in \hat{E}_1 \text{ and } (v_1, v_2) \in \hat{E}_2\}.$$

There is an intuitive way to visualize the product of two graphs G_1, G_2. Suppose that $V_1 = \{u_1, \ldots, u_m\}$ and $V_2 = \{v_1, \ldots, v_n\}$. Take n copies of G_1, which we will call $H_1, \ldots, H_n$; and place them at the location of the vertices $v_1, \cdots, v_n$ respectively, in G_2. We join the vertex labeled u_i in H_j to the vertex labeled u_i in H_k if and only if $(v_j, v_k) \in \hat{E}_2$ and $(u_i, u_l) \in \hat{E}_1$. With this way of visualizing the graph product it is easily seen that $\omega(G_1 \times G_2) = \omega(G_1)\omega(G_2)$.

Now consider the graph $G^k = G \times \cdots \times G$ (k times). Then

$$\omega(G^k) = \begin{cases} m^k, & \omega(G) = m, \\ \left(\frac{m}{(1+\varepsilon)}\right)^k, & \omega(G) < \frac{m}{1+\varepsilon} \end{cases}$$

For having a polynomial-time reduction, we need here $k = O(1)$. Therefore, the strength of this gap-producing reduction is somewhat limited. Better bounds can be obtained by using the notion of booster product.

Definition 3 *A (k, n, α)* booster *is a collection $\mathcal{S}$ of subsets of $\{1, 2, \ldots, n\}$, each of size k, such that for every subset $A \subseteq \{1, 2, \ldots, n\}$, the sets in the collection that are subsets of A constitute a fraction between $(\rho - \alpha)^k$ and $(\rho + \alpha)^k$ of all sets in $\mathcal{S}$ where $\rho = \frac{|A|}{n}$. When $\rho < 1.1\alpha$, the quantity $(\rho - \alpha)^k$ should be considered to be 0.*

We will use the following result from [AFWZ93] about constructibility of booster graphs.

Theorem 3 *For any $k = O(\log n)$ and $\alpha > 0$, an (n, k, α) booster of size poly(n) can be constructed in poly(n) time.*

Given a graph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$ and a booster $\mathcal{S}$ we define their *booster product* to be constructed by taking as vertices the sets of $\mathcal{S}$, and there is an edge between sets $S_i, S_j \in \mathcal{S}$ if and only if $S_i \cup S_j$ is a clique in G. The following lemma is is easily proved from the definitions.

Lemma 2 *For any graph G, and any (k, n, α) booster, the clique number of the booster product of G lies between $(\frac{\omega(G)}{n} - \alpha)^k|\mathcal{S}|$ and $(\frac{\omega(G)}{n} + \alpha)^k|\mathcal{S}|$.*

PROOF. Take the parameters in the definitions as follows: $|\mathcal{S}| = poly(n) = n^c$, the gap in the clique number becomes $(1 - \varepsilon - \alpha)^k$. Therefore choose $\alpha \leq \varepsilon/10 \approx (1 + \varepsilon)^{\log n} = n^{\delta'} = (n^c)^{\delta'/c}$. ■

Now we have all the definitions and facts we need to prove that *CLIQUE* is hard to approximate.

Theorem 4 *There is an $\epsilon > 0$ such that approximating* CLIQUE *within a factor n^ϵ is* **NP***-hard, where n is the number of vertices in the input graph.*

PROOF. Take the graph G resulting from the reduction in lemma 1, and suppose that it has m vertices. The reduction guarantees that, for some fixed $\beta > 0$, that $\omega(G)$ is either at least $\frac{m}{3}$ or at most $m(1-\beta)/3$, and it is **NP**-hard to decide which case holds. Because of theorem 3 we can construct in polynomial time a $(m, \log m, \alpha)$ booster, call it $\mathcal{S}$. Choose $\alpha = \frac{\beta}{9}$, and build the booster product. Now, the clique number of the product is either at least $(\frac{3-\beta}{9})^{\log m}|\mathcal{S}|$ or at most $(\frac{3-2\beta}{9})^{\log m}|\mathcal{S}|$. Thus, the gap is now m^{γ} for some $\gamma > 0$, and further, $|\mathcal{S}| = \text{poly}(m)$, so this gap is $|\mathcal{S}|^{\epsilon}$ for some $\epsilon > 0$. ■

Compare this result with the best known approximation to the clique number which is $n/\log^2 n$.

3 Hardness of Chromatic Number

Sanjeev's talk finished by showing the **NP**-hardness of computing the chromatic number of a graph. The actual proof of this result is a bit long; so, we will present only the sequence of ideas and facts that prove the result without proof. The reduction presented here is based in the one presented in [KLS93].

