CSE 533: The PCP Theorem and Hardness of Approximation **(Autumn 2005)**

Lecture 1: Introduction

Sep. 28, 2005

Lecturer: Ryan O'Donnell *Scribe: Ryan O'Donnell*

1 Proof, and The PCP Theorem

The PCP Theorem is concerned with the notion of "proofs" for mathematical statements. We begin with a somewhat informal definition:

Definition 1.1. *A* traditional proof system *works as follows:*

- *A statement is given. (e.g., "this graph $G = \cdots$ is 3-colorable" or "this CNF formula $F = (x_1 \vee \overline{x}_2 \vee \overline{x}_{10}) \wedge \cdots$ is satisfiable").*
- *A* prover *writes down a proof, in some agreed-upon format.*
- *A* verifier *checks the statement and the proof, and accepts or rejects.*

From the perspective of theory of computer science, we usually fix a constant size alphabet and assume the statement and proof are strings over this alphabet; we also usually write n for the length of the statement and measure lengths of strings and running times in terms of n. The familiar complexity class NP can be cast in this setup:

Remark 1.2. *Language L is in NP iff there is a* polynomial time *deterministic verifier V (a Turing Machine, say) and an arbitrarily powerful prover P, with the following properties:*

- *"Completeness": For every $x \in L$, P can write a proof of length $\mathrm{poly}(|x|)$ that V accepts.*
- *"Soundness": For every $x \notin L$, no matter what $\mathrm{poly}(|x|)$-length proof P writes, V rejects.*

To equate the notion of the verifier being efficient with it being a deterministic polynomial time algorithm is nowadays a bit quaint; ever since the late '70s we have been quite happy to consider randomized polynomial time algorithms to be efficient. As it turns out, when proof systems are allowed to have randomized verifiers, some very surprising things can happen. This line of research was begun in the early-to-mid '80s by Goldwasser, Micali, and Rackoff [9] and also independently by Babai [3, 4]. See the accompanying notes on the history of the PCP theorem. One pinnacle of research in this area is *The PCP Theorem*:

Theorem 1.3. *(due to Arora-Safra (AS) [2] and Arora-Lund-Motwani-Sudan-Szegedy (ALMSS) [1])* "The PCP (Probabilistically Checkable Proof) Theorem"*:*

All languages $L \subseteq NP$ have a P.C.P. system wherein on input $x \in \{0,1\}^n$:

- *Prover P writes down a $\mathrm{poly}(n)$-bit-length proof.*
- *Verifier V looks at x and does polynomial-time deterministic computation. Then V uses $O(\log n)$ bits of randomness to choose C random locations in the proof. Here C is a absolute universal constant; say, 100. V also uses these random bits to produce a deterministic test (predicate) ϕ on C bits.*
- *V reads the bits in the C randomly chosen locations from the proof and does the test ϕ on them, accepting or rejecting.*
- *Completeness: If $x \in L$ then P can write a proof that V accepts with probability 1.*
- *Soundness: For every $x \notin L$, no matter what proof P writes, V accepts with probability at most $1/2$.*

Remark 1.4. *This P.C.P. system has "one-sided error": true statements are always accepted, but there is a chance a verifier might accept a bogus proof. Note that this chance can be made an arbitrarily small constant by naive repetition; for example, V can repeat its same spot-check 100 times independently, thus reading $100C$ bits and accepting false proofs with probability at most 2^{-100}.*

The first time one sees this theorem, it seems a little hard to conceive how it can be true. It's even more striking when one learns that essentially C may be taken to be 3. (See Theorem 1.12 below.) How could you possibly be convinced a proof is true just by spot-checking it in 3 bits?

Remark 1.5. *By the classical theory NP-completeness, it suffices to prove the PCP Theorem for one particular NP-complete language — say, 3-COLORING or 3-SAT — since poly-time reductions can be built into the verifier's initial step (and into the prover's plans).*

Remark 1.6. *The PCP that the prover needs to write down can be obtained in deterministic polynomial time from the "standard" proofs for $x \in L$ (i.e., the coloring for 3-COLORING, the assignment for 3-SAT).*

Remark 1.7. *Sometimes enthusiastic descriptions of the PCP Theorem make it seem like it greatly reduces the* time *a verifier needs to spend to check a proof. This is not accurate since the verifier still dies polynomial-time deterministic pre-computations; these may already take more time then it would have taken to simply check a classical proof. What the PCP Theorem saves is* proof accesses. *There is other work on developing proof systems that let the verifier save on time or space (see the accompanying notes on the history of the PCP Theorem); however it seems to have had fewer interesting applications.*

Our first task in this course will be to prove Theorem 1.3 completely. The fact that this will be possible is only due to a very recent development. The original proof of the PCP Theorem was very intricate and difficult; it might have been up to 100 pages, with subsequent simplifications bringing it down to a very densely packed 30 pages or so. However in April 2005, Irit Dinur gave a new proof [5] which is elegant and clear and only a dozen pages or so long. This is the proof we

will see in the course.

Subsequent to the PCP Theorem were many more "PCP Theorems" that strengthened certain parameters or extended the result in different directions. What follows are a few of these:

Theorem 1.8. *(Feige-Kilian [8], Raz [17]) (Raz's strong version of this result is sometimes called "the Raz Verifier" or "hardness of Label Cover"): For every constant $\epsilon > 0$, there is a poly-size PCP for NP that reads* two *random proof entries written with $O(1)$-size alphabet and has completeness 1, soundness ϵ.*

Remark 1.9. *Note that in this theorem, the poly-size of the PCP and the alphabet size both depend on ϵ; with Raz's version, the proof has length $n^{O(\log 1/\epsilon)}$ and the alphabet has size $\mathrm{poly}(1/\epsilon)$.*

Remark 1.10. *The result proven is actually stronger in a technically subtle but important way: One can additionally have the verifier use a predicate $\phi(x, y)$ with the "projection property", namely, that for every choice of x there is exactly one choice of y that makes $\phi(x, y)$ true.*

Remark 1.11. *Comparing this result to the basic PCP Theorem, we see that it uses a constant-size alphabet and two queries to get arbitrarily small constant soundness, whereas the basic PCP Theorem uses constantly many queries to a size-two alphabet. It might not be immediately clear which is better, but it is indeed the former. There are several ways to look at this: For example, with fewer queries you have fewer opportunities to "cross-check"; as an extreme, it's clear that a verifier that made only one query (to a constant size alphabet) could always be fooled. Or suppose that you tried to encode every triple of bits in a proof with a single character from an alphabet of size 8 — although you could now read three bits with just one query, the prover can cheat you by encoding a single bit in different ways in different triples.*

We hope to prove Theorem 1.8 in this course — at least, the Feige-Kilian version without the projection property.

The following result essentially shows that we can take $C = 3$ in the original PCP Theorem:

Theorem 1.12. *(Håstad [12]) "3-LIN hardness": For every constant $\epsilon > 0$, there is a poly-size PCP for NP that reads just* three *random* bits *and tests their XOR. Its completeness is $1 - \epsilon$ and its soundness is $1/2 + \epsilon$.*

Remark 1.13. *This result has "imperfect completeness". However, if one is willing to allow an* adaptive *three-bit-querying verifier (i.e., the verifier does not have to pick the three bits in advance but can base what bit it reads next on what it's seen so far) then one can get completeness 1. This is due to Guruswami, Lewin, Sudan, and Trevisan [10].*

This result, which we will prove in the course, requires Theorem 1.8.

Finally, here is one more PCP Theorem which we *won't* prove:

Theorem 1.14. *(due to Dinur [5], based heavily on a result of Ben-Sasson and Sudan [**?**]): In the basic PCP Theorem, the proof length can be made $n \cdot \mathrm{polylog}(n)$ rather than $\mathrm{poly}(n)$.*

1.1 Hardness of approximation

Perhaps the most important consequence of the PCP theorems and the most active area of research in the area are results about "hardness of approximation". These will be the major focus of the second half of this course. To be able to state hardness of approximation results, we need to understand the notion of *(NP) combinatorial optimization problems*. Instead of making a formal definition we will just give some examples. Briefly, these are "find the best solution" versions of classic NP-complete problems.

Definition 1.15. *MAX-E3SAT: Given an E3CNF formula — i.e., a conjunction of "clauses" over boolean variables $x_1, \ldots, x_n$, where a clause is an OR of exactly 3 literals, x_i or $\overline{x_i}$ — find an assignment to the variables satisfying as many clauses as possible.*

Definition 1.16. *SET-COVER: Given a bunch of sets $S_1, \ldots, S_m \subseteq \{1, \ldots, n\}$, find the fewest number of them whose union covers all of $\{1, \ldots, n\}$. (We assume that every ground element i is in at least one set S_j.)*

Definition 1.17. *MAX-CLIQUE: Given an undirected graph, find the largest clique in it, where a clique is a subset of vertices which contain all possible edges.*

Definition 1.18. *KNAPSACK: Given are "weights" $w_1, \ldots, w_n \geq 0$ of n items and also "values" $v_1, \ldots, v_n \geq 0$. Also given is a "capacity" C. Find a set of items S such that $\sum_{i \in S} w_i \leq C$ while maximizing $\sum_{i \in S} v_i$.*

Remark 1.19. *Each of these is associated to a classic NP-complete decision problem; e.g., "CLIQUE: Given G and k, does G have a clique of size at least k?" Notice that frequently the NP decision problem is a contrived version of the more natural optimization problem.*

Remark 1.20. *Combinatorial optimization problems can be divided into two categories: Maximization problems (like MAX-3SAT, MAX-CLIQUE, KNAPSACK) and minimization problems (like SET-COVER).*

It is well-known that these problems are all NP-hard. However, suppose that for, say, MAX-E3SAT, there was a polynomial time algorithm with the following guarantee: Whenever the input instance has optimum OPT — i.e., there is an assignment satisfying OPT many clauses — the algorithm returns a solution satisfying $99.9\% \times OPT$ many clauses. Such an algorithm would be highly useful, and would tend to refute the classical notion that the NP-hardness of MAX-E3SAT means there is no good algorithm for it.

Indeed, such results are known for the KNAPSACK problem. As early as 1975, Ibarra and Kim [14] showed that for every $\epsilon > 0$ there is an algorithm for KNAPSACK that runs in time $\mathrm{poly}(n/\epsilon)$ and always returns a solution which is within a $(1-\epsilon)$ factor of the optimal solution. So, although KNAPSACK is NP-complete, in some sense it's very easy. Let us make some definitions to capture these notions:

Definition 1.21. *Given a combinatorial optimization* maximization *problem, we say algorithm A is an α-*approximation algorithm *(for $0 < \alpha \leq 1$) if whenever the optimal solution to an instance*

has value OPT*,* A *is guaranteed to return a solution with value at* least $\alpha \cdot \mathrm{OPT}$*. We make the analogous definition for* minimization *problems, with* $\alpha \geq 1$ *and* A *returning a solution with value at* most $\alpha \cdot \mathrm{OPT}$*. Unless otherwise specified, we will also insist that* A *runs in polynomial time.*

Remark 1.22. *Our definition for maximization problems is sometimes considered unconventional; some like to always have* $\alpha \geq 1$*, in which case their notion is that the algorithm* A *returns a solution with value at least* OPT/α*.*

Definition 1.23. *A maximization (resp., minimization) combinatorial optimization problem is said to have a* PTAS *(Polynomial Time Approximation Scheme) if it has a* $(1-\epsilon)$*-approximation algorithm (resp.,* $(1+\epsilon)$*-approximation algorithm) for every constant* $\epsilon > 0$*.*

As mentioned, the KNAPSACK problem has a PTAS, and this is true of certain other combinatorial optimization problems, mostly related to scheduling and packing. But what about, say, MAX-E3SAT? It is a remarkable consequence of the PCP Theorem that MAX-E3SAT has no PTAS unless $\mathrm{P} = \mathrm{NP}$. In fact, the two statements are basically equivalent!

Theorem 1.24. *(credited to an unpublished 1992 result of Arora-Motwani-Safra-Sudan-Szegedy): Speaking roughly, the PCP Theorem is equivalent to the statement, "MAX-E3SAT has no PTAS assuming* $\mathrm{P} \neq \mathrm{NP}$*".*

We will precisely formulate and prove this theorem in the next lecture.

Indeed, most work in the PCP area these days is centered on proving "hardness of approximation" results like Theorem 1.24. We will state here a few striking "optimal hardness-of-approximation results" that have followed from work on PCP theorems.

Theorem 1.25. *(follows from Håstad's Theorem 1.12): MAX-E3SAT has no* $(7/8+\epsilon)$*-approximation algorithm for any constant* $\epsilon > 0$ *unless* $\mathrm{P} = \mathrm{NP}$*.*

We will see the proof of this in the course.

Remark 1.26. *There is a very easy* $7/8$*-approximation algorithm for MAX-E3SAT: Just picking a random assignment gives a* $7/8$*-approximation in expectation, and this algorithm is easily derandomized.*

Regarding SET-COVER:

Theorem 1.27. *(Feige [6]): SET-COVER has no* $(1-\epsilon)\ln n$ *approximation algorithm for any constant* $\epsilon > 0$ *unless* $\mathrm{NP} \subseteq \mathrm{DTIME}(n^{\log\log n})$*.*

Remark 1.28. *The greedy algorithm achieves a* $(\ln n + 1)$*-approximation algorithm. (Johnson [15])*

Remark 1.29. *The fact that we have the conclusion "unless* $\mathrm{NP} \subseteq \mathrm{DTIME}(n^{\log\log n})$*" is due to technical difficulties; however since the conclusion is almost as unlikely as* $\mathrm{NP} = \mathrm{P}$*, we don't really mind much.*

We won't prove Feige's theorem about SET-COVER in this course but we will prove a due to Lund and Yannakakis [16], that shows hardness of giving a $\Omega(\log n)$-approximation.

The situation for MAX-CLIQUE is direst of all:

Theorem 1.30. *(due to Håstad [11], with a significant simplification by Samorodnitsky-Trevisan [18], another simplification by Håstad and Wigderson [13], and a slight improvement by Zuckerman in September 2005 [19]): MAX-CLIQUE has no $(1/n^{1-\epsilon})$-approximation for any constant $\epsilon > 0$ unless $\mathrm{P} = \mathrm{NP}$.*

Remark 1.31. *There is a trivial $1/n$-approximation: Output a single vertex.*

We won't prove this theorem in the course, but the weaker result that $(1/n^{\Omega(1)})$-approximating is hard follows relatively easily from the main PCP Theorem, via a reduction given by Feige-Goldwasser-Lovasz-Sudan-Szegedy [7]. We will give this reduction later in the course.

References

[1] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, 1998.

[2] S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of NP. *J. ACM*, 45(1):70–122, 1998.

[3] L. Babai. Trading group theory for randomness. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, STOC'85 (Providence, RI, May 6-8, 1985)*, pages 421–429, New York, 1985. ACM, ACM Press.

[4] L. Babai and S. Moran. Arthur-Merlin games: A randomized proof system, and a hierarchy of complexity classes. *Journal of Computer and System Sciences*, 36(2):254–276, Apr. 1988.

[5] I. Dinur. The pcp theorem by gap amplification. ECCC, TR05-046, 2005.

[6] U. Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

[7] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Interactive proofs and the hardness of approximating cliques. *J. ACM*, 43(2):268–292, 1996.

[8] U. Feige and J. Kilian. Two-prover protocols — low error at affordable rates. *SIAM J. Comput.*, 30(1):324–346, 2000.

[9] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput*, 18(1):186–208, 1989.

[10] V. Guruswami, D. Lewin, M. Sudan, and L. Trevisan. A tight characterization of NP with 3 query PCPs. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science(FOCS-98)*, pages 8–17, Los Alamitos, CA, Nov.8–11 1998. IEEE Computer Society.

[11] J. Håstad. Clique is hard to approximate to within $n^{1-\epsilon}$. *Acta Mathematica*, 182:105–142, 1999.

[12] J. Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, 2001.

[13] J. Håstad and A. Wigderson. Simple analysis of graph tests for linearity and PCP. *Random Struct. Algorithms*, 22(2):139–160, 2003.

[14] O. H. Ibarra and C. E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the ACM*, 22(4):463–468, Oct. 1975.

[15] D. S. Johnson. Approximation algorithms for combinatorial problems. *J. Comput. Syst. Sci*, 9(3):256–278, 1974.

[16] C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. *J. ACM*, 41(5):960–981, 1994.

[17] R. Raz. A parallel repetition theorem. *SIAM J. Comput*, 27(3):763–803, 1998.

[18] A. Samorodnitsky and L. Trevisan. A PCP characterization of NP with optimal amortized query complexity. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, STOC'2000 (Portland, Oregon, May 21-23, 2000)*, pages 191–199, New York, 2000. ACM Press.

[19] D. Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. ECCC, TR05-100, 2005.

CSE 533: The PCP Theorem and Hardness of Approximation **(Autumn 2005)**

Lecture 2: PCP Theorem and GAP-SAT; intro to expanders

Oct. 3, 2005

Lecturer: Ryan O'Donnell and Venkatesan Guruswami *Scribe: Atri Rudra*

1 The GAP-SAT problem and the PCP theorem

In the last lecture we said we will prove NP-hardness of approximation algorithms. To do this, we will follow the same approach that is used in NP-completeness — we convert an optimization problem into a decision problem. For a concrete example recall the $\mathrm{MAX\text{-}3ESAT}$ problem: each clause of the input boolean formula is an OR of exactly three literals and the goal is to find an assignment that maximizes the number of satisfied clauses. We now define the decision version of this problem.

Definition 1.1. $\mathrm{GAP\text{-}E3SAT}_{c,s}$ $(0 < s \leq c \leq 1)$*: Given an* $\mathrm{E3SAT}$ *formula on* m *clauses,*

- *output YES if* $OPT \geq cm$*;*
- *output NO if* $OPT < sm$*;*
- *output anything if* $sm \leq OPT < cm$*.*

Remark 1.2. *Recall that* $\mathrm{GAP\text{-}E3SAT}$ *being NP-hard means that there is a deterministic polynomial time reduction,* R*, from your favorite NP-complete language (say* $3\text{-}\mathrm{COLOR}$*) to* $\mathrm{E3SAT}$*, such that*

- *Completeness: given* $G \in 3\text{-}\mathrm{COLOR}$*,* $R(G)$ *gives an* $\mathrm{E3SAT}$ *formula with* $OPT \geq cm$*;*
- *Soundness: given* $G \notin 3\text{-}\mathrm{COLOR}$*,* $R(G)$ *gives an* $\mathrm{E3SAT}$ *formula with* $OPT < sm$*.*

Remark 1.3. $\mathrm{GAP\text{-}E3SAT}_{c,s}$ *being NP-hard implies that there is no polynomial time* $\left(\frac{s}{c}\right)$*-factor approximation algorithm for* $\mathrm{MAX\text{-}3ESAT}$ *unless* $\mathrm{P} = \mathrm{NP}$*. To see this implication, assume we have such an algorithm; we then show how to solve* $3\text{-}\mathrm{COLOR}$ *in polynomial time. To do this, given a graph* G *apply the reduction* R *reducing it to* $\mathrm{E3SAT}$ *and then run the supposed* $\left(\frac{s}{c}\right)$*-factor approximation algorithm. If* $G \in 3\text{-}\mathrm{COLOR}$ *then this will produce an assignment that satisfies at least* $\left(\frac{s}{c}\right) cm = sm$ *clauses in* $R(G)$*. If* $G \notin 3\text{-}\mathrm{COLOR}$*, the algorithm will be unable to produce an assignment that satisfies as many as* sm *clauses of* $R(G)$*. Thus, we can distinguish the two cases and get a polynomial time algorithm for* $3\text{-}\mathrm{COLOR}$*.*

In this lecture, we will show the following result.