Definition 4 *A* clique cover of size k *in a graph $G = (V, E)$ is a partition of the vertices into k sets, each of which is a clique. The* clique cover *of G, denoted by $\bar{\chi}(G)$, is the minimum size of a clique cover of graph G.*

Notice that the clique number of G is greater or equal to $\mid V(G) \mid / \bar{\chi}(G)$; because a k-coloring of G is a partition of the vertices into k independent sets then this k-coloring may be seen as a clique cover of size k in $\bar{G}$. Therefore $\chi(G) = \bar{\chi}(\bar{G})$, which shows that the clique cover number is a reformulation of the chromatic number.

In order to prove the **NP**-hardness of approximating the chromatic number of a graph, it is given a reduction from the R-CLIQUE problem, which is a generalization of the CLIQUE problem, to the clique cover problem. The R-CLIQUE problem takes as instances a number r and an r-partite graphs. The value of R-CLIQUE is $\omega(G)/r$. It is known that this problem is provably hard to approximate within a ratio $1/n^{\delta}$. The reduction is built for the following class of graphs.

Definition 5 *Given r and k positive integers and $r \times k$* graph *is a graph on rk vertices, which can be partitioned into r independent sets, each of size exactly k. We will assume that the vertices are labeled from the set $R \times K$ where $R = \{0, \ldots, r-1\}$, $K = \{0, \ldots, k-1\}$, and vertices labeled by $\{j\} \times K$ constitute the jth independent set.*

Given an r-partite graph it will be transformed into an $r \times k$ graph and then it is proven the existence of a reduction τ which takes an $r \times k$ graph and produces an $r \times k'$ such that for every constant $\epsilon > 0$:

$$\text{R-CLIQUE}(G) = 1 \Longrightarrow \bar{\chi}(\tau(\tau'(I))) = k'$$
$$\text{R-CLIQUE}(G) < 1/n^{\epsilon} \Longrightarrow \bar{\chi}(\tau(\tau'((I))) > N^{\epsilon/7}k',$$

where $n = rk$ and $N = rk'$ are the numbers of vertices in G and $\tau(G)$, I is an instance of SAT, and τ' is the reduction form SAT to R-CLIQUE. The **NP**-hardness follows once this reduction is composed with the one that proves the **NP**-hardness of R-CLIQUE.

The reduction τ consists of applying two transformations on the graph: the first makes it 2,3-*uniquely shiftable* and then produces the *extension* of the resulting graph. Given an $r \times k$ graph $G = (R \times K, E)$ we will use the following type of graphs: for $i = 0, \ldots, k-1$, define the i^{th} *copy* of G as the graph $G_i = (R \times K, E_i)$ containing the same number of vertices and edges as G, but whose edges are *shifted* according to the following rule:$((j,q),(j',q')) \in G_i(E)$ iff $((j,(q+i) \mod k),(j',(q'+i) \mod k)) \in G(E)$. Now we define the *extension* of an $r \times k$ graph.

Definition 6 *Let H be an $r \times k$ graph and $H_0, \ldots, H_{k-1}$ be its copies. The* extension *of H, denoted by $\tilde{H}$, is an $r \times k$ graph with the same number of vertices as H and whose edges are $\cup_{i=0}^{k-1} E(H_i)$.*

The following propositions suggests how the reduction could ensure the conditions we want for the reduction we are looking for.

Proposition 1 *If H is an $r \times k$ graph, then*

$$\textit{R-CLIQUE}(H) = 1 \Longrightarrow \bar{\chi}(\tilde{H}) = k.$$

Proposition 2 *Let H be an $r \times k$ graph such that $\omega(\tilde{H}) = \omega(H)$. then for every $\rho > 1$,*

$$\textit{R-QLIQUE}(H) < \frac{1}{\rho} \Longrightarrow \bar{\chi}(\tilde{H}) > k\rho$$

The problem now is that the previous proposition is not general enough to ensure the second condition for τ, because not every $r \times k$ graph G satisfies the property $\omega(G) = \omega(\tilde{G})$. however, even if G does not satisfy this property, it is shown how to convert it in polynomial time into an $r \times k'$ graph G' that does, where k' is roughly $(rk)^5$; further this new graph G' has the same clique number as G. Thus it follows from the previous propositions that

$$\text{R-CLIQUE}(G) = 1 \quad \Longrightarrow \quad \text{R-CLIQUE}(G') = 1 \quad \Longrightarrow \quad \bar{\chi}(\tilde{G'}) = k'$$

and

$$\text{R-CLIQUE}(G) = \frac{1}{\rho} \quad \Longrightarrow \quad \text{R-CLIQUE}(G') < \frac{1}{\rho} \quad \Longrightarrow \quad \bar{\chi}(\tilde{G}') = k'\rho.$$

The transformation from G to G' uses the following definition.