Theorem 1.4. *The following two statement are equivalent:*

1. *The PCP theorem;*

2. *There exists an universal constant $s < 1$ such that* $\mathrm{GAP\text{-}E3SAT}_{1,s}$ *is* NP*-hard.*

Proof. We first show that the second statement implies the first. To this end assume that $\mathrm{GAP\text{-}E3SAT}_{1,s}$ is NP-hard. We will construct a PCP system for 3-COLOR. Given an instance G of 3-COLOR on n vertices, the verifier V runs the reduction from 3-COLOR to E3SAT — let ψ be the constructed formula on m clauses. The proof that the prover P will provide will be an assignment to the variables in ψ (note that ψ has polynomial (in n) many variables). Finally, V uses $\log m = O(\log n)$ random bits to choose a random clause (call this clause ϕ), queries the proof on the variables in the clause, and checks if the given assignment satisfies ϕ. Note that the number of positions probed here, C, is 3. We now show that the above PCP system has the required properties.

- Completeness: If $G \in$ 3-COLOR then ψ has $OPT = m$. In this case P can write down the optimal assignment, which implies that all the clauses are satisfied, and hence V accepts with probability 1.

- Soundness: If $G \notin$ 3-COLOR then ψ has $OPT < sm$. Thus for any assignment P provides, V picks a a satisfied clause with probability less than s; that is, V accepts with probability less than s. The soundness can be brought down to $1/2$ by repeating the check $O(1)$ many times independently in parallel.

We now show that the first statement of the theorem implies the second. To this end, assume the PCP theorem. We will now give a deterministic polynomial time reduction from 3-COLOR to $\mathrm{GAP\text{-}E3SAT}_{1,s}$. We will think of the bits of the proof as variables $x_1, x_2, \cdots, x_{poly(n)}$ for anE3SAT formula. Given G, R will first run the verifier's polynomial time pre-computation steps. Then R enumerates all the $2^{O(\log n)} = \mathrm{poly}(n) = N$ random choices of V — each choice gives some C proof locations $(x_{i_1}, x_{i_2}, \cdots, x_{i_C})$ and a predicate ϕ on the C bits. R further canonically converts $\phi(x_{i_1}, x_{i_2}, \cdots, x_{i_C})$ to an equivalent E3CNF formula (in this step R may need to add some auxiliary variables, $y_1, y_2, \cdots, y_{C'}$). Without loss of generality we may assume that each equivalent E3CNF has exactly K clauses where $K = C \cdot 2^C$. Finally, R outputs the conjunction of all these $m = N \cdot K$ clauses. We now argue that this reduction works.

- Completeness: If $G \in$ 3-COLOR then we know there is a proof that satisfies all of the verifier's checks. Thus all of the E3CNF formulas the reduction outputs can be simultaneously satisfied; i.e., $OPT = m$ as needed.

- Soundness: If $G \notin$ 3-COLOR, then for every proof (assignment to the x_i's) and assignment to the auxiliary variables, at least half of the verifier's N checks must fail. Whenever a check fails, the corresponding E3CNF has at most $K - 1 = K(1 - 1/K)$ many satisfied clauses. Thus overall, the number of simultaneously satisfiable clauses is at most

$$\frac{N}{2}K(1-1/K) + \frac{N}{2}K = NK\left(1 - \frac{1}{2K}\right) = m\left(1 - \frac{1}{2K}\right).$$

Thus, $OPT \leq sm$, where $s = (1 - \frac{1}{2K})$, and this is an absolute constant less than 1 as needed.

□

2 The proof of the PCP theorem and expanders

Armed with Theorem 1.4, we will prove the PCP theorem by showing that $\mathrm{GAP\text{-}E3SAT}_{1,s}$ is NP-hard for some $s < 1$. Dinur's paper in fact proves this version of the PCP theorem. Her proof uses objects called expanders — in this and the next lecture, we will spend some time developing facts about expanders.

To give a rough idea of where expanders fit in the scheme of things, here is a brief overview of Dinur's proof. Note that it is easy to see from the proof of Theorem 1.4 that the PCP theorem is also implied by showing that $\mathrm{GAP3\text{-}COLOR}_{1,s}$ is NP-hard, where in this gap version, the quantity we are interested in is the number of edges in a 3-coloring that are colored properly. The way Dinur's proof work is to start with the fact that $3\text{-}\mathrm{COLOR}$ is NP-hard, from which one immediately deduces that $\mathrm{GAP3\text{-}COLOR}_{1,1-\frac{1}{m}}$ is NP-hard, where m is the number of edges. (This is because in any illegal 3-coloring, at least one edge must be violated.) The proof will try to amplify the "soundness gap" from $\frac{1}{m}$ up to some universal constant. At each stage the proof will be working with a *constraint graph* G (initially, the constraints in the input to $3\text{-}\mathrm{COLOR}$ is that the endpoints of each edge have differnt colors from $\{1, 2, 3\}$). In the most important step of the proof, a new graph G^t is constructed from G, where the constraints in G^t correspond to walks in G of length t. If the constraint graphs are nicely structured (i.e., are constant-degree *expanders*) then these walks in G will mix nicely.

3 Expanders

Roughly speaking, expanders are graphs that have no "bottlenecks". In other words, they are graphs with high connectivity. More formally, we will be interested in the following quantity:

Definition 3.1. *The* edge expansion *of a graph $G = (V, E)$, denoted by $\phi(G)$, is defined as*

$$\phi(G) = \min_{S \subseteq V, |S| \leq \frac{|V|}{2}} \frac{|E(S, \overline{S})|}{|S|},$$

where $\overline{S} = V \setminus S$ and $E(S, \overline{S}) = \{(u, v) \in E \mid u \in S \text{ and } v \in \overline{S}\}$.

We say G is an *expander* if $\phi(G)$ is "large" (at least some positive constant). Note however that it is not hard to find such a graph; for example, the complete graph has $\phi(G) \geq \Omega(n)$. The challenge is to find *sparse* expanders, especially d-regular expanders for some constant d. In fact such sparse expanders exist and can be constructed explicitly.

Theorem 3.2. *There exist constants $d > 1$ and $\phi_0 > 0$ and an explicit family of d-regular graphs $\{G_n\}_{n \geq 1}$ such that $\phi(G_n) \geq \phi_0$.*

3.1 Alternate definition of expanders

We now consider an alternate way of looking at expanders. For any d-regular graph G, let A^G denote the *adjacency matrix*, that is, A^G_{ij} is 1 if $(i,j) \in E(G)$ and 0 otherwise (one could also work with multi-graphs in which case for an edge (i,j), A^G_{ij} would be the multiplicity of that edge).

For the all-ones vector, $\vec{v} = \vec{1}$, $A^G\vec{v} = d \cdot \vec{v}$; that is, $\vec{v}$ is an *eigenvector* with *eigenvalue* d. If A is a real and symmetric $n \times n$ matrix (as A^G is) then A has n real-valued eigenvalues $\lambda_1 \geq \lambda_2 \cdots \geq \lambda_n$. For A^G, $\lambda_1 = d$ is easily seen to be the largest eigenvalue.

Definition 3.3. *A d-regular graph G is an* (n,d,λ)-expander *if* $\lambda = \max\{|\lambda_i(G)| : i \neq 1\} = \max\{\lambda_2(G), |\lambda_n(G)|\}$ *and* $\lambda < d$.

This definition of an expander is closely related to the definition we saw before.

Theorem 3.4. *If G is a (n,d,λ)-expander then*

$$\frac{\phi(G)^2}{2d} \leq d - \lambda \leq 2\phi(G).$$

In other words, large expansion is equivalent to large *spectral gap* (that is, $d - \lambda$). We will see the proof of the upper bound (which is the direction we actually need) next lecture. Explicit constructions of expanders tend to work with this spectral definition:

Theorem 3.5. *There exist explicit constants $d \geq 3$ and $\lambda < d$ and an explicit (polynomial-time computable) family of (n,d,λ)-expanders.*

The second-largest eigenvalue λ of a real symmetric $n \times n$ is characterized as follows (via a "Rayleigh quotient"):

$$\lambda = \max_{x \in \mathbb{R}^n, x \cdot \vec{1} = 0, x \neq 0} \frac{|\langle Ax, x\rangle|}{\langle x, x\rangle}. \tag{1}$$

Let us show this. As A is a real symmetric matrix, there exists an orthonormal basis $\vec{v}_1, \vec{v}_2, \cdots, \vec{v}_n$ where each $\vec{v}_i$ is an eigenvector of A. Letting $x = \vec{v}_2$ yields a ratio in (1) of $|\lambda_2|$; similarly, we can let $x = \vec{v}_n$ and get a ratio of $|\lambda_n|$. Thus, we certainly have $\leq$ in (1). For the other direction, write any x as $\sum_{i=1}^n a_i \vec{v}_i$. Using $\langle x, \vec{v}_1\rangle = 0$, we conclude $a_1 = 0$. Now $Ax = \sum_{=1}^n a_i \lambda_i \vec{v}_i$ and thus,

$$|\langle Ax, x\rangle| = \left|\sum_{i=2}^n a_i^2 \lambda_i\right| \leq \sum_{i=2}^n |\lambda_i| a_i^2 \leq \lambda \sum_{i=2}^n a_i^2 = \lambda \langle x, x\rangle$$

as required.

Finally, we conclude this lecture by proving a simple lemma that will be used in the proof of the PCP theorem.

Lemma 3.6. *If G is a d-regular graph on the vertex set V and H is a d'-regular graph on V then $G' = G \cup H = (V, E(G) \cup E(H))$[1] is a $d + d'$-regular graph such that*

$$\lambda(G') \leq \lambda(G) + \lambda(H)$$

[1] Here the union of edges results in a mutliset.

Proof. Choose x such that $\|x\| = 1$, $x \cdot \vec{1} = 0$ and $\lambda(G') = \langle A^{G'}x, x\rangle$. Now

$$\begin{aligned} \langle A^{G'}x, x\rangle &= \langle A^{G}x, x\rangle + \langle A^{H}x, x\rangle \\ &\leq \lambda(G) + \lambda(H). \end{aligned}$$

The equality follows from the definition of G' and the inequality follows from (1). □

CSE 533: The PCP Theorem and Hardness of Approximation **(Autumn 2005)**

Lecture 3: Expander Graphs and PCP Theorem Proof Overview

Oct. 5, 2005

Lecturer: Venkatesan Guruswami *Scribe: Matt Cary*

1 Key Expander Graph Lemmas

Recall in last lecture that we defined a (n, d, λ)-expander to be a d-regular n-vertex undirected graph with second eigenvalue λ. We also defined the edge expansion of a graph G with vertex set V to be

$$\phi(G) = \min_{\substack{S \subset V \\ |S| \leq n/2}} \frac{|E(S, \overline{S})|}{|S|},$$

where $E(S, \overline{S})$ is the set of edges between a vertex set S and its complement.

The following lemma shows that the eigenvalue formulation of an expander is essentially equivalent to edge expansion.

Lemma 1.1. *Let G be a (n, d, λ) expander. Then*

$$\phi(G) \geq (d - \lambda)/2.$$

Remark 1.2. *In the other, harder, direction, it is possible to show that $\phi(G) \leq \sqrt{2d(d-\lambda)}$. For our purposes of constructing and using expanders, the easier direction shown in this lemma is enough.*

Proof. Let V and E be the vertex and edge sets of G. Let $S \subset V$ with $|S| \leq n/2$. We will set up a vector x that is similar to the characteristic vector of S, but perpendicular to $\overrightarrow{1}$. Then by the Rayleigh coefficient formulation of λ, we have that $\|Ax\| \leq \lambda\|x\|$, where $A = A(G)$ is the adjacency matrix of G.

Accordingly, we define x by

$$x_v = \begin{cases} -|\overline{S}| & \text{if } v \in S \\ |S| & \text{if } v \in \overline{S} \end{cases},$$

and you can confirm that $\sum x_v = 0$ so that $x \perp \overrightarrow{1}$. Now combining the Raleigh coefficient with the fact that $\langle Ax, x\rangle \leq \|Ax\| \cdot \|x\|$ we get

$$\langle Ax, x\rangle \leq \lambda\|x\|^2.$$

Note that $(Ax)_u = \sum_{(u,v)\in E} x_v$, so that as A is symmetric

$$\begin{aligned}\langle Ax, x\rangle &= \sum_u x_u \sum_{(u,v)\in E} x_v \\ &= 2\sum_{(u,v)\in E} x_u x_v \\ &= 2|E(S,\overline{S})|\cdot\big(-|S|\cdot|\overline{S}|\big) + \big(d|S| - E(S,\overline{S})\big)|\overline{S}|^2 + \big(d|\overline{S}| - E(S,\overline{S})\big)|S|^2\end{aligned}$$

where in the last two terms we count the number of edges wholly in S and $\overline{S}$, respectively: there are $d|S|$ edge originating in S, minus those that cross over to $\overline{S}$, cut in half as we have double counted. We then simplify by noting that $|S| + |\overline{S}| = n$ and $(|S|^2 + 2|S||\overline{S}| + |\overline{S}|^2 = (|S| + |\overline{S}|)^2 = n^2$ to get

$$= d|S||\overline{S}|n - |E(S,\overline{S})|n^2.$$

Hence as $\langle Ax, x\rangle \le \lambda\|x\|^2$ and $\|x\|^2 = |S||\overline{S}|^2 + |\overline{S}||S|^2 = |S||\overline{S}|n$, we can say

$$d|S||\overline{S}|n - |E(S,\overline{S})|n^2 \le \lambda|S||\overline{S}|n$$

implying

$$|E(S,\overline{S})| \ge \frac{d-\lambda}{n}|S||\overline{S}|$$

which shows

$$\frac{|E(S,\overline{S})|}{|S|} \ge \frac{d-\lambda}{2}$$

as $|\overline{S}|/n \ge 1/2$ by our assumption on S. □

Suppose we take a walk of length t from a random vertex v in G, as shown in Figure 1, and are interested in the probability that the entire walk takes place within the set B of size βn. If each vertex in the walk were picked independently, the probability would be β^t. The next lemma shows that even when the vertices are taken from a walk, in a good expander we are not that far off from independent vertex choices. This illustrates why expanders are so useful: structures that should be very dependent, such as walks, look very close to independent. We will not give the proof of this lemma. We will prove a different (easier) result concerning random walks in expanders in Lemma 3.1, and later use it when proving the PCP theorem.

Lemma 1.3. *Let G be a (n, d, λ)-expander, and $B \subset V(G)$ a set of size βn. Then the probability that a t step walk starting from a vertex v never leaves B is at most*

$$\left(\sqrt{\beta^2 + \left(\frac{\lambda}{d}\right)^2(1-\beta^2)}\right)^t,$$

where the probability is taken over the uniform choice of v as well as the steps in the walk.

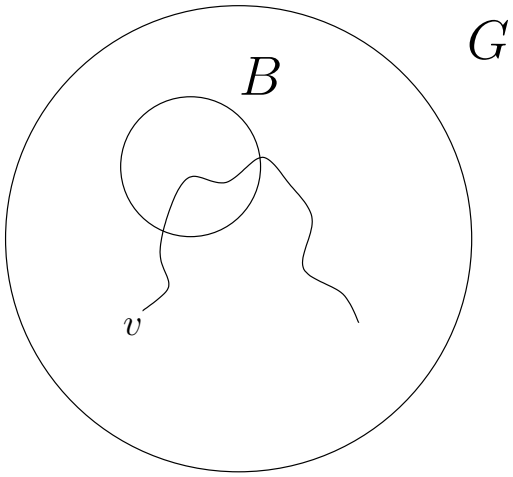

Figure 1: Hitting a set B during a walk in a graph G. Here the walk only intersects B; Lemma 1.3 bounds the probability that the entire walk starts and remains inside B.

Remark 1.4. *Note that as $\lambda \to 0$, the probability approaches β^t as our intuition about independent vertex choices suggests. Also, if $\beta = 1 - \epsilon$ for $\epsilon \to 0$, then the probability is at most $(1 - (1 - \lambda^2/d^2)(1 - \beta^2))^{t/2} \leq 1 - O(t\epsilon)$, or in other words, a t-step random walk has probability $\Omega(t\epsilon)$ of hitting a set of density ϵ, for small ϵ. This fact will later be appealed to sketch the basic intuition behind Dinur's proof.*

2 Constructions of Explicit Expanders

In case you are starting to wonder if such marvelous objects as expanders can exist at all, let alone with interesting parameters, we survey couple of constructions that show that some constant-degree regular graphs exist with good expansion and they can be described very explicitly.

2.1 The Margulis/Gaber-Galil Expander

We construct an n^2 vertex graph G whose vertex set is $\mathbb{Z}_n \times \mathbb{Z}_n$, where $\mathbb{Z}_n$ is the ring of integers modulo n. Given a vertex $v = (x, y)$, we connect it to the following vertices:

$$\begin{array}{cc} (x + 2y, y) & (x, 2x + y) \\ (x + 2y + 1) & (x, 2x + y + 1) \end{array},$$

where all operations are done modulo n. We also add the edges corresponding to the inverse transformations. This is then an 8-regular undirected graph, where there may be self loops or multiedges, depending on n. One can prove that for this graph $\lambda \leq 5\sqrt{2} < 8$.

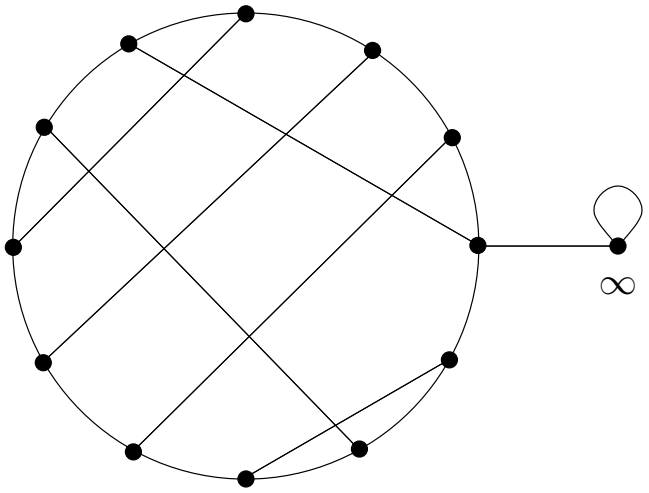

Figure 2: The LPS Expander

2.2 The Lubotzky-Phillips-Sarnak Expander

The construction presented here is much simplified from the original construction. Let $V = \mathbb{Z}_p \cup \{\infty\}$, where p is prime. We view V as a p-element field defined by addition and multiplication modulo p, where we extend the multiplicative inverse by defining 0^{-1} to be the special point ∞, and $\infty + x = \infty$ for all $x \in V$. Given any vertex x, connect it to $x+1$, $x-1$ and x^{-1}. That's it! This gives a 3-regular graph with second largest eigenvalue $\lambda \leq \lambda_0 < 3$ for some absolute constant λ_0. The structure of the graph as shown in Figure 2. The graph is a cycle with a matching between the edges. Note that the extra point with a self-loop that is introduced by the point ∞ can be removed along with 0, and 1 connected to $p-1$ without much disturbing the expansion. Actually since 1 and $p-1$ also have self loops for their inverses (instead of matching edges, though this isn't reflected in the figure!), we can remove them, and simply have a simple graph that is a cycle on $p-3$ nodes with a matching.

3 A Final Expander Lemma

In this section we prove a version of Lemma 1.3 that will be used in the proof of the PCP theorem. Here the set of interest will be an *edge* set $F \subset E$, we consider a walk that starts along a particular edge in F, and consider the chance that the edge used in the t^{th} step is also in F.

Lemma 3.1. *Let G be an (n, d, λ)-expander and $F \subset E(G) = E$. Then the probability that a random walk, starting in the zero-th step from a random edge in F, passes through F on its t^{th} step is bounded by*

$$\frac{|F|}{|E|} + \left(\frac{\lambda}{d}\right)^{t-1}.$$

Proof. To prove this lemma we will use a very useful technique essentially that used for random walks on Markov chains. Let x be the distribution on the vertices of G for the start of the walk. That is, x_v is the probability that our walk begins at vertex v. Consider the first step in the random walk. The probability that this ends at a vertex u is the sum, over all edges (v, u), of the probability that we were on v before this step, times the probability we chose the edge (v, u) out of all the other edges leaving v. As G is d-regular, v has exactly d edges leaving it, the chance we take the one heading to u is just $1/d$ (we will ignore the possibilities of multi-edges—as you will see, the final expression we actually use takes this into account). Hence if x' is the distribution on the vertices of G after the first step,

$$x'_u = \sum_{(v,u)\in E} x_v/d.$$

Now let A be the adjacency matrix of G. Then the row A_u has ones in exactly the columns (v, u) where there is an edge (v, u) in G. Hence we can write the above expression compactly as $x' = Ax/d$. Note that in this case, multi-edges are handled correctly, for if there is a multi-edge of multiplicity m between v and u, the corresponding entry in A will be m, giving the probability we take that edge from v as m/d as desired. This notation is so convenient we will normalize by defining $\tilde{A} = A/d$ so that simply

$$x' = \tilde{A}x.$$

If we take i steps, the distribution we reach is given by $\tilde{A}^i x$. Let P be the probability we're interested in, which is that of traversing an edge of F in the t^{th} step. Suppose w is the vertex we arrive at after the $(t-1)^{\text{th}}$ step. Let y_w be the number of edges of F incident on w, divided by d. Then $P = \sum_{w\in V} (\tilde{A}^{i-1}x)_w y_w$, where x is the initial distribution.

To calculate x, we pick an edge in F, then pick one of the endpoints of that edge to start on. If v has k edges of F incident on it, we have a $k/|F|$ chance to pick that edge, then a further $1/2$ chance to pick v. Now, y_w is the same quantity k, but divided by d instead of $2|F|$. Hence, we can write that $y_w = x_w \cdot 2|F|/d$. Without calculating x further, we now write

$$\begin{aligned} P &= \sum_{w\in V} (\tilde{A}^{i-1}x)_w y_w \\ &= \sum (\tilde{A}^{i-1}x)_w x_w \cdot \frac{2|F|}{d} \\ &= \frac{2|F|}{d} \langle \tilde{A}^{i-1}x, x\rangle. \end{aligned}$$

To finish our calculation, we rely on an as-yet unused property of G: its regularity. As each vertex in G has exactly d neighbors, each row in $\tilde{A}$ sums to one. Hence if $x^{\|}$ is the uniform distribution on G—$x^{\|}_v = 1/n$—then $\tilde{A}x^{\|} = x^{\|}$. As x is a probability distribution, we can decompose it as $x = x^{\|} + x^{\perp}$ with $\langle x^{\|}, x^{\perp}\rangle = 0$. Then by linearity and the fact just mentioned,

$$\begin{aligned} \tilde{A}^{i-1}x &= \tilde{A}^{i-1}x^{\|} + \tilde{A}^{i-1}x^{\perp} \\ &= x^{\|} + \tilde{A}^{i-1}x^{\perp}. \end{aligned}$$

Hence,

$$\begin{aligned}
\langle \tilde{A}^{i-1}x, x\rangle &= \langle \tilde{A}^{i-1}x^{\|}, x\rangle + \langle \tilde{A}^{i-1}x^{\perp}, x\rangle \\
&= \langle x^{\|}, x\rangle + \langle \tilde{A}^{i-1}x^{\perp}, x\rangle \\
&= \|x^{\|}\|^2 + \langle \tilde{A}^{i-1}x^{\perp}, x\rangle \\
&= \frac{1}{n} + \langle \tilde{A}^{i-1}x^{\perp}, x\rangle \\
&\leq \frac{1}{n} + \|\tilde{A}^{i-1}x^{\perp}\| \cdot \|x\| \\
&\leq \frac{1}{n} + \left(\frac{\lambda}{d}\right)^{i-1} \|x^{\perp}\| \cdot \|x\| \\
&\leq \frac{1}{n} + \left(\frac{\lambda}{d}\right)^{i-1} \|x\|^2,
\end{aligned}$$

as $\|x^{\perp}\| \leq \|x\|$. Now notice as the entries of x are positive that $\|x\|^2 = \sum x_v^2 \leq \max x_v \sum x_v = \max x_v$, as $\sum x_v = 1$, x being a probability distribution. The maximum x_v is achieved when all edges incident to v are in F, and in that case $x_v = d/(2|F|)$, so the calculation above continues

$$\leq \frac{1}{n} + \left(\frac{\lambda}{d}\right)^{i-1} \frac{d}{2|F|}.$$

Hence

$$P \leq \frac{2|F|}{dn} + \left(\frac{\lambda}{d}\right)^{i-1}$$

which finishes the proof as $|E| = nd/2$.

□

4 Overview of the GAP-3SAT Hardness Proof

In this section we give an overview of the proof of hardness for GAP-3SAT that will occupy us over the next several lectures. We will actually prove the hardness of a problem that can be seen as a generalization of graph optimization problems, which has an easy reduction to GAP-3SAT.

Definition 4.1. *A* constraint graph *is given by an alphabet Σ, a graph $G = (V, E)$ and a set of constraints $C = \{c_e \subseteq \Sigma \times \Sigma \mid e \in E\}$. A* labeling *on G is an assignment $\sigma : V \to \Sigma$ of elements from Σ to each vertex of G. A labeling σ* satisfies *an edge $(u, v) \in E$ if $\big(\sigma(u), \sigma(v)\big) \in c_e$ (for each edge a canonical orientation $u \to v$ is assumed).*

The optimization problem *for a constraint graph is to find a labeling that maximizes the number of satisfied edges. The* gap problem *for constraint graphs with* gap parameter ϵ, $0 < \epsilon \leq 1$, *is the following: given a constraint graph, such that either (i) there is a labeling that satisfies all the edges, or (ii) every labeling fails to satisfy at least a fraction ϵ of edges, determine which of the cases (i) or (ii) holds.*

Remark 4.2. *Many graph problems can be expressed as a constraint graph problem. For example, given a graph G to k-color, let Σ be a k-element alphabet, and define the set of constraints C as $\{(a, b)\}_{a \neq b}$. The optimization problem is to find a coloring for G that maximizes the number of valid edges, those whose endpoints are different colors.*

Note that the hardness of the decision problem for constraint graphs implies the hardness of the gap problem with gap parameter $1/|E|$. We will amplify this hardness in a series of stages, where at each stage we take a constraint graph G_i over alphabet Σ, and produce G_{i+1} also over alphabet Σ, where the number of unsatisfied edges in G_{i+1} is at least twice the number in G_i. If the size of G_{i+1} increases polynomially over the size of G_i, this will not be enough, as we will apply this $\log |E|$ times. Hence we must also insure that the size of G_{i+1} is at most a constant factor larger than that of G_i, so that the size of $G_{\log |E|}$ is a polynomial in the size of G_1.

Each stage will be split into 4 steps. Let $\mathrm{gap}(G)$ denote the minimum fraction of unsatisfied edges over all labelings of G.

Sparsification (degree-reduce): $G_i \rightarrow G^{(1)}$, a d-regular graph where $\mathrm{gap}(G_i^{(1)}) \geq \beta_1 \mathrm{gap}(G_i)$, with integer d, $\beta_1 > 0$ being absolute constants. This step is achieved by placing a $(d-1)$-regular expander at each vertex of G_i with number of vertices of the j'th expander being equal to the degree of vertex j.

Expanderize: $G^{(1)} \rightarrow G^{(2)}$, that is a good expander. We can achieve this by unioning $G^{(1)}$ with an expander, and apply Lemma 3.6 of the previous lecture. This step will also reduce the gap by an absolute constant factor.

Amplify the Gap: $G^{(2)} \rightarrow G^{(3)}$ by powering, which will increase $\mathrm{gap}(G^{(2)})$ a lot, more than making up for the loss of the other steps. This will also increase Σ, where we want the final graph to be over the same alphabet as the original graph. This step was the main innovation of Dinur.

Composition (alphabet-reduce): $G^{(3)} \rightarrow G_{i+1}$ by reducing the alphabet back to the original Σ.

In addition, each step will only increase the size of the graph by a constant factor.

It is worth pointing out that expanders are used in each of the first three steps above!

CSE 533: The PCP Theorem and Hardness of Approximation **(Autumn 2005)**

Lecture 4: PCP Theorem proof: Degree reduction, Expanderization

Oct. 10, 2005

Lecturer: Ryan O'Donnell *Scribe: Vibhor Rastogi and Ryan O'Donnell*

1 Constraint graphs and the PCP Theorem

Definition 1.1. *A* constraint graph $(G, \mathcal{C})$ *over alphabet* Σ *is defined to be an undirected graph* $G = (V, E)$ *together with a set of "constraints"* $\mathcal{C}$*, one per edge. The constraint specified for edge* $e \in E$*, call it* $c_e \in \mathcal{C}$*, is just a subset of* $\Sigma \times \Sigma$*.*

Note that the graph is allowed to have multi-edges (representing multiple constraints over the same pair of vertices) and self loops.

Definition 1.2. *For a fixed* Σ*, the algorithmic problem* $\mathrm{MAX\text{-}CG}(\Sigma)$ *is defined as follows: Given is a constraint graph* $(G, \mathcal{C})$*; the goal is to find an "assignment"* $\sigma : V \to \Sigma$ *such that number of "satisfied" edges in* E *— i.e., edges* $e = (u, v)$ *such that* $(\sigma(u), \sigma(v)) \in c_e$ *— is maximized.*

Note that the input size of an instance of $\mathrm{MAX\text{-}CG}(\Sigma)$ is $\Theta(m \log n)$, where m is the number of edges in G, n is the number of vertices in G, and *the constant hidden in the* $\Theta(\cdot)$ *depends on* $|\Sigma|$. This is because the input includes an explicitly written constraint — i.e., a subset of $\Sigma \times \Sigma$ — on each edge.

To study the approximability of $\mathrm{MAX\text{-}CG}(\Sigma)$, we define the gapped version of it:

Definition 1.3. $\mathrm{GAP\text{-}CG}(\Sigma)_{c,s}$ *(for* $0 < s < c \leq 1$*) is the following algorithmic decision problem: Given a constraint graph* $(G, \mathcal{C})$*,*

- *output YES if* $\exists\ \sigma$ *such that fraction of edge-constraints* σ *satisfies is* $\geq c$*;*
- *output NO if* $\forall\ \sigma$ *the fraction of edge-constraints* σ *satisfies is* $< s$*;*
- *otherwise output anything.*

Our goal for the next few lectures will be to prove the following theorem. This theorem is easily seen to imply the PCP Theorem, via the connections we saw in Lecture 2.

Theorem 1.4. *(Implies the PCP Theorem.) There is a fixed alphabet* Σ_0 *(the alphabet* $\Sigma_0 = \{1, 2, \ldots, 64\}$ *suffices) and a fixed constant* $s_0 < 1$ *(*$s_0 = 1 - 10^{-6}$ *probably suffices) such that* $\mathrm{GAP\text{-}CG}(\Sigma_0)_{1,s}$ *is NP-hard.*

2 Outline of the proof that $\mathrm{GAP\text{-}CG}(\Sigma_0)_{1,s_0}$ is NP-hard

To prove that $\mathrm{GAP\text{-}CG}(\Sigma_0)_{1,s_0}$ is NP-hard we must reduce some NP-complete problem to it. We will reduce from the $3\text{-}\mathrm{COLOR}$ problem. The $3\text{-}\mathrm{COLOR}$ can be viewed as a constraint graph problem: here the alphabet is $\Sigma = \{1,2,3\}$ and the constraints are all the same, inequality constraints. Note that it doesn't hurt to let the alphabet be $\Sigma_0 \supset \{1,2,3\}$; the constraints can disallow any vertex-label that is not in $\{1,2,3\}$. Also note that in the YES case of $3\text{-}\mathrm{COLOR}$, all of the constraints can be simultaneously satisfied, whereas in the NO case of $3\text{-}\mathrm{COLOR}$, every coloring must violate at least one edge; hence we can say that in the NO case, at most a $1 - 1/m$ fraction of constraints can be simultaneously satisfied. (Here m is the number of edges.) Thus the fact that $3\text{-}\mathrm{COLOR}$ is NP-hard can be stated as:

Theorem 2.1. $\mathrm{GAP\text{-}CG}(\Sigma_0)_{1,1-1/m}$ *is NP-hard.*

Our goal is to make a polynomial-time reduction from this problem that brings the factor in the NO case down from $1 - 1/m$ to $s_0 < 1$.

To understand how this reduction works, we will keep track of a number of parameters of constraint graphs:

Definition 2.2 (Parameters of Constraint Graphs). *Given a constraint graph $G = ((V,E),\mathcal{C})$ over alphabet Σ, we define the following parameters:*

- $\mathrm{size}(G)$ *is the total* size *in bits of expressing the constraint graph. This is something like $\Theta(m \log m \mathrm{poly}(|\Sigma|)$. Since $|\Sigma|$ will always be a "constant", keeping track of the size mostly means keeping track of the number of edges, m.*
- $|\Sigma|$ *is the* alphabet size.
- $\deg(G)$ *is the maximum vertex degree in G. We will also keep track of whether G is a regular graph.*
- $\lambda(G)$ *is the size of the second-largest eigenvalue of the graph G. We will only define this parameter when the graph G is d-regular.*
- *Most importantly,* $\mathrm{gap}(G)$*, the* satisfiability gap *of the constraint graph system, is defined to be the* fraction of constraints that must be violated in any assignment to G*. In other words,*

$$\begin{aligned} \mathrm{gap}(G) &= 1 - \max_{\sigma: V \to \Sigma} \{\textit{fraction of constraints } \sigma \textit{ satisfies}\} \\ &= \textit{fraction of constraints violated by the "best" assignment } \sigma. \end{aligned}$$

  $\mathrm{gap}(G) = 0$ *means that the constraint graph has a completely valid assignment.*

To prove Theorem 1.4, our plan is to reduce $\mathrm{GAP\text{-}CG}(\Sigma_0)_{1,1-1/m}$ (i.e., $3\text{-}\mathrm{COLOR}$) to $\mathrm{GAP\text{-}CG}(\Sigma_0)_{1,s_0}$. In other words, we need a polynomial-time algorithm that takes a constraint graph G over alphabet

Σ_0 and gap equal to either 0 (the YES case) or $\geq 1/m$ (the NO case) and outputs a new constraint graph G' with parameters

$$\mathrm{size}' = \mathrm{poly}(\mathrm{size}), \qquad \Sigma' = \Sigma_0, \qquad \mathrm{gap}' = \begin{cases} 0 & \text{if } \mathrm{gap} = 0, \\ \geq 10^{-6} & \text{if } \mathrm{gap} \geq 1/m \end{cases}$$

where here the unprimed parameters denote the parameters of the input constraint graph G and the primed parameters denote the parameters of the output constraint graph G'. Here also 10^{-6} is equated with $1 - s_0$; i.e., it represents some small positive universal constant. We will accomplish this reduction by running many small subroutines. Each subroutine is designed to "improve" one parameter of the current constraint graph at hand. However, each will have "side effects" that hurt the other parameters. Nevertheless, combining them in the right order will give a reduction that slowly increases the gap, while keeping the other parameters reasonable.

Specifically, we will design emphfour polynomial-time subroutines which take as input a constraint graph G and output a constraint graph G':

Degree-reduction.
Assumption: the alphabet is Σ_0, constant sized.
Main effect: G' becomes a regular graph, with $\deg' = d_0$, some universal constant.
Side effects:

- $\mathrm{size}' = O(\mathrm{size})$ (i.e., the size goes up by a constant factor).
- The alphabet does not change.
- If $\mathrm{gap} = 0$ then $\mathrm{gap}' = 0$. I.e., satisfiable c.g.'s get transformed to satisfiable c.g.'s.
- Otherwise, $\mathrm{gap}' \geq \mathrm{gap}/O(1)$. I.e., the gap goes down by at most a universal constant.

Expanderization.
Assumption: G is regular and $\deg$ is a constant.
Main effect: G' becomes a *constant degree expander*. I.e., G' is d_1-regular for some universal constant d_1, and $\lambda' < d_1$.
Side effects: Same side effects as in the Degree-reduction routine.

Gap amplification. *This is the critical step and the main novelty in Dinur's proof.*
Assumption: G is an (n, d, λ)-expander, with $\lambda < d$ universal constants; and, the alphabet is Σ_0, a constant.
Extra parameter: This subroutine is parameterized by a fixed constant we call t. We will later explicitly say how to choose this constant.
Main effect: The main effect is that in the case where $\mathrm{gap} > 0$, the gap increases by a factor of roughly t. Specifically:

$$\mathrm{gap}' \begin{cases} = 0 & \text{if } \mathrm{gap} = 0, \\ \geq \frac{t}{O(1)} \cdot \min(\mathrm{gap}, 1/t) & \text{else.} \end{cases}$$

In other words, if G was satisfiable then so is G'; however, if G was not satisfiable, then G' has gap which is larger by a factor of t — unless this is already bigger than the universal constant $1/t$, in which case it just becomes at least $1/t$.
Side effects:

- $\text{size}' = O(\text{size})$.
- The new alphabet Σ is a much huger constant; something like $\Sigma_0^{d^t}$.
- $\deg'$, λ' may become bad; we don't care what happens to them.