Definition 7 *For a given integer l, an $r \times k$ graph G is l* uniquely shiftable *if it satisfies that for every clique C in G of size l, there is a unique shift $i \in \{0, \ldots, k-1\}$ such that C is a clique in some copy G_i of G, and C is an independent set in every other copy $G_{i'}$ for $i \neq i'$. G is said to be 2,3* uniquely-shiftable *if it is both 2 uniquely-shiftable and 3 uniquely-shiftable.*

Note that if G is $2,3$ uniquely-shiftable then its extension $\widetilde{G}$ has certain special properties. For instance, 2 uniquely-shiftability implies that every edge e of $\widetilde{G}$ is obtained from exactly one edge of G, by using an appropriate shift i. (That is, for every two distinct edges e', e'' in G, the s th shift of e' does not coincide with the t th shift of e'' for every pair of shifts s, t.) Thus if G is 2 uniquely-shiftable, we may talk of this shift i as *the* shift of the edge e. If in addition G is 3 uniquely-shiftable, then in every triangle in $\widetilde{G}$, all three edges of the triangle have the same shift.

Lemma 3 *If an $r \times k$ graph G is 2,3 uniquely-shiftable, then $\omega(G) = \omega(\tilde{G})$.*

PROOF. Clearly, $\omega(G) \leq \omega(\widetilde{G})$, since a clique in G is a clique in $\widetilde{G}$. Now let C be any clique in $\widetilde{G}$ of size $\omega(\widetilde{G})$. Since G is 2 uniquely-shiftable, we know that every edge in C has a unique shift associated with it. We show that this shift is the same for all edges in C, whence it follows that C is just the shifted version of some clique in in G and the proof is complete. So assume the shifts are not all the same. Then there exist three vertices v_1, v_2, v_3 in C such that the edges (v_1, v_2) and (v_2, v_3) have different shifts. But this contradicts the assumption that G is 3 uniquely-shiftable, since $\{v_1, v_2, v_3\}$ is a triangle in $\widetilde{G}$. Hence it follows that every edge in C has the same shift. ■

It only remains to show that every $r \times k$ graphs can be made $2,3$ uniquely-shiftable without much increase in size. We will need the mapping defined in the following lemma.

Lemma 4 *For every positive integer m, we can in poly(m) time construct an injective map $T : \{0, 1, \ldots, m-1\} \to \{0, 1, \ldots, m'-1\}$, where $m' > 6m^5$ and for all distinct ordered triples (u_1, u_2, u_3) and (v_1, v_2, v_3) (i.e., $u_1 \leq u_2 \leq u_3$, $v_1 \leq v_2 \leq v_3$, and $(u_1, u_2, u_3) \neq (v_1, v_2, v_3)$),*

$$T(v_1) + T(v_2) + T(v_3) \quad \neq \quad T(u_1) + T(u_2) + T(u_3) \ \mathit{mod}\ m'. \tag{5}$$

PROOF. Construct T by greedily assigning values, in order, to $T(0), T(1), \ldots ,$. Assuming all values up to $T(i)$ have been assigned, we only have to ensure that $T(i+1)$ does not satisfy any of $6i^5$ equality constraints imposed by the values of $T(0), \ldots , T(i)$. Since $6i^5 < m'$, we can choose a value for $T(i + 1)$ easily, and continue on. ■

The next lemma shows how to make every $r \times k$ graph $2, 3$ uniquely-shiftable.

Lemma 5 *Every $r \times k$ graph G can be changed in polynomial time to an $r \times k'$ graph G' that is $2, 3$ uniquely-shiftable and satisfies $\omega(G) = \omega(G')$. Further, this is possible for $k' \leq 6(rk)^5$.*

PROOF. Transform G as follows. Choose a map T according to Lemma 4 by substituting $m = rk = |R \times K|$. Denote the integer m' obtained from the lemma as k', and let $W = \{0, 1, \ldots , k' - 1\}$. We will think of T as a function from $R \times K$, the vertices of G, to W. The vertex set of the new graph G' is $R \times W$, and its edges are defined according to the following rule: For $j_1, j_2 \in R$ and $w_1, w_2 \in W$, the vertices (j_1, w_1) and (j_2, w_2) are adjacent iff there is an edge $((j_1, q_1), (j_2, q_2))$ in G such that $T(j_1, q_1) = w$ and $T(j_2, q_2) = w_2$. In other words, G' is obtained by mapping every vertex (j, q) in G to the vertex $(j, T(j, q))$ in G', and putting in edges accordingly. Since T is injective, it follows that cliques in G and G' correspond to each other and so $\omega(G) = \omega(G')$.