Composition (alphabet-reduction).
Main effect: The new alphabet $\Sigma' = \Sigma_0$. Side effects:

- $\text{size}' = O(\text{size})$, where the constant depends on the input alphabet size $|\Sigma|$. (The dependence is very bad, in fact; at least exponential.)
- If $\text{gap} = 0$ then $\text{gap}' = 0$.
- Otherwise, $\text{gap}' \geq \text{gap}/O(1)$.

We claim that if we can show how to do all four of these steps, then this can be used to prove Theorem 1.4 and hence the PCP Theorem:

Proof. (of Theorem 1.4) We begin from the NP-hard 3-COLOR problem, $\text{GAP-CG}(\Sigma_0)_{1,1-1/m}$ (see Theorem 2.1). Here we have a constraint graph G over alphabet Σ_0 which has $\text{gap} = 0$ in the YES case and $\text{gap} \geq 1/m$ in the NO case. Suppose we run all four subroutines in order on G, producing some new constraint graph G'. What results is that $\text{size}' = O(\text{size})$, the alphabet ends up still being Σ_0, and the new gap' is either still 0 in the YES case, or it increases by a factor of $t/O(1)$ in the NO case (assuming this is still less than $1/t$). Let us select the constant t to be large enough so that this $t/O(1)$ is at least 2. (By inspecting the proofs of the four subroutines, one can convince oneself that $t = 10^6$ is more than sufficient.)

Treating these four subroutines as one black box now, we have a polynomial-time algorithm that doubles the satisfiability gap (in the NO cases, assuming it's below 10^{-6}), keeps the alphabet equal to Σ_0, and only blows up by the size by a constant factor. Our overall reduction is to simply run this black box $\log m$ times. Note that the overall size blowup is $O(1)^{\log m} = \text{poly}(m)$; i.e., polynomial. Thus the overall running time is also polynomial. Further, the gap either always stays 0 (in the YES case) or it goes up from $1/m$ to at least the constant 10^{-6} in the NO case. This completes the proof. □

Having outlined the proof, the rest of this lecture and the next four lectures are devoted to showing how to do the four subroutines.

3 Degree-reduction

In this section we show how to do the degree-reduction step. The way to do this step is "well-known" in the literature; it is sometimes called the "Expander Replacement Lemma" of Papadimitriou and Yannakakis.

In this step we will use the fact from Lectures 2 and 3 that there exist universal constants $\lambda_0 < d_0$ such that (n, d_0, λ_0)-expanders can be explicitly constructed in time $\mathrm{poly}(n)$.

Given an input constraint graph $(G, \mathcal{C})$, the following transformation gives a new constraint graph $(G', \mathcal{C}')$ achieving our goals:

- Replace each vertex $u \in V$ by $deg(u)$ many vertices to get the new vertex set V'. Denote the set of new vertices corresponding to u by $\mathrm{cloud}(u)$. Each vertex in $\mathrm{cloud}(u)$ naturally corresponds with a neighbor of u from G.
- For each edge $(u, v) \in E$, place an "inter-cloud" edge in E' between the associated cloud vertices. This gives exactly one inter-cloud edge per vertex in V'. Whatever the old constraint on (u, v) was, put the exact same constraint on this inter-cloud edge.
- For each $u \in V$, put a $(\deg(u), d_0, \lambda_0)$-expander on $\mathrm{cloud}(u)$. Further, put *equality* constraints on these expander edges.

We can observe that in this process each new vertex in V' has degree exactly equal to $d_0 + 1$. Thus we have created a $(d_0 + 1)$-regular graph, as desired. Also number of newly added edges is equal to $\sum_{u \epsilon V} \frac{deg(u) d_0}{2} = d_0 \sum_{u \epsilon V} \frac{deg(u)}{2} = d_0 |E|$. Hence $|E'| = (d_0 + 1)|E|$; i.e., the number of edges only went up by a constant factor. This implies that the overall size went up by only a constant factor.

Thus it remains to show the properties of the new satisfiability gap. It is easy to see that if the old gap was 0 then so is the new gap — given a satisfying assignment for the old constraint graph one can just give each vertex in $\mathrm{cloud}(u)$ the assignment to u, and this produces a satisfying assignment in the new graph. Hence we only need to show that in the NO case, $\mathrm{gap}' \geq \mathrm{gap}/O(1)$.

Lemma 3.1. *$gap' \geq \mathrm{gap}/O(1)$.*

Proof. Let $\sigma' : V' \to \Sigma_0$ be a best assignment for $(G', \mathcal{C}')$. To relate the fraction of edges in E' that σ' satisfies back to the gap in the old constraint graph, we define an "extracted" assignment $\sigma : V \to \Sigma$ as follows: $\sigma(u)$ is defined to be the "plurality vote" of σ' on $\mathrm{cloud}(u)$. By definition, we know that σ violates at least $\gamma|E|$ constraints in $(G, \mathcal{C})$, where we write $\gamma = \mathrm{gap}$ for brevity.

Let us define S^u to be the set of vertices in $\mathrm{cloud}(u)$ on which σ' disagrees with $\sigma(u)$. Suppose $e = (u, v)$ is one of the at least $\gamma|E|$ edges in G that are violated by σ. Let e' be the corresponding inter-cloud edge in E'. The key observation to make is the following: Either σ' violates the edge e' or one of the endpoints of e' belongs to S^u or S^v. Thus we conclude:

$$\gamma|E| \leq (\text{\# edges violated by } \sigma') + \sum_{u \in V} |S^u|.$$

From this key equation we immediately deduce that either a) the number of edges violated by σ' is at least $(\gamma/2)|E|$, or b) $\sum_{u \in V} |S^u| \geq (\gamma/2)|E|$. We now consider these two cases.

In case a), we can quickly finish. Since σ' was a best assignment for $(G', \mathcal{C}')$, we get

$$\mathrm{gap}' = \#\text{ edges violated by } \sigma') \geq \frac{\gamma}{2}|E| = \frac{\gamma}{2(d_0+1)}|E'|,$$

by our earlier calculation $|E'| = (d_0+1)|E|$. Since d_0 is an absolute constant we indeed get $\mathrm{gap}' \geq \mathrm{gap}/O(1)$, as claimed.

To deal with case b), for any $u \in V$ let S^u_a denote the vertices in S^u which σ' labels by $a \in \Sigma_0$. By definition of S^u as the non-plurality set, we must surely have $|S^u_a|/|\mathrm{cloud}(u)| \leq 1/2$. Thus by the fact that the cloud is an expander, we get that there are at least $\Omega(1) \cdot |S^u_a|$ edges within the cloud that come out of S^u_a. (Here the $\Omega(1)$ depends on d_0 and λ_0, but is some explicit positive constant.) Further, *every such edge is violated by* σ', since these edges all have "equality" constraints. Thus overall σ' violates at least the following number of edges:

$$\begin{aligned}
& \sum_{u \in V} \sum_{a} (\Omega(1)/2)|S^u_a| && \text{(each edge counted twice)} \\
= \;& (1/O(1)) \sum_{u \in V} |S^u| \\
\geq \;& (1/O(1))(\gamma/2)|E| && \text{(since we are in case b))} \\
= \;& (1/O(1))(\gamma/2)(|E'|/(d_0+1)) \\
= \;& (\gamma/O(1))|E'|,
\end{aligned}$$

as desired. This completes the proof. □

4 Expanderize

In this section we show how to do the Expanderization subroutine. This subroutine is very easy. Given the constraint graph G with constant degree d, all we need to do is to superimpose an (n, d_0, λ_0)-expander. (This may lead to multiple edges.) On each edge from the expander we simply put a "null" constraint; i.e., a constraint that is always satisfied.

Let us now check what the parameters of G' are. The new graph is regular with degree $d + d_0$, a constant. The new number of edges is $n(d+d_0)/2$; since the old number of edges was $nd/2$, we see that the size of the new constraint graph has only gone up by a constant factor, as desired.

Next, the new constraint graph is indeed a constant degree expander. This is because the new λ' is at most $d + \lambda_0 < d + d_0$, using the Lemma from Lecture 2 about superimposing expanders.

Finally, it remains to check the properties of the gap. In the case that the original gap was 0, the new gap is still 0 — the old satisfying assignment is still a satisfying assignment. In general, suppose we have any assignment σ' for the new constraint graph. Viewing it as an assignment for the old constraint graph, we see that it must violate at least $\mathrm{gap}|E|$ many old constraints. These constraints are still violated in the new graph, and the total number of constraints in the new graph is $O(|E|)$. Hence every assignment in the new graph violates at least $\mathrm{gap}/O(1)$ fraction of the constraints, as needed.

5 "After Stopping Random Walks" and "Before Stopping Random Walks"

For the next lecture's discussion of the Gap Amplification step, we will need to understand special kinds of random walks on regular graphs. We will now discuss these random walks and prove a lemma about their properties. Note that the names "After/Before Stopping Random Walks" are not standard — we just made them up for this proof!

In both of the following definitions, $t \geq 1$ is some parameter.

Definition 5.1. *An "After Stopping Random Walk" (A.S.R.W.) in a regular graph $G = (V, E)$, starting from a random vertex, consists of the following steps:*

1. *Pick a random vertex $a \in V$ to start at.*
2. *Take a step along a random edge out of the current vertex.*
3. *Decide to stop with probability $\frac{1}{t}$. Otherwise go back to step 2.*
4. *Name the final vertex b.*

Definition 5.2. *A "Before Stopping Random Walk" (B.S.R.W.) in a regular graph $G = (V, E)$, starting from a vertex v, consists of the following steps:*

1. *Stop with probability $\frac{1}{t}$.*
2. *Take a step along a random edge out of the current vertex.*
3. *Go to step 1.*

Note that A.S.R.W.'s always have length at least 1, whereas B.S.R.W.'s could have length 0. It is also easy to see that the expected length of an A.S.R.W. is $1/t$.

A crucial lemma we will need for the Gap Amplification step is the following:

Lemma 5.3. *Let $k \geq 1$ be a fixed constant and (u, v) be a fixed edge in the regular graph $G = (V, E)$. Do an A.S.R.W. in G,* conditioned on making exactly k $u \to v$ steps. *Then:*

1. *The distribution on the final vertex b is the same as if we did a B.S.R.W. starting from v.*

2. *The distribution on the initial vertex a is same as if we did an B.S.R.W. starting from u.*

3. *a and b are independent.*

Proof. Let us start with 1. Suppose that instead of conditioning on making exactly k $u \to v$ steps, we instead condition on making *at least* k $u \to v$ steps. Then the proof of 1 becomes immediate. This is because conditioned on the fact that we have to step $u \to v$ at least k times, the instant we reach the vertex v for the kth time (before we decide to stop or not), there are no more additional conditional restrictions. Thus the distribution on b, the final vertex, just becomes the same as the distribution of the final vertex if we were doing a B.S.R.W. from v.

For an A.S.R.W., let Y be a random variable counting the number of $u \to v$ steps. Thus our previous argument demonstrates that for every $w \in V$, the probability $\Pr[b = w \mid Y \geq k]$ is a fixed constant P_w *independent of* k.

We now have the following calculation:

$$\begin{aligned} P_w &= \Pr[b = w \mid Y \geq 1] \\ &= \frac{\Pr[(b = w) \wedge (Y \geq 1)]}{\Pr[Y \geq 1]} \\ &= \frac{\Pr[(b = w) \wedge (Y = 1)] + \Pr[(b = w) \wedge (Y \geq 2)]}{\Pr[Y = 1] + \Pr[Y \geq 2]}. \end{aligned}$$

But we know that

$$\frac{\Pr[(b = w) \wedge (Y \geq 2)]}{\Pr[Y \geq 2]},$$

which is $\Pr[b = w \mid Y \geq 2]$, is also equal to P_w! It thus follows that

$$\frac{\Pr[(b = w) \wedge (Y = 1)]}{\Pr[Y = 1]}$$

is equal to P_w; i.e., $\Pr[b = w \mid Y = 1] = P_w$. It is now easy to see how to show $\Pr[b = w \mid Y = \ell]$ is also equal to P_w for each $\ell = 2, 3, 4, \ldots$: just expand out the above calculation ℓ steps and use induction. This completes the proof of part 1 of the Theorem.

Part 2 of the Theorem follows immediately from the fact that A.S.R.W.'s are completely reversible; i.e., one can get the exact same distribution by picking b at random and walking "backwards" to a, stopping with probability $1/t$ after each step.

Finally, Part 3 is easy: Look at the chain of events in the A.S.R.W.:

$a = v_0$, (DON'T STOP), v_1, (DON'T STOP), …, v_{T-1}, (DON'T STOP), v_T, (STOP) $= b$.

Conditioning on there being exactly k $u \to v$ steps just fixes some middle portion of this chain. But the chain is memoryless and reversible, so a and b can be generated independently once this middle part is fixed. $\square$

CSE 533: The PCP Theorem and Hardness of Approximation **(Autumn 2005)**

Lecture 5: PCP Theorem proof: Gap Amplification

Oct. 12, 2005

Lecturer: Ryan O'Donnell *Scribe: Chris Ré and Ryan O'Donnell*

1 Overview

Recall from last lecture that we were in the middle of Dinur's four step process for amplifying the gap of a constraint graph.

1. Degree-Reduce
2. Expanderize
3. Powering
4. Alphabet-Reduce

Recall that the powering stage is the stage where the gap of the problem increases. We will first give a sketch of the proof of this stage in order to give some intuition and then return to details for the remainder of the lecture. The details presented use a slick improvement due to J. Radakrishnan (unpublished).

2 Powering Stage (Sketch)

2.1 Parameter Effects

In this section, we will be sketchy about some details. Entering the powering stage, we have an input constraint graph denoted $(G, \mathcal{C})$. G is an a (n, d, λ)-expander, with $\lambda < d$ universal constants, and the constraints are over some fixed constant alphabet $\Sigma = \Sigma_0$. Our goal is to produce a new constraint graph $(G', \mathcal{C}')$ with a larger gap. We will denote parameters of $(G, \mathcal{C})$ (e.g. size) in the input without a prime, and constraint graph parameters of $(G', \mathcal{C}')$ in the output with a prime (e.g.. size'). Our goal for gap' is as below:

$$\mathrm{gap}' = \begin{cases} 0 & \text{if gap = 0,} \\ \geq \frac{t}{O(1)} \cdot \min(\mathrm{gap}, \frac{1}{t}) & \text{otherwise.} \end{cases}$$

As in previous stages we worry about some ancillary parameters of the graph. The effects of this stage on parameters besides the gap are summarized in Figure 1.

Output Params	Input Params
size'	= $O(\text{size})$
$\lvert\Sigma'\rvert$	$\approx \lvert\Sigma\rvert^{d^t}$
$\deg'$, λ'	we don't care

Figure 1: Effect on parameters

2.2 The beginning of the sketch

Now we begin the sketch in earnest: We need to describe the construction by which we come to the output graph G'. This construction will have an integer parameter $t \geq 1$ in it. For the moment, we will treat t like a variable but it will eventually be filled in with some universal constant, perhaps 10^6.

Vertex Set. The output graph G' will have the same vertex set; i.e. $V' = V$.

The Alphabet. The new alphabet, Σ', will be given by $\Sigma' = \Sigma^{1+d+d^2+\cdots+d^t}$. This alphabet size is chosen because $1 + d + d^2 + \cdots + d^t$ is an upper bound on the number of vertices at distance at most t from a given vertex in G, since G is d-regular. As a result, we can identify for each node, w, in the t-neighborhood of a node v a particular position in the label. We say that the value of this position corresponds to an "opinion" about the value of w in the old graph. Given an assignment $\sigma' : V' \to \Sigma'$, we write $\sigma'(w)_v \in \Sigma$ for the portion of w's label that corresponds to the node v; i.e., the "opinion" of w about v's value.

Constraints. The new edges e' will correspond to paths in G of length about t. (We are being intentionally fuzzy here.) For such a path from a to b in G, we make an edge on (a, b) in G'. The constraint we put on the edge is, roughly speaking, "check everything". For example, in checking the assignment $\sigma' : V' \to \Sigma'$, we can check that for any v on the path that $\sigma'(a)_v$ and $\sigma'(b)_v$ agree. More importantly, we can also check that for each edge (u, v) on the path, the assignment $(\sigma'(a)_u, \sigma'(b)_v)$ is acceptable for the old constraint on (u, v), namely $\mathcal{C}(u, v)$. Actually, when we eventually analyze the gap, we will only worry about such checks.

The Effect on the Gap. Notice that if $\text{gap} = 0$, then in $(G', \mathcal{C}')$ one can just use the labelling that honestly writes down the actual opinions giving the satisfying assignment for $(G, \mathcal{C})$. This labelling will pass all the constraints in $(G', \mathcal{C}')$ (indeed, the constraints were chosen with this in mind); hence gap' will still be 0, as desired. When the gap does not equal 0, we hope to make $gap' \geq \frac{t}{O(1)} \cdot \text{gap}$. (Note that if gap is already at least $\frac{1}{t}$ then this might not make sense; hence the reason for actually writing $\frac{t}{O(1)} \cdot \min(\text{gap}, \frac{1}{t})$.) So our approach is to take a best assignment for $(G', \mathcal{C}')$, call it σ', and extract from it some assignment for the original constraint graph, $\sigma : V \to \Sigma$. This assignment by definition must violate at least a gap fraction of the constraints in $(G, \mathcal{C})$. We use this fact to get a lower bound on how many constraints σ' violates in $(G', \mathcal{C}')$.

2.3 Rough analysis

Recall that given σ', we write $\sigma'(w)_v$ for the "opinion" of w about v. To define the extracted assignment $\sigma : V \to \Sigma$, given a vertex $v \in V$, we consider the vertices w at distance about t from it (again, we are intentionally fuzzy here). Each of these has an opinion $\sigma'(w)_v$ about what v's label in Σ should be, and for $\sigma(v)$ we take the plurality of these opinions.

Having defined σ, we know that it violates the constraints on some edges $F \subseteq E$ with $|F|/|E| \geq \text{gap}$. Throw away some edges from F if necessary so that $|F|/|E| = \min(\text{gap}, \frac{1}{t}) =: \gamma$.

Now we want to show that σ' has about t times more unsatisfied edge constraints. Consider an edge e' in E'; i.e., a path $a \to b$ in G. If this path passes through an edge in F, say (u, v), then there is a "good chance" that $\sigma'(a)_u = \sigma(u)$ and $\sigma'(b)_v = \sigma(v)$. This is because $\sigma(u)$ and $\sigma(v)$ are the plurality opinions about u and v among neighbors at distance around t — and a and b are such neighbors (fuzzily speaking). Further, if these things happen, then σ' violates the constraint on edge $e' = (a, b)$!