To show that G' is 2-uniquely shiftable, we need to show that for every edge in $\widetilde{G'}$, the extension of G', there is a unique shift $i \leq k' - 1$ such that the edge belongs to the ith copy G'_i of G. Assume this is not the case. Then there are distinct edges e_1 and e_2 in G and distinct shifts $s, t \leq k' - 1$ such that the sth shift of e_1 coincides with the tth shift of e_2. Hence if $e_1 = \{(j_1, q_1), (j_2, q_2)\}$ and $e_2 = \{(j_1, q'_1), (j_2, q'_2)\}$ we have

$$\begin{aligned} (T(j_1, q_1) + s) &= (T(j_1, q'_1) + t) \\ (T(j_2, q_2) + s) &= (T(j_2, q'_2) + t) \end{aligned}$$

This implies that $T(j_1, q_1) - T(j_1, q'_1) = T(j_2, q_2) - T(j_2, q'_2)$ and hence that $T(j_1, q_1) + T(j_2, q'_2) + T(j_2, q'_2) = T(j_2, q_2) + T(j_1, q'_1) + T(j_2, q'_2)$. The property of T from Equation 5 implies that $(j_i, q_i) = (j_i, q'_i)$ for $i = 1, 2$, and thus e_1 and e_2 are the same edge. This is a contradiction. It follows that G' is 2-uniquely shiftable. Similarly, we can prove that G' is also 3 uniquely-shiftable. ■

References

[AFWZ93] N. Alon, U. Feige, A. Widgerson, and D. Zuckerman. *Derandomized graph products.*, Manuscript, 1993.

[AL96] S. Arora, and C. Lund. *Hardness of Approximation.* Approximation Algorithms for **NP**-hard problems, Dorit Houchbaum, ed., 1996, *to appear.*

[C91] F.R.K. Chung. Constructing random-like graphs. *Probabilistic Combinatorics and Its Applications.* B. Bollobás (Ed.), 21–55, Amer. Math. Soc. 1991.

[GG81] O. Gabber, Z. Galil. Explicit construction of linear sized superconcentrators. *Journal of Comput. Systems Sci.* 22:407–420, 1981.

[KLS93] S. Khanna, N. Linial, and S. Safra. On the hardness of approximating the chromatic number. In *Proc. of the 2nd Israel Symposium on Theory and Computing systems, ISTCS*, pages 250–260. IEEE Computer Society Press, 1993.

[LPS88] A. Lubotzky, R. Phillips, P. Sarnak. Ramanujan graphs, *Combinatorica*, 8:261–277, 1988.

[PY91] C. Papadimitriou and M. Yannakakis. Optimization, approximation and complexity classes. *J. of Computer and System Sciences* **43**, pp. 425–440, 1991. (Prelim. version in ACM STOC 1988.)

McGill/Barbados Workshop'96	Randomness in Complexity Theory
Lecture 6: Applications of graph exponentiation	
Lecturer: *Sanjeev Arora*	Notes: *Michael Saks*

By an exponentiation operator on graphs we mean a mapping $G \longrightarrow G^j$ that maps a graph G to a graph whose vertices are j-tuples or j-subsets of vertices of G. There are a variety of exponentiation operators. Many (but not all) arise in the obvious way by iterating some graph product operation.

Graph exponentiation is often used to *amplify* certain graph invariants. One example is the *clique number* of the graph. In Lecture 5 we used a graph exponentiation technique (which we called *booster product*) to show the hardness of approximating the clique number to within a factor n^ϵ for some $\epsilon > 0$.

In this lecture, two beautiful applications of graph exponentiation are described. The first shows that if there are polynomial time algorithms for determining whether a graph has a clique of logarithmic size then NTIME(t) has deterministic simulations that run in $2^{\tilde{O}(\sqrt{t})}$ time. The second shows that the Lovasz θ number of a graph (to be defined below) does not even approximate the clique cover number within a factor $n^{1-\epsilon}$ for any ϵ.

Throughout these notes, we denote the complementary graph of graph G — namely, the graph that contains exactly the edges that G does not contain — by $\bar{G}$. A *clique* of G is a (not necessarily maximal) subset of vertices that induces a complete subgraph. The *independence number*, $\alpha(G)$, is the size of the largest independent set of G. The *clique number*, $\omega(G)$ is the size of the largest subset of vertices that induces a clique. The *chromatic number* is denoted $\chi(G)$. The *clique cover number*, $\bar{\chi}(G)$, is the minimum k such that V can be covered by k cliques; without loss of generality we may assume that the sets are disjoint. Note that $\bar{\chi}(G) = \chi(\bar{G})$.

Finally, $\tilde{O}(t)$ is shorthand for $t \cdot \mathrm{poly}(\log t)$.

1 On the hardness of the LOG-CLIQUE problem.

The clique problem $CLIQUE(G, J)$ is the problem: given a graph G and integer J, does G have a clique of size J? The special case in which J is restricted to $\lceil \log_2 |V(G)| \rceil$ is the LOG-CLIQUE problem for G. This problem was studied by Papadimitriou and Yananakakis [PY] in the context of limited non-determinism and is complete for the class LOG-SNP of languages that are nondeterministically computable using $O(\log n)$ nondeterministic bits. Note that trivial bruteforce solves the LOG-CLIQUE problem in $n^{O(\log n)}$ time; no polynomial-time algorithm is known.

The following result of Feige and Killian provides some evidence for the belief that the LOG-CLIQUE problem is not in P:

Theorem 1.1 *[FK] If LOG-CLIQUE $\in$ P, then for any (suitably well behaved) function $t(n) \geq n$, $NTIME(t) \subseteq DTIME(2^{\tilde{O}(\sqrt{t})})$.*

A simplified proof of this result was given by Nisan. The proof uses the following exponentiation operator on graphs. For any integer $k \geq 1$, the exponentiation operator N_k on graphs where for graph $G = (V, E)$, $N_k(G)$ is the graph whose vertex set is the set $\binom{V}{k}$ of all k-sets of V and whose edge set is all pairs $\{U, W\}$ of k-sets such that $U \cap W = \emptyset$ and $U \cup W$ induces a clique. Trivially $N_k(G)$ has $\binom{|V|}{k}$ vertices and $\omega(N_k(G)) = \lfloor m/k \rfloor$.

Proof.

If LOG-CLIQUE $\in P$, then so is the problem $CLIQUE(G, J)$ under the restriction $J \leq \log_2 |V|$, since given G and J we can add about $\log_2 |V| - J$ vertices to G that are joined to every other vertex and produce a graph G' that has a clique of size $\log_2 |V(G')|$ if and only if G has a clique of size J.

Let L be a language in $NTIME(t)$ and x an instance. Using an "efficient" version of Cook's theorem ([Cook]) the question of whether x is in L can be reduced to a 3-SAT problem for a formula ψ_x with $m = O(t \log t)$ clauses and at most m variables. By a standard reduction, one constructs from ψ_x a graph G_x on $3m$ vertices has a clique of size m such that ψ_x is satisfiable if and only if G_x has a clique of size m. Assuming LOG-CLIQUE is in P, we will show that the last problem can be solved determinstically in $poly(N) = 2^{\tilde{O}(\sqrt{t})}$.

Assume wlog that m is a perfect square and set $k = \sqrt{m}$. Then $N_k(G)$ has $\binom{3m}{\sqrt{m}}$ vertices, which is greater than $2^{\sqrt{m}}$ and less than $2^{\tilde{O}(\sqrt{m})}$. Thus the problem $CLIQUE(G_x, m)$ is equivalent to the problem $CLIQUE(N_k(G_x), \sqrt{m})$. Since $\sqrt{m}$ is at most $\log_2 |V(N_k(G))|$, under the assumption that LOG-CLIQUE is in P, we can determine whether $N_k(G)$ has a clique of size $\sqrt{m}$ in time polynomial in the number of vertices of G, which is $2^{\tilde{O}(\sqrt{t})}$.

Remark: The theory of *fixed parameter intractability*[DF] studies, among other things, whether the $CLIQUE(G, J)$ can be solved in $n^{O(1)} \cdot f(J)$ time, where f is any function and J is any constant. This question is related to but not the same as "LOG-CLIQUE $\in$ P?", since f is allowed to grow arbitrarily fast. For example $f(J)$ could be 2^{2^J}, which is a constant for every fixed J but is 2^n for $J = \log n$.