This discussion leads us to conclude that, roughly speaking,

$$\text{gap}' = \mathop{\mathbf{P}}_{e'}[\sigma' \text{ violates } e'] \geq \frac{1}{O(1)} \cdot \mathop{\mathbf{P}}_{e'}[e' \text{ passes through } F].$$

Finally, we can easily analyze the probability that a random path of length about t hits F, using the fact that G is an expander:

$$\begin{aligned}
\text{gap}' &\geq \frac{1}{O(1)} \cdot \mathop{\mathbf{P}}_{e'}[e' \text{ passes through } F] \\
&= \frac{1}{O(1)} \cdot (1 - \mathop{\mathbf{P}}_{e'}[e' \text{ completely misses } F]) \\
&= \frac{1}{O(1)} \cdot \left(1 - \left(1 - \frac{|F|}{|E|}\right)^t\right) && \text{(since } G \text{ is an expander)} \\
&\approx \frac{1}{O(1)} \cdot (1 - (1 - t\gamma)) && \text{(since } (1-\alpha)^t \approx 1 - \alpha t \text{ for } \alpha \leq 1/t) \\
&= \frac{t}{O(1)} \cdot \gamma,
\end{aligned}$$

as desired.

3 Details for the gap amplification argument

We now explicitly describe the powering step. Recall we start with $G = (V, E)$ an (n, d, λ)-expander and constraints $\mathcal{C}$ over Σ on the edges E. We are building a new constraint graph $(G', \mathcal{C}')$. As mentioned, the new vertex V' is the same as the old vertex set, and the new alphabet is $\Sigma' = \Sigma^{1+d+d^2+\cdots+d^t}$, where a label in this set is thought of as giving an "opinion" on what label

from Σ should be given to all vertices w within *shortest-path graph-distance t in G*. We will write $\mathrm{dist}_G(v, w)$ for this distance between v and w.

To define the new edge set and constraints, we will go through the notion of a *verifier*; the verifier will be a randomized process for generating an edge (a, b) for E' along with a constraint to go with it, a subset of $\Sigma' \times \Sigma'$. Note that this does not exactly give a new proper constraint graph. There are two reasons for this: First, our verifier will actually give a probability distribution over *all possible edges*; i.e., all of $V \times V$. Second, a probability distribution over edges/constraints can really only be viewed as a *weighted* constraint graph, where the weight associated to an edge is equal (or proportional) to the probability the verifier chooses that edge. In fact, we actually have to view the verifier as generating a weighted constraint graph with *multiple parallel edges*; the reason is that the verifier may produce the same pair of endpoints (a, b) in different ways with different constraints when it does its random test. So once we describe our verifier, what we've really described a *weighted* constraint graph on the complete graph, with some multiple parallel edges.

This is not so good, since we want an unweighted constraint graph (we don't mind multiple parallel edges) and since we get at least n^2 edges ($n = |V|$), rather than $O(n)$; i.e., the size has gone up way too much. Nevertheless, we will disregard these two problems for the duration of this lecture and just try to do the gap analysis for the verifier we present. In the next lecture we will show how to slightly change the verifier so that we get a weighted constraint graph with *constant* degree (the constant depends on t, d, and $|\Sigma|$). Having fixed this problem, we can also easily get an equivalent unweighted constraint graph by replacing weighted edges by (constantly many) multiple parallel unweighted edges.

3.1 The verifier

We now describe our randomized verifier by describing how it picks a random edge $e' = (a, b) \in E'$ and what test it performs on the resulting labelling $(\sigma'(a), \sigma'(b)) \in \Sigma' \times \Sigma'$. First, the verifier does an After-Stopping Random Walk (A.S.R.W. — see Lecture 4) starting from a random vertex a and ending at some random vertex b. *This walk may be of any finite length.* (Note: it ends after finitely many steps with probability 1. Also note: it can start and end at a and b in multiple different ways.) Once the verifier completes its walk, it does the following test:

For every step $u \to v$ it took, if $\mathrm{dist}_G(u, a) \leq t$ and $\mathrm{dist}_G(v, b) \leq t$ and if $(\sigma'(a)_u, \sigma'(b)_v)$ is a violation of the edge-constraint $\mathcal{C}(u, v)$ from the old constraint graph, then V rejects. If the verifier doesn't reject for any of these $u \to v$ steps, then it accepts.

Note: the verifier may take a step $u \to v$ more than once on its path; if so, it will be equally satisfied or unsatisfied about each of these steps. The reason is that it doesn't matter *when* the verifier takes the $u \to v$ step; u is either within distance t of a or it isn't, and the same goes for v and b.

Having defined our verifier, we now want to analyze its satisfiability gap. To that end, let σ' be the best assignment $\sigma' : V' \to \Sigma'$ for this verifier. We want to "extract" from it an assignment $\sigma : V \to \Sigma$. Here is how we do this:

Definition 3.1. *To define $\sigma(v)$: Consider the probability distribution on vertices $w \in V$ gotten as follows:*

- *Do a Before-Stopping Random Walk (B.S.R.W. — see Lecture 4) starting from v, ending on w.*
- *Condition on this walk stopping within t steps.*

Since any walk from v of at most t steps ends at a vertex that is within distance t of v in G, the above gives a probability distribution on vertices w that have opinions on v. *Thus we get a well-defined associated probability distribution $\mathcal{L}$ on letters of Σ — namely, the distribution $\sigma'(w)_v$. Finally, given this distribution, $\sigma(v)$ is defined to be the letter in Σ with highest probability under distribution $\mathcal{L}$. Ties can be broken by, say, lexicographical order on Σ.*

Note: this is not a random assignment; given σ', the assignment σ is fixed. Note also that this definition is a completely abstract notion we make for analysis purposes. Our overarching poly-time deterministic reduction does not need to do or understand this definition.

With σ defined, as in class we let $F \subset E$ be the subset of edges from the old constraint graph it violates. (And, we throw away some edges from F if necessary so that $|F|/|E| = \min(\text{gap}, 1/t)$.) Given this F, we introduce the notion of a *faulty step* within the verifier's walk:

Definition 3.2. *Within the verifier's A.S.R.W., we say a particular step $u \to v$ is* faulty *if:*

- $(u, v) \in F$*;*
- $\text{dist}_G(u, a) \leq t$ and $\sigma'(a)_u = \sigma(u)$*;*
- $\text{dist}_G(v, b) \leq t$ and $\sigma'(b)_v = \sigma(v)$*;*

Define also N to be the random variable counting the number of faulty steps the verifier makes in its A.S.R.W.

The key observation about this definition is that whenever $N > 0$, the verifier rejects. This is by definition of the verifier's actions. (Note that the verifier may reject in other cases too. For instance, it may reject because it made a step $u \to v$ where u is within distance t of a, v is within distance t of b, and $(\sigma'(a)_u, \sigma'(b)_v)$ is a bad assignment for $\mathcal{C}(u, v)$; however, this can happen without $\sigma'(a)_u$ equalling $\sigma(u)$ or $\sigma'(b)_v$ equalling $\sigma(v)$, which is required for faultiness.) Note also that if a verifier makes the step $u \to v$ many times, then *either all of them are faulty or none is faulty*.

We would like to show that $\mathbf{P}[N > 0]$ — and hence gap' — is large. We can do this by using the *second moment method*, a basic tool from probability:

Lemma 3.3. *(Second Moment Method) If X is a nonnegative random variable then*

$$\mathbf{P}[X > 0] \geq \frac{\mathbf{E}[X]^2}{\mathbf{E}[X^2]}.$$

Proof.

$$\mathbf{E}[X] = \mathbf{E}[X \cdot \mathbf{1}[X > 0]] \leq \sqrt{\mathbf{E}[X^2]}\sqrt{\mathbf{E}[(\mathbf{1}[X > 0])^2]} = \sqrt{\mathbf{E}[X^2]}\sqrt{\mathbf{P}[X > 0]},$$

where we used Cauchy-Schwarz in the the inequality. Rearrangement completes the proof. □

Thus the first thing we should show is that $\mathbf{E}[N]$ is large. Having done this, we will see the upper bound on $\mathbf{E}[N^2]$ and also the fix that truncates the verifier's walks in the next lecture.

Lemma 3.4. $\mathbf{E}[N] \geq \frac{1}{4|\Sigma|^2} \cdot t\frac{|F|}{|E|}$.

Proof. By linearity of expectation, it is enough to argue that for any particular edge $(u, v) \in F$, the expected number of faulty $u \to v$ steps is at least $\frac{1}{8|\Sigma|^2} \cdot \frac{t}{|E|}$. So for $(u, v) \in F$,

$$\begin{aligned}&\mathbf{E}[\#\text{ faulty } u \to v \text{ steps}] \\ &\quad = \sum_{k \geq 1} \mathbf{E}[\#\text{ faulty } u \to v \text{ steps} \mid \text{exactly } k\ u \to v \text{ steps}] \cdot \mathbf{P}[\text{exactly } k\ u \to v \text{ steps}]. \quad (1)\end{aligned}$$

As we said before, for a given random path, either all $u \to v$ steps are faulty or none is. So we have the relation

$$\mathbf{E}[\#\text{ faulty } u \to v \text{ steps} \mid \text{exactly } k\ u \to v \text{ steps}] = k \cdot \mathbf{P}[u \to v \text{ steps are faulty} \mid \text{exactly } k\ u \to v \text{ steps}].$$

We will argue that

$$\mathbf{P}[u \to v \text{ steps are faulty} \mid \text{exactly } k\ u \to v \text{ steps}] \geq \frac{1}{4|\Sigma|^2}. \quad (2)$$

Hence we can lower-bound (1) by

$$\sum_{k \geq 1} k \cdot \frac{1}{4|\Sigma|^2} \cdot \mathbf{P}[\text{exactly } k\ u \to v \text{ steps}] = \frac{1}{4|\Sigma|^2} \cdot \mathbf{E}[\text{number of } u \to v \text{ steps}] = \frac{1}{4|\Sigma|^2} \cdot \frac{t}{2|E|},$$

where the last step follows from: a) the expected total number of steps is easily seen to be t; b) each step is equally likely to be any of the $2|E|$ possibilities (this uses the fact that G is regular); c) linearity of expectation.

It now remains to prove (2). So suppose we condition the verifier's walk on making exactly k $u \to v$ steps. Since (u, v) is in F we have that $u \to v$ steps are faulty for this walk iff:

(i) a is within distance t of u in the graph G and $\sigma'(a)_u = \sigma(u)$;

(ii) b is within distance t of v in the graph G and $\sigma'(b)_v = \sigma(v)$.

We now invoke the lemma from Lecture 4 which says that conditioned on the walk taking exactly k $u \to v$ steps, a is distributed as a B.S.R.W. from u, b is distributed as a B.S.R.W. from v, and a and b are independent. Our task is to show that $\mathbf{P}[\text{(i) and (ii)}] \geq \frac{1}{4|\Sigma|^2}$; since a and b are independent, this probability is equal to $\mathbf{P}[\text{(i)}] \cdot \mathbf{P}[\text{(ii)}]$. We will show that $\mathbf{P}[\text{(ii)}] \geq \frac{1}{2|\Sigma|}$ and the proof for (i) is the same; thus this will complete the proof.

So finally, we must prove $\mathbf{P}[\text{(ii)}] = \mathbf{P}[\text{dist}(b, v) \leq t \text{ and } \sigma'(b)_v = \sigma(v)] \geq \frac{1}{2|\Sigma|}$. By the lemma, if we let w denote a random vertex generated by taking a B.S.R.W. starting at v, then

$$\mathbf{P}[\text{dist}(b, v) \leq t \text{ and } \sigma'(b)_v = \sigma(v)] = \mathbf{P}[\text{dist}(w, v) \leq t \text{ and } \sigma'(w)_v = \sigma(v)]. \tag{3}$$

Now consider the B.S.R.W. that generated w and condition on it *having taken at most t steps*. So

$$\begin{aligned}
(3) &= \mathbf{P}[\text{dist}(w, v) \leq t \text{ and } \sigma'(w)_v = \sigma(v) \mid \text{at most } t \text{ steps taken to generate } w] \times \\
&\qquad \times \mathbf{P}[\text{at most } t \text{ steps taken to generate } w] \\
&= \mathbf{P}[\sigma'(w)_v = \sigma(v) \mid \text{at most } t \text{ steps taken to generate } w] \times \mathbf{P}[\text{at most } t \text{ steps taken to generate } w] \\
&\geq (1/2) \cdot \mathbf{P}[\sigma'(w)_v = \sigma(v) \mid \text{at most } t \text{ steps taken to generate } w], \qquad (4)
\end{aligned}$$

where the last step is because a B.R.S.W. stop within t steps with probability at least $1-(1-1/t)^t \geq 1/2$. But now, finally, if we look at the probability in (4), we see that the distribution on w is *precisely the distribution used in defining $\sigma(v)$* — specifically, that generated by taking a B.S.R.W. from v and conditioning on stopping within t steps. Hence the probability in (4) is actually that the most common (plurality) event occurs for a Σ-valued random variable; this is clearly at least $\frac{1}{|\Sigma|}$, so we are done. □

CSE 533: The PCP Theorem and Hardness of Approximation **(Autumn 2005)**

Lecture 6: Powering Completed, Intro to Composition

Oct. 17, 2005

Lecturer: Ryan O'Donnell and Venkat Guruswami *Scribe: Anand Ganesh*

1 Powering Completed

In the previous lecture we began the Powering step; this step takes $(G, \mathcal{C})$, a constraint graph on an (n, d, λ)-expander over a constant-sized alphabet Σ, and outputs $(G', \mathcal{C}')$. The properties of the new constraint graph are that $\text{size}' = O(\text{size})$, the alphabet size remains a constant (albeit a much much larger one), $\text{gap} = 0 \Rightarrow \text{gap}' = 0$, and the main property:

$$\text{gap}' \geq \frac{t}{O(1)} \cdot \min(\text{gap}, 1/t).$$

Here t is a fixed constant parameter. The transformation is based on the idea of constructing G' from G by having edges in G' correspond to walks of length about t in the original graph.

Let us go over the construction of G' in more detail. As we said in the previous lecture, the new alphabet is $\Sigma' = \Sigma^{1+d+d^2+d^3...+d^t}$. The new edge set E' and constraint set C' are defined as follows:

- E': Pick a random vertex a. Do an A.S.R.W. from a, ending on b. This generates a weighted edge $e' = (a, b)$. As we noted, this seems like it might be a problem, since we get a) weighted edges, and b) too many edges. We discuss this shortly.
- $\mathcal{C}'$: Here is the test that is then applied to the labelling $\sigma' : V' \to \Sigma'$:
  - If the number of steps in the A.S.R.W. was greater than $B := (10ln|\Sigma|)t$, then accept.
  - Otherwise, for each step $u \to v$ along the path, if $dist_G(a, u) \leq t$ and $dist_G(b, v) \leq t$ (i.e. the vertices are close enough that there will be opinions), *and* if $(\sigma'(a)_u, \sigma'(b)_v)$ is violating for the old constraint on $(u, v) \in E$, reject

Fixing the problems: Given this construction of a weighted constraint graph (with multiple edges and edges between all possible pairs!), throw away all paths $e' \in E'$ corresponding to walks of length greater than B. Since these all had *always-satisfied* constraints, this can only cause the gap to go up. Having done this, we get:

$$deg(G) \leq 1 + d + d^2 + d^3... + d^B = \text{constant}$$
$$size' \leq O(size).$$

The only problem left is that the edges still have weights, and we want unweighted constraint graphs. However note that all the weights are simply rational numbers depending only on the universal constants d and t; hence we can replace them appropriately with parallel unweighted edges. Thus we have fixed the annoying details associated with our randomized way of describing the edges E' and can proceed to analyze the gap.

We now need to show that $\forall \sigma' : V' \to \Sigma'$,

$$\Pr[\text{test rejects } \sigma'] \geq \frac{t}{O(1)} \min(\text{gap}, \frac{1}{t}).$$

So let σ' be the "best" assignment for our $\mathcal{C}'$. We extract an assignment $\sigma : V \to \Sigma$ from it based on the plurality vote method analyzed in the previous lecture. Let $F \subset E$ be the set of edges in G violated by σ; we know that $|F|/|E| \geq \text{gap}$. Throw away some edges from F is necessary so that $|F|/|E| = \min(\text{gap}, 1/t)$.

We will now recall the notion of a "faulty step" from the previous lecture; however we will also introduce the notion of a "faulty* step", which is needed to analyze our B-truncated verifier.

Definition 1.1. *Let $e' : a \to b$ be a random path as chosen by our verifier. Step $u \to v$ is* faulty *if the following hold:*

- $(u, v) \in F$,
- $dist_G(a, u) \leq t$ *and* $\sigma'(a)_u = \sigma(u)$,
- $dist_G(b, v) \leq t$ *and* $\sigma'(b)_v = \sigma(v)$.

We further define a step to be faulty* *if*

- *the step is faulty,*
- *the number of steps in the overall $a \to b$ walk was at most B.*

Let us also define some random variables based on the verifier's A.S.R.W.:

Definition 1.2.