2 The Lovasz θ number is not a good approximator for independence number

It is well known that the trivial inequality $\bar{\chi}(G) \geq \alpha(G)$ is in general very weak: for random graphs $\bar{\chi}(G)$ is $\Theta(n/\log n)$ (with high probability) while $\alpha(G)$ is $\Theta(\log n)$. Lovász [Lov2] introduced a parameter of graphs (defined below), $\theta(G)$, that satisfies $\bar{\chi}(G) \geq \theta(G) \geq \alpha(G)$. Moreover, $\theta(G)$ is approximable to arbitrary precision in polynomial time using the ellipsoid method [GLS]. It is natural to ask: how well does

$\theta(G)$ approximate $\alpha(G)$ and $\bar{\chi}(G)$? Again, for random graphs, $\theta(G) = \theta(\sqrt{n}\log n)$, and hence $\theta(G)/\alpha(G)$ can be as large as $\Omega(\sqrt{n})$, and $\bar{\chi}(G)/\theta(G)$ can be as large as $\tilde{\Omega}(\sqrt{n})$. We have the following two questions:

1. Is there an $\epsilon > 0$ such that $\frac{\theta(G)}{\alpha(G)} = O(n^{1-\epsilon})$?

2. Is there an $\epsilon > 0$ such that $\frac{\bar{\chi}(G)}{\theta(G)} = O(n^{1-\epsilon})$?

Lovász asked the first question with $\epsilon = 1/2$. Szegedy [Sze] showed that the two questions are equivalent. In this section, we present a result of Feige that shows that the answer to the second question (and hence also the first) is no.

Theorem 2.1 *[Fei] For any $\delta > 0$ there exists a constant $K = K(\delta)$ so that for any n' there is a graph G with $n > n'$ vertices such that $\theta(G) \leq K$ and $\bar{\chi}(G) \geq n^{1-\delta}$.*

There are various equivalent definitions of the θ number of G. The one we will use (see [KMS]) is based on an optimization problem. Given a graph $G = (V, E)$ we consider labelings of the vertices that assign to each vertex i a unit vector u^i in $\mathbf{R}^n$. Such a labeling is *C-admissible* for a positive integer C if for any two non-adjacent vertices i and j, the inner product of u^i and u^j is equal to $-C$. We define $\theta(G)$ as the infimum of all real numbers $k \geq 1$ for which there is a $1/(k-1)$-admissible labeling.

Geometrically, this means that we are trying to assign vectors to each vertex so that for non-adjacent vertices, the angle between them is "close" to 180°. To see that $\alpha(G) \leq \theta(G)$ observe that for any set of l unit vectors $v^1, v^2, \ldots, v^l$ in Euclidean space, the average of the inner products $\langle v^i, v^j \rangle$ for $i \neq j$ is at least $-1/(l-1)$, and hence if there is a $1/(k-1)$-admissible labeling then $k \geq l$.

On the other hand, for any $m \leq n$, one can find m vectors $w^1, w^2, \ldots, w^m$ in $\mathbf{R}^n$ such that the inner product of any two is exactly $-1/(m-1)$. Hence given a m-clique cover of G, if one assigns a distinct w^i to each clique in the cover, then the resulting cover is $1/(m-1)$-admissible and so $\theta(G) \leq \bar{\chi}(G)$.

The proof of Theorem 2.1 makes use of the $\otimes$ product of graphs, which is defined as follows. If G, H are graphs then $G \otimes H$ is the graph with vertex set $V(G) \times V(H)$, and whose edges are pairs $\{(v, w), (v', w')\}$ such that either $v = v'$ or $\{v, v'\} \in E(G)$ and either $w = w'$ or $\{w, w'\} \in E(H)$. (In other words, if $\hat{E}(G)$ denotes the set of edges with all self-loops adjoined, then $E(G \otimes H) = \hat{E}(G) \times \hat{E}(H)$.) The associated exponentiation operator is denoted $G^{\otimes j}$.

It is trivial to show that $\omega(G \otimes H) = \omega(G)\omega(H)$. Lovász [Lov1] showed that the θ number is also multiplicative:

Lemma 6 *For any two graphs G and H, $\theta(G \otimes H) = \theta(G)\theta(H)$.*

The clique cover number is not in general multiplicative. However, one can show that a quantity approximating the clique cover number is multiplicative. For a graph

G, let $\mathcal{C}(G)$ denote the set of maximal cliques of G. A *fractional clique cover* of G is nonnegative (weight) function w on $\mathcal{C}(G)$ with the property that for every vertex v, the sum of weights assigned to cliques containing v is at least 1. The sum of $w(C)$ over all cliques $C \in \mathcal{C}$ is the *total weight* of the cover. The *fractional clique cover number* of G, $\bar{\chi}_f(G)$, is defined to be the minimum total weight of any fractional clique cover. A clique cover of size k corresponds naturally to a 0-1 valued fractional clique cover of total weight k, and so $\bar{\chi}_f(G) \leq \bar{\chi}(G)$. On the other hand, Lovasz showed that one can bound $\bar{\chi}(G)$ from above:

Lemma 7 *For any graph G, $\bar{\chi}_f(G) \leq \bar{\chi}(G) \leq \bar{\chi}_f(G)(1 + \ln n)$.*

Here's a proof of a slightly weaker upper bound. Suppose we are given an optimal fractional cover that assigns weight w_C to clique C. Define $p_C = \min\{1, w_C \ln 2n\}$. Choose a random subset of cliques where each clique C is chosen independently with probability p_C. We show that (i) the probability that some vertex stays uncovered (i.e., not contained in some clique) is less than 1/2 and (ii) the probability that the number of cliques exceeds $\bar{\chi}_f(G)(2\ln(2n))$ is less than 1/2. This implies that with positive probability, we get a clique cover of size at most $\bar{\chi}_f(G)(2\ln(2n))$, and hence there must *exist* such a clique cover. To prove (i), note that for each vertex v, if $p_C = 1$ for some clique containing v, then v is covered with probability 1. Otherwise, the probability it is not covered is the product of $1 - p_C = 1 - w_C(\ln 2n) < e^{-\ln(2n)w_C}$ over all cliques C containing v. Since w is a fractional cover, $\sum_{C \ni v} w_C \geq 1$, and so this product is at most 1/2n. Hence the probability that some vertex is uncovered is less than 1/2. To see (ii), note that for any random variable the probability that it exceeds twice its expectation is less than 1/2 and the expected number E of cliques selected is at most $\ln(2n)\bar{\chi}_f(G)$.

The definition of $\bar{\chi}_f(G)$ is a linear program. The dual linear program is the fractional independent set problem: find the weighting of the vertices of G of minimum total weight so that the sum of the weight of the vertices in any clique is at most 1. This minimum is denoted $\alpha_f(G)$ and by the duality theorem of linear programming is equal to $\bar{\chi}_f(G)$.

While $\bar{\chi}(G)$ is not multiplicative, $\bar{\chi}_f(G)$ is:

Lemma 8 *$\bar{\chi}_f(G \otimes H) = \bar{\chi}_f(G)\bar{\chi}_f(H)$.*

To show that $\bar{\chi}_f(G \otimes H) \leq \bar{\chi}_f(G)\bar{\chi}_f(H)$, one checks that given a fractional clique cover w^G for G and w^H for H, then one gets a fractional clique cover for $G \otimes H$ by defining, for $C \in \mathcal{C}(G)$ and $C' \in \mathcal{C}(H)$, $w(C \times C') = w^G(C)w^H(C')$. To prove the reverse inequality, one uses a similar argument to the above to show that $\alpha_f(G \otimes H) \geq \alpha_f(G)\alpha_f(H)$.

Finally, we also need the following Chernoff-type large deviation inequality (see theorems A.12 and A.13 from [ASE]):

Lemma 9 *Let $X_1, X_2, \ldots, X_N$ be independent 0-1 valued random variables. Let $X = \sum_{i=1}^{N} X_i$ and let $M = E[X]$. Then for any positive real a, the probability that $X \geq M + a$ is at most $(\frac{e}{1+a/M})^a$ and the probability that $X \leq M - a$ is at most $e^{-a^2/2M}$.*

We can now proceed to Feige's proof. The starting point is a result by Karger, Motwani and Sudan (its proof uses a certain *Kneser* graph).

Theorem 2.2 *[KMS] There is an $\epsilon > 0$ such that for any n' there is a graph G on $n > n'$ vertices such that $\theta(G) = 3$ and $\bar{\chi}(G) \geq n^{\epsilon}$.*

Feige constructs the graph of Theorem 2.1 as follows. He exponentiates the graph G of Theorem 2.2 j times with the $\otimes$ product, to get the graph $G^{\otimes j}$. Then he takes a random subset of the vertices of $G^{\otimes j}$ and proves that with high probability the induced graph on those vertices satisfies the properties desired in Theorem 2.1.

Let $s_j = |V(G^{\otimes j})|$, and so $s_j = n^j$. Since the θ function is multiplicative, we have $\theta(G^{\otimes j}) = 3^j$. Hence $\bar{\chi}(G^{\otimes j}) \geq \bar{\chi}_f(G^{\otimes j}) = (\bar{\chi}_f(G))^j \geq (\bar{\chi}(G)/(1 + \ln n))^j \geq n^{j\epsilon/2} = s_j^{\epsilon/2}$ for all j sufficiently large.