- $N =$ *number of faulty steps.*
- $N^* =$ *number of faulty* steps.*
- $S =$ *total number of steps.*
- $N_F =$ *number of steps that were in F.*

Note that by the definitions we have

$$N^* = N \cdot \mathbf{1}_{\{S \leq B\}},$$

and

$$N^* \leq N \leq N_F.$$

1.1 Analysis

The key point of the definition of faulty* steps is that whenever the verifier picks a random walk that has a faulty* step (i.e., whenever $N^* > 0$), the verifier rejects σ'. (Note that the verifier may still reject even if it has no faulty* steps.) Thus we have

$$\mathrm{gap}' = \Pr[\text{test rejects } \sigma'] \geq \Pr[N^* > 0] \geq \frac{\mathrm{E}[N^*]^2}{\mathrm{E}[(N^*)^2]},$$

where we used the Second-Moment Method in the last step as discussed in the last lecture. To complete the proof we need to show that $\mathrm{gap}' \geq \frac{t}{O(1)} \cdot \frac{|F|}{|E|}$. Hence we are done if we can show the following two lemmas:

Lemma 1.3. $\mathrm{E}[N^*] \geq \frac{t}{8|\Sigma|^2} \cdot \frac{|F|}{|E|}$

Lemma 1.4. $\mathrm{E}[(N^*)^2] \leq O(1) \cdot t \cdot \frac{|F|}{|E|}$

Proof. (Lemma 1.3.) To prove the lower bound on $\mathrm{E}[N^*]$ we use the lemma from the last lecture that said $\mathrm{E}[N] \geq \frac{t}{4|\Sigma|^2} \cdot \frac{|F|}{|E|}$. We have

$$\begin{aligned}
\mathrm{E}[N^*] &= \mathrm{E}[N \cdot \mathbf{1}_{\{S \leq B\}}] \\
&= \mathrm{E}[N \cdot (1 - \mathbf{1}_{\{S > B)\}}] \\
&= \mathrm{E}[N] - \mathrm{E}[N \cdot \mathbf{1}_{\{S > B\}}] \\
&\geq \frac{t}{4|\Sigma|^2} \cdot \frac{|F|}{|E|} - E[N \cdot \mathbf{1}_{\{S > B\}}],
\end{aligned}$$

using the earlier lemma. Now,

$$\begin{aligned}
\mathrm{E}[N \cdot \mathbf{1}_{\{S > B\}}] &= \Pr[S > B] \cdot \mathrm{E}[N \mid S > B] \\
&= (1 - 1/t)^B \cdot \mathrm{E}[N \mid S > B] \\
&\geq \exp(-B/t) \cdot \mathrm{E}[N_F \mid S > B] \\
&= \exp(-B/t) \cdot \mathrm{E}[S \mid S > B] \cdot \frac{|F|}{|E|} \\
&= \exp(-B/t) \cdot (B + t) \cdot \frac{|F|}{|E|} \\
&\geq \frac{1}{|\Sigma|^{10}} \cdot (20 \ln |\Sigma| \cdot t) \cdot \frac{|F|}{|E|} \qquad \text{(by definition of } B) \\
&\geq \frac{t}{8|\Sigma|^2} \cdot \frac{|F|}{|E|},
\end{aligned}$$

since $(20 \ln |\Sigma|)/|\Sigma|^{10} \leq 1/(8|\Sigma|^2)$. Combining the above two calculations completes the proof. □

Proof. (Lemma 1.4) The proof of this lemma is the only place where we use the fact that G is an (n, d, λ)-expander. In fact, this is all we use — that in a length L random walk on an expander, the number of times you hit a fixed set of edges is about what you would get if you just picked L random edges. In particular, we start with the trivial upper bound

$$\mathrm{E}[(N^*)^2] \leq \mathrm{E}[(N_F)^2].$$

Let us express $N_f = \sum_{i=1}^{\infty} \chi_i$, where $\chi_i = \mathbf{1}[i\text{th step is in } F]$. We have:

$$\begin{aligned}\mathrm{E}[(N_F)^2] &= \sum_{i,j=1}^{\infty} \mathrm{E}[\chi_i \cdot \chi_j] \\ &\leq 2\sum_{i=1}^{\infty} \Pr[\chi_i = 1] \cdot \sum_{j \geq i} \Pr[\chi_j = 1 \mid \chi_i = 1] \qquad (*)\end{aligned}$$

Now $\Pr[\chi_j = 1 \mid \chi_i = 1]$ is 1 if $j = i$; otherwise it equals

$$\begin{aligned}&\Pr[\text{the walk takes at least } j-i \text{ more steps}] \\ &\qquad \times \Pr[\text{a walk, starting from a random } F \text{ endpoint, takes its } (j-i)\text{th step in } F].\end{aligned}$$

The first quantity here is just $(1 - 1/t)^{j-i}$. The second quantity, from Lecture 3's expander lemma on this subject, is at most $|F|/|E| + (\lambda/d)^{j-i-1}$. Now substituting this into (*), we get:

$$\begin{aligned}\mathrm{E}[(N^*)^2] &\leq 2\sum_{i=1}^{\infty} \Pr[\chi_i = 1] \cdot \left(1 + \sum_{\ell=1}^{\infty} (1-1/t)^{\ell} \left(\frac{|F|}{|E|} + (\lambda/d)^{\ell-1}\right)\right) \\ &\quad 2\sum_{i=1}^{\infty} \Pr[\chi_i = 1] \cdot \left(1 + \sum_{\ell=1}^{\infty} (1-1/t)^{\ell} \cdot \frac{|F|}{|E|} + \sum_{\ell=1}^{\infty} (\lambda/d)^{\ell-1}\right) \\ &\leq 2\sum_{i=1}^{\infty} \Pr[\chi_i = 1] \cdot \left(1 + (t-1) \cdot \frac{|F|}{|E|} + O(1)\right) \qquad \text{(since } \lambda < d \text{ are consts)} \\ &\leq O(1) \cdot \sum_{i=1}^{\infty} \Pr[\chi_i = 1] \qquad \text{(since } |F|/|E| \leq 1/t) \\ &= O(1) \cdot \mathrm{E}[N_F] \\ &= O(1) \cdot t\frac{|F|}{|E|},\end{aligned}$$

as claimed. □

2 Introduction to Composition

We are now at stage 4 of the proof of the PCP theorem called Alphabet Reduction or Composition. Given a constraint graph with a large alphabet size, the problem is to reduce the alphabet size

($\Sigma^{d^t} \to \Sigma_0$ where $|\Sigma_0|$ is an absolute constant, say 64) without adversely affecting other parameters like the gap. In simplified terms, the composition step may be thought of as a recursive step (with some extra features) within the larger PCP reduction as we will describe below.

Recall that the PCP may be cast in the following form. It is a reduction P such that maps a Boolean constraint (a 3SAT formula, or a Boolean circuit) Φ into a constraint graph $(G, \mathcal{C})$ over a fixed alphabet Σ_0 such that

$$\begin{aligned} P \ : \ \Phi &\to (G, \mathcal{C}) \text{ such that} \\ \Phi \text{ satisfiable} &\implies (G, \mathcal{C}) \text{ satisfiable} \\ \Phi \text{ not satisfiable} &\implies \mathrm{gap}(G) > \epsilon \text{ i.e. } < 1 - \epsilon \text{ of the constraints in } \mathcal{C} \text{ are satisfiable} \end{aligned}$$

Consider a constraint graph $H = G^t$ obtained after the powering step. Let c_e be the constraint on edge e. This is a binary constraint over a large alphabet (like Σ^{d^t}). We can express it as a Boolean constraint Φ_{c_e} over several Boolean variables (using some standard encoding), and apply a PCP reduction as above, call it P_e, to this constraint to get a new set of binary constraints over the alphabet Σ_0. Thus, we can reduce the alphabet size, and if the original c_e was not satisfied at least an ϵ fraction of the newly produced constraints over Σ_0 must be unsatisfied by any assignment. Therefore, one also expect that the new gap is at least $\epsilon \cdot \mathrm{gap}(H)$, and the alphabet is now Σ_0.

Of course, our whole goal is to construct a PCP reduction, so how can we use a PCP reduction recursively without falling prey to a circular argument? The key is that we will apply this "inner" PCP reduction only to constraints of *constant size* and thus the reduction can be arbitrarily inefficient (and hence perhaps easier to construct). Therefore, the hope would be to construct from scratch a highly inefficient PCP reduction, and use it as above.

Consistency. However, there is a subtle issue which makes the above recursion not as straightforward as we suggested. Suppose $e = (u, v)$ and $e' = (v, w)$ are two edges in the constraint graph H that share a vertex v. The reductions P_e and $P_{e'}$ ensure that the constraints c_e and $c_{e'}$ are both individually satisfiable. However, we need to ensure that the constraints in H are all satisfied by a single, common assignment to the variables. In particular, this means that we need to check not only that c_e and $c_{e'}$ are satisfiable, but that they are satisfied by assignments which are consistent on the shared variable v. This consistency will be achieved by imposing additional requirements on the PCP reduction using an entity called the Assignment Tester.

Roughly speaking, an assignment tester checks that a constraint is satisfied by a *particular* assignment to its variables. So in the above recursive appraoch, we assume that we are given an assignment $\sigma(v)$ to each of the variables in H, and the assignment tester corresponding to edge $e = (u, v)$ must check that $(\sigma(u), \sigma(v))$ satisfies the constraint c_e. Now consider the case where $(\sigma(u), \sigma(v))$ differs from a satisfying assignment to c_e in just one bit. Then, most constraints of the assignement tester may accept, whereas we want a constant fraction of them to reject. Thus this is too stringent a task to impose on the assignment tester.

We relax the requirement so that the assignment tester must check *proximity* of the claimed assignment to a satisfying assignment of the constraint in the Hamming metric. As we will see

in the next two lectures, this task becomes feasible. But let us see how this relaxation affects the composition idea.

Consider edges $e = (u, v)$ and $e = (v, w)$ sharing a vertex v. Given vertex assignments for vertices u, v, w, P_e checks that $\sigma(v)$ is close to x_{1v} that (together with some assignment to u) satisfies c_e. $P_{e'}$ checks that $\sigma(v)$ is close to x_{2v} thatsatisfies $c_{e'}$. We want to enforce consistency on the label to v, i.e., $x_{1v} = x_{2v}$ — this is the goal. In other words, for any assignment $\sigma(v)$ there should be a unique legal, satisfying assignment that is very close to $\sigma(v)$. This condition can be met if and only if the legal assignments to the vertices are all pairwise far apart from each other, or in other words they form an error-correcting code. Hence, codes enter naturally into this framework.

CSE 533: The PCP Theorem and Hardness of Approximation **(Autumn 2005)**

Lecture 7: Composition and Linearity Testing

Oct. 17, 2005

Lecturer: Venkat Guruswami *Scribe: Anand Ganesh & Venkat Guruswami*

Last Class

- End Gap Amplification
- High level view of Composition

Today

- Formal Definition of AT = Assignment Tester
- Composition Theorem

1 Composition

We saw informally in the previous lecture that the composition step was recursion with a twist: the "inner" the verifier needed to perform the following "assignment testing" task: Given an assignment a to the variables of a constraint Φ, check if a is close to a satisfying assignment of Φ. The formal definition follows. We say that two strings x, y are δ-far from each other if they differ on at least a fraction δ of coordinates.

Definition 1.1 (Assignment Tester)**.** *A q-query Assignment Tester $AT(\gamma > 0, \Sigma_0)$ is a reduction algorithm P whose input is a Boolean circuit Φ over Boolean variables X and P outputs a system of constraints Ψ over X and set Y of auxiliary variables such that*

- *Variables in Y take values in Σ_0*
- *Each $\psi \in \Psi$ depends on at most q variables in $X \cup Y$*
- $\forall a : X \to \{0,1\}$
  - *If $\phi(a) = 1$, then $\exists b : Y \to \Sigma_0$ such that $a \cup b$ satisfies all $\psi \in \Psi$*
  - *If assignment 'a' is δ-far from every a' such that $\phi(a') = 1$ then $\forall b : Y \to \Sigma_0$, atleast $\gamma_0\delta$ fraction $\psi \in \Psi$ are violated by $a \cup b$*

We first observe that the above definition is stronger than a regular PCP reduction:

- Φ satisfiable $\implies$ $\exists a \cup b$ that satisfy all constraints in Ψ

- Φ not satisfiable $\implies$ every assignment $a : X \to \{0,1\}$ is 1-far from any satifsying assignment (since no satisfying assignment exists!), and hence for every $b : Y \to \Sigma_0$, $a \cup b$ violates $\Omega(1)$ fraction of constraints in Ψ. In particular, every assignment $a \cup b$ to the variables of Ψ violates $\Omega(1)$ fraction of the constraints, like in a PCP reduction.

Theorem 1.2 (Composition Theorem)**.** *Assume existence of 2-query Assignment tester $P(\gamma > 0, \Sigma_0)$. Then $\exists \beta > 0$ (dependent only on P and $poly(size(G))$) such that any constraint graph $(G, \mathcal{C})_\Sigma$ can transformed in time polynomial in the size of G into a constraint graph $(G', \mathcal{C}')_{\Sigma_0}$ denoted by $G \circ P$ such that*

- $size(G') \leq O(1) size(G)$
- $\text{gap}(G) = 0 \implies \text{gap}(G') = 0$
- $gap(G') \geq \beta \cdot gap(G)$

Proof. The basic idea is to apply the AT P to each of the constraints $c \in \mathcal{C}$ and then define the new constraint graph G' based on the output of P. Since the AT expects as input a constraint over Boolean variables, we need to first express the constraints of G with Boolean inputs. For this, we encode the elements of Σ as a binary string.

The trivial encoding uses $\log(|\Sigma|)$ bits. Instead we will use an error correcting code $e : \Sigma \to \{0,1\}^l$ where $l = O(\log |\Sigma|)$ of relative distance $\rho = 1/4$, i.e., with the following property:

$$\begin{aligned} \forall x, y, x \neq y &\implies e(x) \text{ is } \rho\text{-far from } e(y) \\ \text{i.e. } x \neq y &\implies \Delta(e(x), e(y)) \geq \rho \cdot l \end{aligned}$$

Let $[u]$ denote the block of Boolean variables supposed to represent the bits of the encoding of u's label. For a constraint $c \in \mathcal{C}$ on variables u, v of G, define the constraint $\tilde{c}$ on the $2l$ Boolean variables $[u] \cup [v]$ as follows:

$$\begin{aligned} \tilde{c}(a,b) = 1 \text{ iff } \exists\, \alpha \in \Sigma \text{ and } \alpha' \in \Sigma \text{ such that the following hold:} \\ e(\alpha) = a \\ e(\alpha') = b \\ c(\alpha, \alpha') = 1. \end{aligned}$$

Let $\tilde{c} : \{0,1\}^{2l} \to \{0,1\}$ be regarded as a Boolean circuit and fed to a 2-query AT. The output is a list of constraints Ψ_c which can regarded as a constraint graph over Σ_0, call it $(G_c = (V_c, E_c), \mathcal{C}_c)$ (where $[u] \cup [v] \subset V_c$), with the two variables in each constraint taking the place of vertices in the constraint graph. To get the new constraint graph G', we will paste together such constraint graphs G_c obtained by applying the 2-query AT to each of the constraints of the original constraint graph.

Formally, the new constraint graph is $(G' = (V', E'), \mathcal{C}')$ over Σ_0 where

- $V' = \cup_{c \in \mathcal{C}} V_c$
- $E' = \cup_{c \in \mathcal{C}} E_c$

- $\mathcal{C}' = \cup_{c \in \mathcal{C}} \mathcal{C}_c$

We will assume wlog that $|E_c|$ is the same for every $c \in C$ (this can be achieved by duplicating edges if necessary).

We will now prove that the graph G' obtained by the reduction above satisfies the requirements of the Composition Theorem (**??**). Clearly, since the size of G' is at most a constant times larger than that of G, since each edge in G is replaced by the output of the assignment tester on a constant-sized constraint, and thus by a graph of constant (depending on $|\Sigma|$) size. Also, G' can be produced in time polynomial in the size of G.

The claim that $\mathrm{gap}(G') = 0$ when $\mathrm{gap}(G) = 0$ is also obvious – beginning with a satisfying assignment $\sigma : V \to \Sigma$, we can label the variables in $[u]$ for each $u \in V$ with $e(\sigma(u))$, and label the auxiliary variables introduced by the assignment testers P in a manner that satisfies all the constraints (as guaranteed the property of the assignment tester when the input assignment satisfies the constraint).

It remains to prove that $\mathrm{gap}(G') \geq \beta \cdot \mathrm{gap}(G)$ for some $\beta > 0$ depending only on the AT.

Let $\sigma' : V' \to \Sigma_0$ be an arbitrary assignment. We want to show that σ' violates at leats a fraction $\beta \cdot \mathrm{gap}(G)$ of the constraints in $\mathcal{C}'$. First we extract an assignment $\sigma : V \to \Sigma$ from σ' as follows: $\sigma(u) = \arg\min_a \Delta(\sigma'([u]), e(a))$, i.e., we pick the closest codeword to the label to the block of variables (here we assume without loss of generality that σ' assigns values in $\{0, 1\}$ to variables in $[u]$ for all $u \in V$).

We know that σ violates at a fraction $\mathrm{gap}(G)$ of constraints in $\mathcal{C}$. Let $c = c_e \in \mathcal{C}$ be such a violated constraint where $e = (u, v)$. We will prove that at least a $\gamma \cdot \rho/4$ fraction of the constraints of G_c are violated by σ'. Since the edge sets E_c all have the same size for various $c \in C$, it follows that σ' violates at least a fraction $\gamma \cdot \rho/4\mathrm{gap}(G)$ of constraints of G'. This will prove the composition theorem with the choice $\beta = \gamma \cdot \rho/4$.

By the property of the assignment tester P, to prove at least $\gamma \cdot \rho/4$ of the constraints of G_c are violated by σ', it suffices to prove the following.

Claim: $\sigma'([u] \cup [v])$ is at least $\rho/4$-far from any satisfying assignment to $\tilde{c}$.

Proof of Claim: Let $(\sigma''([u]), \sigma''([v]))$ be a satisfying assignment for $\tilde{c}$ that is closest to $\sigma'([u] \cup [v])$. Any satisfying assignment to $\tilde{c}$ must consist of codewords of the error-correcting code e. Therefore, let $\sigma''([u]) = e(a)$ amd $\sigma''([v]) = e(b)$. Moreover, $c(a, b) = 1$. Since σ violates c, we have $c(\sigma(u), \sigma(v)) = 0$. It follows that either $a \neq \sigma(u)$ or $b \neq \sigma(v)$, let us say the former for definiteness. We have

$$\rho \leq \Delta(e(a), e(\sigma(u))) \leq \Delta(e(a), \sigma'([u])) + \Delta(\sigma'([u]), e(\sigma(u))) \leq 2\Delta(e(a), \sigma'([u]))$$

where the last step follows since $e(\sigma(u))$ is the codeword closest to $\sigma'([u])$. Recalling, $e(a) = \sigma''([u])$, we find that at least a $\rho/2$ fraction of the positions $\sigma'([u])$ must be changed to obtain a satisfying assignment to $\tilde{c}$. It follows that $\sigma'([u] \cup [v])$ is at least $\rho/4$-far from any satisfying assignment to $\tilde{c}$.

This completes the proof of the claim, and hence also that of Theorem 1.2. □

The composition theorem needed a 2-query AT. We now show that bringing down the number of queries to 2 is easy once we have a q-query AT for some constant q.

Lemma 1.3. *Given a q-query Assignment Tester AT over $\Sigma_0 = \{0,1\}, \gamma_0 > 0$, it is possible to construct a 2-query AT over alphabet $\Sigma_0' = \{0,1\}^q$ and $\gamma_0' = \frac{\gamma_0}{q}$.*

Proof. Let the q-query Assignment Tester AT be on Boolean variables $X \cup Y$ (where Y is the set of auxiliary variables), with set of constraints Ψ. Define 2-query AT as follows. The auxiliary variables are $Y \cup Z$ where $Z = \{z_\psi | \psi \in \Psi\}$ is a set of variables over the alphabet $\Sigma_0' = \{0,1\}^q$, and the set of constraints Ψ' include for each $\psi \in \Psi$ a set of q constraints on two variables: $(z_\psi, v_1), (z_\psi, v_2), ...(z_\psi, v_q)$ where $v_1, v_2, \ldots, v_q$ are the variables on which ψ depends (if ψ depends on fewer than q variables, we just repeat one of them enough times to make the number q). The constraint (z_ψ, v_i) is satisfied by (a, b) a satisfies ψ and a is consistent with b on the value given to v_i.

Clearly, if all constraints in Ψ can be saisfied by an assignment to $X \cup Y$, then it can be extended in the obvious way to Z to satisfy all the new constraints. Also, if $a : X \to \{0,1\}$ is δ-far from satisfying the input circuit Φ to the AT, then for every $b : Y \to \{0,1\}$, at least $\gamma_0\delta$ fraction of $\psi \in \Psi$ are violated. For each such ψ, for any assignment $c : Z \to \{0,1\}^q$, at least one of the q constraints that involve z_ψ must reject. Thus, at least a fraction $\frac{\gamma_0\delta}{q}$ of the new constraints reject. □

Later on, we will give a 6-query AT over the Boolean alphabet. By the above, this also implies a 2-query AT over the alphabet $\{1, 2, \ldots, 64\}$.