We will show that for j sufficiently large (depending only on ϵ and not on n), $V(G^{\otimes j})$ has a subset W of size at least $t_j = s_j^{\epsilon/2}/2$ that has no cliques of size larger than jn. Let G' be the graph induced on W. Clearly, $\theta(G') \leq \theta(G^{\otimes j}) = 3^j$, and $\chi(\bar{G'}) \geq |W|/jn \geq |W|^{1-4/j\epsilon}$ (assuming n is sufficiently larger than j). Choosing $j \geq \frac{4}{\epsilon\delta}$ and setting $K = 3^j$ we obtain Feige's theorem.

So we now verify the existence of the desired W. Say that W is *good* if W has no cliques of size larger than jn and W is bad otherwise. It suffices to show that there is a good W with at least $s_j^{\epsilon/2}/2$ vertices. We will do this by selecting the vertices of W according to a random process and showing that with positive probability W is both good and has at least $s_j^{\epsilon/2}/2$ vertices.

Let y be the optimal fractional independent set of $G^{\otimes j}$, having total weight $\bar{\chi}(G^{\otimes j}) \geq s_j^{\epsilon/2}$. Select W at random by selecting each vertex v independently with probability y_v. For each $v \in V(G^{\otimes j})$, define the random variable X_v to be 1 if $v \in W$ and 0 otherwise. Then W is bad if and only if there is a clique C such that $\sum_{v \in C} X_v \geq jn + 1$. Trivially, we need only consider maximal cliques C of $G^{\otimes j}$ of size at least $jn + 1$. Let C be such a clique. Note that the expectation of $\sum_{v \in C} X_v = \sum_{v \in C} y_v \leq 1$, by the fact that y is a fractional independent set. Hence, applying Lemma 9, with $N = |C|$ and $a = jn$, we have that the probability that W contains at least $jn + 1$ vertices from C is less than $(\frac{e}{1+jn)})^j n$. The probability that W is bad is bounded above by this quantity times the number of maximal cliques of $G^{\otimes j}$. But every maximal clique of $G^{\otimes j}$ is a product of j maximal cliques of G, and so there are at most 2^{jn} such cliques. Thus the probability that G is bad is at most $(\frac{2e}{1+jn})^{jn}$ which is certainly less than 1/2.

Also, the expected size of W is $\bar{\chi}(G^{\otimes j}) \geq s_j^{\epsilon/2}$, and it is easy to show (again by Lemma 9) that the probability that W is less than $s_j^{\epsilon/2}/2$ is less than 1/2.

Hence, with positive probability, W is good and contains at least $s_j^{\epsilon/2}/2$ vertices, as required to prove the theorem.

References

[1] N. Alon, J. Spencer, and P. Erdös, *The Probabilistic Method*, John Wiley & Sons, Inc, 1992.

[2] S. Cook, "Short Propositional Formulas Represent Nondeterministic Computations", Information Processing Letters (26) 1988.

[3] R. Downey and M. Fellows, "Fixed parameter intractability", Proc. IEEE Conf. on Structure in Complexity Theory, (1992) 36-49.

[4] U. Feige, "Randomized graph products, chromatic numbers and the Lovasz θ-function", Proc. 27th ACM Symp. on Theory of Computing (1995) 635-640.

[5] U. Feige, J. Killian, "On limited versus polynomial nondeterminism", manuscript.

[6] M. Grötschel, L. Lovász and A. Schrijver, *Geometric algorithms and combinatorial optimization* Springer-Verlag , 1987.

[7] D. Karger, R. Motwani, M. Sudan, "Approximate graph coloring by semidefinite programming", Proc. 35th IEEE Symp. on Foundations of Computer Science (1994) 2-13.

[8] L. Lovász, "On the ratio of the optimal integral and fractional covers", Discrete Mathematics **13** (1975) 383-390.

[9] L. Lovász, "On the Shannon capacity of a graph", IEEE Trans. on Information Theory, **IT-25** No. 1 (1979) 1-7.

[10] C. Papadimitriou and M. Yannakakis", "On Limited Nondeterminism and the Complexity of the V-C Dimension (Extended Abstract)", SCT: Annual Conference on Structure in Complexity Theory", (1993).

[11] M. Szegedy, "A note on the θ number of Lovasz and the generalized Delsarte bound", Proc. 35th IEEE Symp. on Foundations of Computer Science (1994) 36-39.