2 Linearity Testing

We will now take a break from PCPs and do a self-contained interlude on "linearity testing". Consider a function $f : \{0,1\}^n \to \{0,1\}$ as a table of values, the question we now consider is "Is f linear ?". Such questions are part of a larger body of research called property testing. First, we define what we mean by a linear function.

Definition 2.1. *(Linear functions) A function $f : \{0,1\}^n \to \{0,1\}$ is linear if $\exists S \subset \{1,2,...n\}$ such that $f(x) = \bigoplus_{i \in S} x_i$. Or equivalently, f is linear if there exists $a \in \{0,1\}^n$ such that $f(x) = \bigoplus_{i=1}^n a_i x_i$.*

Fact 2.2. *The following two statements are equivalent:*

- *f is linear*
- $\forall x, y : f(x+y) = f(x) + f(y)$

For $S \subset \{1,2,...n\}$, define $L_S : \{0,1\}^n \to \{0,1\}$ as $L_S(x) = \bigoplus_{i \in S} x_i$. Say $L_s(X) = \sum_{i \in S} X_i$. Given access to the truth table of a function f, linearity testing tries to distinguish between the following cases, using very few probes into the truth table of f:

- $f = L_S$ for some S
- f is "far-off" from L_S for *every* S

A randomized procedure for Linearity Testing uses 2.2 above. Instead of testing whether $f(x+y) = f(x) + f(y)$ for every pair x, y, we pick one pair (x, y) at random and apply the following test: Is $f(x+y) = f(x)+f(y)$? Thus we look at the value of f on only 3 places. We will explore actual guarantees that this test provides in the next lecture, and go on to connect this with the proof of the PCP theorem.

CSE 533: The PCP Theorem and Hardness of Approximation **(Autumn 2005)**

Lecture 8: Linearity and Assignment Testing

26 October 2005

Lecturer: Venkat Guruswami *Scribe: Paul Pham & Venkat Guruswami*

1 Recap

In the last class, we covered the Composition Theorem except for the $O(1)$-query assignment tester (AT). Today we will develop some machinery relating to linearity testing to be used in the construction of such an AT. This can be treated as a self-contained lecture on linearity testing and the necessary Fourier analysis.

2 Linearity Testing

We work in an n-dimensional binary vector space $\{0,1\}^n$ with inner product $x \cdot y = \sum_{i=1}^{n} x_i y_i$ defined as usual, where the summation is done modulo 2 (in other words, it is the binary $\oplus$ operator). The binary $\oplus$ operator will denote bitwise XOR on two vectors. The linear functions on $\{0,1\}^n$ are of the form $L(x) = a \cdot x$ for some $a \in \{0,1\}^n$. There are 2^n such functions. If $S \subseteq \{1,2,\ldots,n\}$ is the set $\{i : S \ni i\}$, then clearly $L(x) = \sum_{i \in S} x_i$ (again, the summation is in the field $\{0,1\}$). We will use such subsets to index linear functions — the linear function L_S corresponding to a subset $S \subset \{1,2,\ldots,n\}$ is defined as

$$L_S(x) = \sum_{i \in S} x_i$$

Definition 2.1 (linear function)**.** *A function $f : \{0,1\}^n \to \{0,1\}$ is linear if*

$$\forall_{x,y \in \{0,1\}^n} : f(x+y) = f(x) + f(y) \tag{1}$$

(where $x + y$ denotes coordinatewise addition mod 2).

Alternatively, a function f is linear if it is equal to L_S for some $S \subset \{1,2,\ldots,n\}$.

The goal of *linearity testing* is then: given $f : \{0,1\}^n \to \{0,1\}$ as a table of 2^n values, check whether f is linear using just a few probes into the table f. Clearly, we cannot check that f is exactly linear without looking at the entire table, so we will aim to check whether f is close to a linear function. A natural test is the Blum-Luby-Rubinfeld (BLR) test where we check the above condition (1 for a single triple $(x, y, x+y)$ with x, y chosen randomly and independently. Formally the BLR test proceeds as follows:

1. Pick $x, y \in \{0,1\}^n$ uniformly and independently at random.

2. If $f(x+y) = f(x) + f(y)$ accept, otherwise reject.

We can describe the distance/closeness between two functions as follows.

Definition 2.2. *We say two functions f and g are δ-*far *if they differ in at least a fraction δ of places, $0 \leq \delta \leq 1$, i.e.,*

$$\Pr_{x \in \{0,1\}^n} [f(x) \neq g(x)] \geq \delta$$

We say two functions f and g are δ-close if they differ in at most a fraction δ of places, $0 \leq \delta \leq 1$, i.e.,

$$\Pr_{x \in \{0,1\}^n} [f(x) \neq g(x)] \leq \delta$$

The completeness of the above test is obvious.

Theorem 2.3 (BLR Completeness)**.** *If f is linear, the BLR test accepts with probability 1.*

In the rest of this lecture we will show the following on the efficacy of the test in detecting the non-linearity of f as a function of the distance of f from linearity.

Theorem 2.4 (BLR Soundness)**.** *If f is δ-far from every linear function then the BLR test rejects with probability at least δ.*

3 Notation Change

We now introduce a change in notation from Boolean binary values in $\{0,1\}$ to $\{1,-1\}$ which turns out to be more convenient. The transformation for $a \in \{0,1\}$ is:

$$a \to (-1)^a$$

which maps 0 to 1, 1 to -1. The advantage with this change is that the xor (or summation mod 2) operation becomes multiplication.

$$a + b \to (-1)^{a+b} = (-1)^a (-1)^b$$

We now consider functions $f : \{-1,1\}^n \to \{-1,1\}$ and have a new form for the linear function associated with a subset $S \subset \{1,2,\ldots,n\}$:

$$\chi_S(x) = \prod_{i \in S} x_i \tag{2}$$

and a new linearity test that checks

$$f(x \cdot y) = f(x) \cdot f(y)$$

for randomly chosen $x, y \in \{1,-1\}^n$, where $x \cdot y$ denotes coordinatewise multiplication, i.e., $(x \cdot y)_i = x_i y_i$.

The expression $f(x)f(y)f(xy)$ equals 1 if the test accepts for that choice of x, y, and equals -1 if the test rejects. The quantity

$$\left(\frac{1 - f(x)f(y)f(xy)}{2}\right)$$

is thus an indicator for whether the test accepts. It follows that the probability that the BLR test rejects using this new notation can be expressed as:

$$\Pr[\text{ BLR test rejects}] = \mathop{\mathrm{E}}_{x,y\in\{1,-1\}^n}\left[\frac{1 - f(x)f(y)f(xy)}{2}\right] \tag{3}$$

In order to calculate the value of this expectation, we will need some background in discrete Fourier analysis.

4 Fourier Analysis on Discrete Binary Hypercube

Consider the vector space $\mathbb{G}$ consisting of all n-bit functions from $\{-1, 1\}^n$ to the real numbers:

$$\mathbb{G} = \{g \mid g : \{-1, 1\}^n \to \mathbb{R}\}$$

$\mathbb{G}$ has dimension 2^n, and a natural basis for it are the indicator functions $e_a(x) = \mathbb{1}(x = a)$ for $a \in \{1, -1\}^n$. The coordinates of $g \in \mathbb{G}$ in this basis is simply the table of values $g(a)$ for $a \in \{1, -1\}^n$.

We will now describe an alternate basis for $\mathbb{G}$. We begin with an inner product on this space defined as follows:

$$\langle f, g\rangle = \frac{1}{2^n}\sum_{x\in\{-1,1\}^n} f(x)g(x) \tag{4}$$

The linear functions $\chi_S(x)$ for various subsets S form an orthonormal basis with respect to the above inner product. Since $|\chi_S(x)| = 1$ for every S, x, clearly $\langle\chi_S, \chi_S\rangle = 1$ for all $S \subseteq \{1, 2, \ldots, n\}$. The following shows that different linear functions are orthogonal w.r.t the inner product (4).

Lemma 4.1.

$$S \neq T \to \langle\chi_S, \chi_T\rangle = 0$$

Proof:

$$\begin{aligned}
\langle\chi_S, \chi_T\rangle &= \frac{1}{2^n}\sum_{x\in\{-1,1\}^n} \chi_S(x)\chi_T(x) \\
&= \frac{1}{2^n}\sum_{x\in\{-1,1\}^n} \left(\prod_{i\in S} x_i\right)\left(\prod_{i\in T} x_i\right) \\
&= \frac{1}{2^n}\sum_{x\in\{-1,1\}^n} \left(\prod_{i\in S\triangle T} x_i\right) \\
&= 0
\end{aligned}$$

$S \triangle T$ denotes the symmetric difference of S and T. This is not empty because we have specified $S \neq T$. The last step follows because we can always pair any x with an $\tilde{x}$ such that $x_j = -\tilde{x}_j$ for a fixed $j \in S \triangle T$. □

Thus we have shown that the $\{\chi_S\}$ form an orthonormal basis for $\mathbb{G}$. We can therefore any function f in this basis as follows (in what follows we use $[n]$ to denote the set $\{1, 2, \ldots, n\}$):

$$f(x) = \sum_{S \subseteq [n]} \hat{f}(S)\chi_S(x) \ ,$$

where the coefficients $\hat{f}(S)$ w.r.t this basis are called the *Fourier coefficients* of f, and are given by

$$\hat{f}(S) = \langle f, \chi_S \rangle = \frac{1}{2^n} \sum_{x \in \{1,-1\}^n} f(x)\chi_S(x) \ .$$

Fact 4.2 (Fourier Coefficients of a Linear Function)**.**

$$f \textit{ linear} \iff \Big(\exists S \subseteq [n] : \hat{f}(S) = 1\Big) \wedge \Big(\forall T \subseteq [n], T \neq S : \hat{f}(T) = 0\Big)$$

The following describes how the Fourier coefficients are useful in understanding of a function to the various linear functions.

Lemma 4.3. *For every $S \subseteq [n]$,*

$$\hat{f}(S) = 1 - 2\text{dist}\,(f, \chi_S) = 1 - 2 \Pr_{x \in \{1,-1\}^n}\Big[f(x) \neq \chi_S(x)\Big] \ . \tag{5}$$

Proof:

$$\begin{aligned} 2^n \hat{f}(S) &= \sum_x f(x)\chi_S(x) \\ &= \sum_{x: f(x) = \chi_S(x)} 1 + \sum_{x: f(x) \neq \chi_S(x)} (-1) \\ &= 2^n - 2|\{x \mid f(x) \neq \chi_S(x)\}| \\ &= 2^n(1 - 2 \Pr_x\Big[f(x) \neq \chi_S(x)\Big]) \end{aligned}$$

It follows that $\hat{f}(S) = 1 - 2\text{dist}\,(f, \chi_S)$ as claimed. □

In particular, the above implies that any two distinct linear functions differ in exactly $1/2$ of the points. This implies that in the *Hadamard code* which we define below the encodings of two distinct messages differ in exactly a fraction $1/2$ of locations.

Definition 4.4 (Hadamard code)**.** *The Hadamard encoding of a string $a \in \{0,1\}^n$, denoted $\text{Had}(a) \in \{0,1\}^{2^n}$, is defined as follows. Its locations are indexed by strings $x \in \{0,1\}^n$, and $\text{Had}(a)_{|x} = a \cdot x$.*

Parseval's identity

We now state a simple identity that the Fourier coefficients of a Boolean function must obey.

Lemma 4.5. *For any $f : \{-1,1\}^n \to \{-1,1\}$,*

$$\sum_{S \subseteq [n]} \hat{f}(S)^2 = 1 \, .$$

Proof:

$$\langle f, f \rangle = \frac{1}{2^n} \sum_{x \in \{-1,1\}^n} f(x) f(x) = 1 \, .$$

On the other hand

$$\begin{aligned} \langle f, f \rangle &= \langle \sum_{S \subseteq [n]} \hat{f}(S) \chi_S, \sum_{T \subseteq [n]} \hat{f}(T) \chi_T \rangle \\ &= \sum_{S \subseteq [n]} \hat{f}(S)^2 \langle \chi_S, \chi_S \rangle \text{ (since } \langle \chi_S, \chi_T \rangle = 0 \text{ for } S \neq T) \\ &= \sum_{S \subseteq [n]} \hat{f}(S)^2 \, . \quad \square \end{aligned}$$

5 Proof of BLR Soundness

We now set out to prove Theorem 2.4. By Equation (3) we need to analyze the expectation:

$$\begin{aligned} & \mathop{\mathrm{E}}_{x,y} [f(x) f(y) f(xy)] \\ = & \mathop{\mathrm{E}}_{x,y} \left[\left(\sum_S \hat{f}(S) \chi_S(x) \right) \left(\sum_T \hat{f}(T) \chi_T(x) \right) \left(\sum_U \hat{f}(U) \chi_U(xy) \right) \right] \\ = & \mathop{\mathrm{E}}_{x,y} \left[\sum_{S,T,U} \left(\hat{f}(S) \hat{f}(T) \hat{f}(U) \chi_S(x) \chi_T(y) \chi_U(xy) \right) \right] \\ = & \sum_{S,T,U} \hat{f}(S) \hat{f}(T) \hat{f}(U) \mathop{\mathrm{E}}_{x,y} [\chi_S(x) \chi_T(y) \chi_U(xy)] \end{aligned}$$

We claim that the expectation in the last line is 0 unless $S = T = U$. Indeed this expectation

equals

$$\begin{aligned}
& \mathop{\mathrm{E}}_{x,y}\left[\left(\prod_{i\in S} x_i\right)\left(\prod_{j\in T} y_j\right)\left(\prod_{k\in U} x_k\right)\left(\prod_{l\in U} y_l\right)\right] \\
= & \mathop{\mathrm{E}}_{x,y}\left[\left(\prod_{i\in S\triangle U} x_i\right)\left(\prod_{j\in T\triangle U} y_j\right)\right] \\
= & \mathop{\mathrm{E}}_{x}\left[\prod_{i\in S\triangle U} x_i\right]\mathop{\mathrm{E}}_{y}\left[\prod_{j\in T\triangle U} y_j\right]
\end{aligned}$$

If $S \neq U$ or $T \neq U$, then one of the symmetric differences is non-empty, and the expectation is 0, as claimed.

Hence we have the following expression for the desired expectation:

$$\begin{aligned}
\mathop{\mathrm{E}}_{x,y}\left[f(x)f(y)f(xy)\right] &= \sum_{S\subseteq[n]} \hat{f}(S)^3 \\
&\leq \max_S \hat{f}(S) \sum_S \hat{f}(S)^2 \\
&= \max_S \hat{f}(S) \text{ (using Parseval's identity) .} \\
&= 1 - 2\min_S \mathrm{dist}\left(f, \chi_S\right)
\end{aligned}$$

where the last step used Lemma 4.3. Together with (3) we have the following conclusion:

$$\Pr\left[\text{ BLR test rejects}\right] \geq \min_S \mathrm{dist}\left(f, \chi_S\right)$$

This is precisely the claim of Theorem 2.4.

6 Self-Correction

Another tool which will be useful in constructing an assignment tester is a self-correction procedure for the Hadamard code. Assume we have $f : \{0,1\}^n \to \{0,1\}$, a function or table of values, that is δ-close to some linear function L. (We now move back to the $\{0,1\}$ notation; the $\{1,-1\}$ notation was used only for analysing the linearity test.)

Remark 6.1. *If $\delta < \frac{1}{4}$ then such a δ-close L is uniquely determined.*

Using f we would like to compute $L(x)$ for any desired x with high accuracy. (Note that simply probing f at x doesn't suffice since x could be one of the points where f and L differ.) Such a procedure is called a self-correction procedure since a small amount of errors in the table f can be corrected using probes only to f to provide access to a noise-free version of the linear function L.

Consider the following procedure:

Procedure Self-Corr(f, x):

1. Select $y \in \{0,1\}^n$ uniformly at random.

2. Return $f(x+y) - f(y)$

Lemma 6.2. *If f is δ-close to a linear function L for some $\delta < 1/4$, then for any $x \in \{0,1\}^n$ the above procedure Self-Corr(f,x) computes $L(x)$ with probability at least $1-2\delta$.*

Proof: Since y and $x+y$ are both uniformly distributed in $\{0,1\}^n$, we have

$$\Pr_y [f(x+y) \neq L(x+y)] \leq \delta$$

$$\Pr_y [f(y) \neq L(y)] \leq \delta$$

Thus with probability at least $1-2\delta$, $f(x+y) - f(y) = L(x+y) - L(y) = L(x)$, and so Self-Corr$(f,x)$ outputs $L(x)$ correctly. □

7 Constant Query Assignment Tester: Arithmetizing Circuit-SAT

We now have a way to test for linearity and self-correction procedure to gain access to the Hadamard encoding of a string using oracle access to a close-by function. How can we use this for assignment testing? Let's review the assignment testing problem.

Let Φ be a circuit on Boolean variables X and Ψ be a collection of constraints on on $X \cup Y$ where Y are auxiliary Boolean variables produced by an assignment tester. We want the following two properties:

1. if $\Phi(a) = 1$, then $\exists b$ such that $\forall \psi \in \Psi$, $a \cup b$ satisfies ψ

2. If a is δ-far from every a' for which $\Phi a' = 1$ then $\forall b \Pr_{\psi \in \Psi} [(a \cup b) \text{ violates } \psi] = \Omega(\delta)$.

We can reduce any given circuit over Boolean variables (CIRCUIT-SAT) to a set of quadratic equations over $\mathbb{F}_2$ (QFSAT). Then the existence of solutions for the system of equations implies that the original circuit is satisfiable, and vice versa. How do we do this? We have one $\mathbb{F}_2$-valued variable w_i for each gate of the circuit Φ (including each input gate that each has an input variable connected to it). We only need to provide quadratic equations that enforce proper operation of AND and NOT gates.

- $(w_k = w_i \wedge w_j) \rightarrow (w_k - w_i w_j = 0)$
- $(w_l = \neg w_i) \rightarrow (w_i + w_l - 1 = 0)$

If w_N denotes the variable for the output gate, we also add the equation $w_N - 1 = 0$ to enforce that the circuit outputs 1.

It is easy to check that all these constraints can be satisfied iff Φ is satisfiable.

In the next lecture, we'll describe how to use linearity testing and self-correction codes to give an assignment tester for input an arbitrary circuit Φ.

CSE 533: The PCP Theorem and Hardness of Approximation **(Autumn 2005)**

Lecture 9: Proof of PCP Theorem - The Final Piece

10/31/2005

Lecturer: Venkat Guruswami *Scribe: Raghavendra Prasad*

1 Overview

The proof of PCP presented so far is complete except for one last piece - The Assignment Tester (AT). In today's lecture, we use the techniques developed in the previous class to design an assignment tester. With this, we complete the proof of the PCP theorem. We will also take a step back to reflect upon the overall structure of the proof, and the tools it used.

As this lecture relies heavily on the techniques of previous class, we recall a few definitions and results here.

Definition 1.1. *An q-query assignment tester AT $(\gamma > 0, \Sigma_0)$ is a reduction algorithm P whose input is a Boolean Circuit Φ over boolean variables X and output is a system of constraints Ψ over X and a set Y of auxilary variables, such that*

- *Variables in Y take values in Σ_0.*
- *Each $\psi \in \Psi$ depends on at most q variables in $X \cup Y$.*
- *For any assignment a to variables X we have*
  - COMPLETENESS: *If $\Phi(a) = 1$ then there exists an assignment b to variables in Y, such that $a \cup b$ satisfy every constraint $\psi \in \Psi$.*
  - SOUNDNESS: *If assignment $a : X \to \{0,1\}$ is δ-far from every satisfying assignment to Φ, then for any $\forall b : Y \to \Sigma_0$ at least $\gamma\delta_0$ fraction $\psi \in Psi$ are violated by $a \cup b$.*

Remark: The soundness condition is implied by the following statement: $\exists \delta_0 > 0$ and $\gamma_0 > 0$ such that for $\delta \leq \delta_0$, if assignment $a : X \to \{0,1\}$ is δ-far from every satisfying assignment to Φ, then for any $\forall b : Y \to \Sigma_0$ at least $\gamma_0\delta_0$ fraction $\psi \in Psi$ are violated by $a \cup b$. Indeed this implies the above definition with $\gamma = \gamma_0\delta_0$. We will use this form to establish the soundness.

1.1 Arithmetizing a Circuit

Any circuit over boolean variables can be reduced to a system of quadratic equations over $\mathbb{F}_2$, such that there exists a satisfying assignment to the circuit, iff there is a solution to the system of quadratic equations. This reduction from circuits to polynomial equations known as Arithmetization of Circuits, is a recurring technique in complexity theory.

Let us assume we have a boolean circuit C with input variables X and gates Y. Arithmetization of C in to quadratic constraints can be done as follows

- Introduce a variable $z_i \in A$ for each input variable X and for each gate Y. (i.e., $|A| = |X| + |Y|$).
- For each gate $y_i \in Y$ introduce a quadratic constraint P_i, to model the relation between input and output.
  - An AND gate with input z_i, z_j and output $z_k : z_i \cdot z_j - a_k = 0$.
  - An OR gate with input z_i, z_j and output $z_k : z_i + z_j + z_i \cdot z_j - z_k = 0$.
  - A NOT gate with input z_i and output $z_j : z_j + z_i - 1 = 0$.
- Introduce a constraint to force the output of the circuit to $1 : z_k - 1 = 0$ where z_k is the variable corresponding to the output gate of the circuit.

1.2 Assignment Tester

Given a boolean circuit Φ over variables X with gates Y, we first arithmetize Φ to obtain a system of quadratic equations $\mathcal{P} = \{P_1 \dots P_m\}$ over variables $A = \{z_1 \dots z_N\}$ where $N = |X| + |Y|$. Therefore the task of an assignment tester is reduced to checking a solution for a system of quadratic equations, using very few queries.

2 Testing a Quadratic System with Few Queries

The naive method is to check if an assignment $a : A \to \{0, 1\}$ is a solution to a system of quadratic equations $\mathcal{P} = \{P_1, P_2 \dots P_m\}$, is to substitute $a_j = a(z_j)$ for various j in each P_i and check if $P_i = 0$. But observe that in order to do this, we need to know the entire assignment a, which would require a large number (not constant) of queries. Therefore, the first objective is to reduce the number of queries in this naive test.

Instead of checking that $P_i(a) = 0$ for each i, let us take a random linear combination of the numbers $P_i(a)$ and check if it is zero.

$$\sum_{i=1}^{m} r_i P_i(a) \quad = \quad 0$$

where $r_i \in \mathbb{F}_2 = \{0, 1\}$ are chosen at random independent of each other.

Observe that the addition in the above linear combination is over $\mathbb{F}_2$. So it is possible that the linear combination sums up to zero, although each of the terms are non-zero. But we show that if the P_i are indeed non-zero, then with probability $1/2$, their linear combination is non-zero. Therefore this test alone has a completeness of 1 and soundness $\frac{1}{2}$.

Lemma 2.1. *If not all $P_i(a), i = 1 \dots m$ are zero, then*

$$Pr[\sum_{i=1}^{m} r_i P_i(a) \neq 0] = \frac{1}{2}$$

Proof: It is known that there exists an i such that $P_i(a) = 1$. Without loss of generality, we assume $P_m(a) = 1$. Then

$$\sum_{i=1}^{m} r_i P_i(a) = \sum_{i=1}^{m-1} r_i P_i(a) + r_m P_m(a) \tag{1}$$

Observe that irrespective of the value of $\sum_{i=1}^{m-1} r_i P_i(a)$, the entire sum takes values $\{0, 1\}$ for the two choices of r_m. Therefore, in one of the two choices of r_m the sum $\sum_{i=1}^{m} r_i P_i(a)$ is not zero. Hence the result follows. □

So instead of testing $P_i(a) = 0$ for every i, we test that $P_{\vec{r}}(a) = 0$ for a random vector $\vec{r}$. However we still have a problem of computing $P_{\vec{r}}(a)$ without looking at the entire assignment a. For a vector $\vec{r} = \{r_1, \dots, r_m\}$ let us express $P_{\vec{r}}(a)$ as follows:

$$P_{\vec{r}}(a) = \sum_{i=1}^{m} r_i P_i(a) = s_0 + \sum_{i=1}^{N} s_i a_i + \sum_{1 \leq i,j \leq N} t_{ij} a_i a_j \tag{2}$$

for some $s_0, s_1, \dots, s_m \in \mathbb{F}_2$ and $t_{ij} \in \mathbb{F}_2$ for $1 \leq i, j \leq m$. Observe that we only require to check that $P_{\vec{r}(a)} = 0$. Therefore, instead of reading the entire assignment a and computing $P_{\vec{r}(a)}$, we let the prover perform most of the computation, and read only the final results. Specifically, we will ask the prover for each of the sums $\sum_i s_i a_i$ and $\sum_{ij} t_{ij} a_i a_j$. Towards this, let us define the following notation.

Definition 2.2. *Given an assignment $a = [a_1, \dots, a_N]$, define*

$$\begin{aligned} L(s) &= \sum_{i=1}^{N} a_i s_i && \text{for a vector } s \in \{0,1\}^N, \\ Q(t) &= \sum_{i=1}^{N} a_i a_j t_{ij} && \text{for a } N \times N \text{ matrix } t \text{ over } \{0,1\} \end{aligned} \tag{3}$$

Note that for every assignment a, L and Q are linear functions over $\mathbb{F}_2$.

The table of values L is just the Hadamard codeword for the assignment a. Recall the definition of the Hadamard code from last lecture.

Definition 2.3. *For a vector $x = \{x_1, \dots, x_r\}$ over a field $\mathbb{F}$, the Hadamard encoding of x, denoted $\mathsf{Had}(x)$ is given by: $\mathsf{Had}(x)(s) = \sum_{i=1}^{r} s_i x_i$ for every $s \in \mathbb{F}^r$.*

The Hadamard code is the longest possible linear code which does not have any repeated symbols, as it contains all possible linear combinations of the symbols of the message x.

The table of values Q defined above is the *Quadratic Function* encoding of the assignment a. We define this encoding formally below.

Definition 2.4. *For a vector $x = \{x_1, \ldots, x_r\}$ over a field $\mathbb{F}$, the quadratic function encoding of x, denoted $\mathsf{QF}(x)$, is given by $\mathsf{QF}(x)(t) = \sum_{1 \le i,j \le r} t_{ij} x_i x_j$ for all $t \in \mathbb{F}^{r \times r}$.*

In other words, quadratic function code for a vector x is the Hadamard code for the vector $x \otimes x$. So every Quadratic function codeword is a Hadamard codeword (or in other words a linear function) but not vice-versa.

Using this new notation, we re-write equation 2 as

$$P_{\vec{r}}(a) \quad = \quad s_0 + L(s) + Q(t)$$

We can directly obtain the values $L(s)$ and $Q(t)$ from the prover, and check if $P_{\vec{r}}(a) = 0$. Observe that the values of s and t depend on the random choice $\vec{r}$. Let us assume that the proof contains tables L and Q, that list the values of $L(s)$ and $Q(t)$ for all values of s and t. Hence, we can get the value for $L(s)$ and $Q(t)$ directly from the proof to check if $P_{\vec{r}}(a) = 0$. By this we have reduced the number of queries to just two $(L(s), Q(t))$. However, there is a catch — there is no guarantee that the tables L and Q are correct.

By the correctness of L and Q, we mean the following

- $\mathcal{C}_1$**:** L is a linear function, say L is the Hadamard encoding of some $c \in \mathbb{F}_2^N$.
- $\mathcal{C}_2$**:** Q is a linear function on $\mathbb{F}_2^{N^2}$, say it is the Hadamard encoding of some $C = (C_{ij})_{1 \le i,j \le N}$.
- $\mathcal{C}_3$**:** Q and L are referring the same assignment c, i.e., the coefficient C_{ij} in Q is $c_i c_j$. Note that this condition also implies that Q is a Quadratic Function codeword and not just a Hadamard codeword.
- $\mathcal{C}_4$**:** The assignment c that both Q and L are referring to is indeed the assignment a.

From the previous lecture, we know how to test conditions $\mathcal{C}_1$ and $\mathcal{C}_2$ using only a few queries. Assuming that the tables have passed the linearity tests, the two tables are close to linear functions with constant probability. Hence we can use the Self-Correction Lemma of the previous lecture, to obtain the correct value of the linear function at s instead of reading $L(s)$ directly. Let us denote by $\mathit{SelfCorrect}(L, s)$ output of the Self-Correction routine, i.e the value of the linear function at s. If L and Q, pass the linearity test, then we have at our disposal the two linear functions (Hadamard codewords) $\mathit{SelfCorrect}(L, s)$ and $\mathit{SelfCorrect}(Q, t)$.

Testing Condition $\mathcal{C}_3$

Given any two vectors $s, \acute{s} \in \mathbb{F}^N$, if $L = \mathsf{Had}(c)$ and $Q = \mathsf{QF}(c)$, then observe that

$$
\begin{aligned}
L(s)L(\acute{s}) &= (\sum_i a_i s_i)(\sum_j a_j \acute{s}_j) \\
&= \sum_i \sum_j (s_i \acute{s}_j) a_i a_j \\
&= Q(s \otimes \acute{s}) \qquad (4)
\end{aligned}
$$

Therefore, in order to test that L and Q are referring to the same assignment, we can check if $L(s)L(\acute{s}) = Q(s \otimes \acute{s})$ for randomly chosen $s, \acute{s}$.

Assuming Q and L satisfy the test for condition $\mathcal{C}_3$, with constant probability it must be true that Q and L are referring to the same assignment . Therefore, to test $\mathcal{C}_4$, it is enough to test if L is the linear function corresponding to a. i.e the coefficients of s_i in $L(s)$ is a_i. Observe that for e_i- the i^{th} unit vector, we get $L(e_i) = a_i$. So inorder to test if L is referring to assignment a, we just check if $L(e_i) = a_i$ for a randomly chosen i (using $SelfCorrect(L, e_i)$ to computer $L(e_i)$ reliably).

Thus we have few query tests for all the conditions $\mathcal{C}_i, i = 1 \ldots 4$, and also a few query test for $P_{\vec{r}}(a) = 0$. We will put this all together in the next section to get an Assignment tester.

3 Assignment Tester

Input: A boolean circuit Φ over variables X and gates Y such that $|X| + |Y| = N$.

Initialization: Arithmetize the circuit to obtain a system of quadratic constraints $\mathcal{P} = \{P_1, \ldots, P_m\}$ over variables $A = \{z_1, \ldots, z_N\}$. Let variables $z_1 \ldots z_{|X|}$ correspond to variables X and variables $z_{|X|+1} \ldots z_N$ correspond to gates Y.

The Proof:

- An assignment $a = (a_1, a_2, \ldots, a_N) \in \{0,1\}^N$ for the variables A (supposedly a satisfying assignment for $\mathcal{P}$).
- A table $L : \mathbb{F}_2^N \to \mathbb{F}_2$, supposedly equal to $\mathsf{Had}(a)$, i.e., satisfying $L(s) = \sum_{i=1}^N a_i s_i$
- A table $Q : \mathbb{F}_2^{N^2} \to \mathbb{F}_2$, supposedly equal to $\mathsf{QF}(a)$, i.e., satisfying $Q(t) = \sum_{i=1}^N a_i a_j t_{ij}$.

The Test:
Step 1

- Run BLR linearity test on L
- Run BLR linearity test on Q

Step 2 Pick random $s, \acute{s} \in \mathbb{F}_2^N$ and check if the following holds

$$\mathit{SelfCorrect}(L, s)\mathit{SelfCorrect}(L, \acute{s}) = \mathit{SelfCorrect}(Q, s \otimes \acute{s})$$

Step 3 Pick a random vector $\vec{r} \in \mathbb{F}_2^m$. Compute the coefficients s_i and t_{ij} such that

$$P_{\vec{r}}(a) = \sum_{i=1}^{m} r_i P_i(a) = s_0 + \sum_{i=1}^{N} s_i a_i + \sum_{1 \le i,j \le N} t_{ij} a_i a_j$$

Check if

$$s_0 + \mathit{SelfCorrect}(L, s) + \mathit{SelfCorrect}(Q, T) = 0$$

Step 4 Pick a random $i \in \{1, \ldots, |X|\}$. Let e_i denote the i^{th} unit vector of dimension N. Check if

$$\mathit{SelfCorrect}(L, e_i) = a_i$$

4 Proof of Soundness

In order to prove the soundness of the assignment tester, we prove the soundness of each of the steps in it by a sequence of lemmas.

Lemma 4.1. *If L or Q is δ_1-far from a linear function then Step 1 will reject with probability greater than or equal to δ_1.*

Proof: This lemma is nothing but the soundness result for BLR test, which was shown in the previous class. □

Lemma 4.2. *Given a non-zero matrix M, for random choice of vectors s and $\acute{s}$, $s^T M s = 0$ with probability at most $\frac{3}{4}$*

Proof: Let $(M)_{ij}$ be a non-zero entry in M. Let e_i and e_j denote the i^{th} and j^{th} unit vectors. Observe that

$$\begin{aligned} s^T M \acute{s} + (s^T + e_i) M \acute{s} + s^T M (\acute{s} + e_j) + (s^T + e_i) M (\acute{s} + e_j) &= e_i M e_j \\ &= (M)_{ij} \end{aligned}$$

Since, $(M)_{ij}$ is non-zero, it follows that at least one of the numbers $s^T M \acute{s}, (s^T + e_i) M \acute{s}, s^T M (\acute{s} + e_j), (s^T + e_i) M (\acute{s} + e_j)$ is non-zero. This implies that with probability at least $\frac{1}{4}$ over random choice of s and $\acute{s}$, $s^T M \acute{s}$ is not zero, which implies the result. □

Lemma 4.3. ***Completeness:*** *If $L = \mathsf{Had}(c)$, and $Q = \mathsf{QF}(c)$ for some vector $c \in \mathbb{F}_2^N$, then Step 2 always accepts.*
Soundness: *If L is δ_1-close to $\mathsf{Had}(c)$ and Q is δ_1-close to $\mathsf{Had}(C)$ such that $C_{ij} \neq c_i c_j$ for some i, j, then Step 2 rejects with probability at least $\frac{1}{4} - 6\delta_1$*

Proof: In Step 2 we are checking for randomly chosen $s, \acute{s}$, that $L(s)L(\acute{s}) = Q(s \otimes \acute{s})$. This identity holds whenever L and Q are Hadamard and quadratic function encodings of the same vector c. This proves the completeness part of the lemma.

For the soundness proof, define two matrices M_1 and M_2 as follows

$$\begin{aligned} (M_1)_{ij} &= c_i c_j & M_1 &= cc^T \\ (M_2)_{ij} &= C_{ij} & M_2 &= C \end{aligned}$$

From the property of self correction discussed in the previous lecture, Observe that if L is δ_1-close to $\mathsf{Had}(c)$, then with probability greater than $1 - 2\delta_1$,

$$SelfCorrect(L, s) = s^T c$$

That is with the value of $SelfCorrect(L, s)$ can be different from the linear function's value with probability at most $2\delta_1$. Likewise, except with probability at most $2\delta_1$, we have $SelfCorrect(L, \acute{s}) = \acute{s}^T c$. Similarly with probability at least $1 - 2\delta_1$, we know that

$$SelfCorrect(Q, s \otimes \acute{s}) = s^T C \acute{s}$$

Therefore with probability at least $1 - 6\delta_1$ the equality being tested in Step 2 is

$$\begin{aligned} s^T M_1 \acute{s} &= s^T M_2 \acute{s} \\ s^T (M_1 - M_2) \acute{s} &= 0 \\ s^T M \acute{s} &= 0 \end{aligned}$$

where $M = M_1 - M_2$ is a non-zero matrix. From Lemma 4.2 we conclude that $s^T M \acute{s}$ is nonzero with probability $\frac{1}{4}$. Therefore, with probability at least $(1 - 6\delta_1) - \frac{3}{4}$, the test made is $s^T M \acute{s} = 0$ and the test fails. So Step 2 rejects with probability at least $\frac{1}{4} - 6\delta_2$. □

Lemma 4.4. *If L is δ_1-close to $\mathsf{Had}(c)$ and Q is δ_1-close to $\mathsf{QF}(c)$, and for some j, $P_j(c) \neq 0$ then Step 3 rejects with probability at least $\frac{1}{2} - 4\delta$.*

Proof: By Self-Correction we know that with probability at most $2\delta_1$, the value $SelfCorrect(L, s) \neq \sum_{i=1}^{N} c_i s_i$. Similarly with probability at most $2\delta_1$, $SelfCorrect(Q, T) \neq \sum_{i=1}^{N} c_i c_j t_{ij}$. So with probability at least $1 - 4\delta_1$ the test

$$s_0 + SelfCorrect(L, s) + SelfCorrect(Q, T) = 0$$

is equivalent to

$$P_{\vec{r}}(a) = s_0 + \sum_{i=1}^{N} s_i c_i + \sum_{1 \leq i,j \leq N} t_{ij} c_i c_j = 0$$

From Lemma 2.1 we know that for random $\vec{r}$, $P_{\vec{r}}(c) = 0$ with probability $1/2$ (since there exists j such that $P_j(c) \neq 0$). Therefore, with probability at least $1 - 6\delta_1 - \frac{1}{2}$,

$$s_0 + SelfCorrect(L, s) + SelfCorrect(Q, T) \neq 0$$

and Step 3 rejects. □

Theorem 4.5. *Perform each of the steps 1,2,3,4 with probability $\frac{1}{4}$ each. Let a_X denote the assignment a restricted to variables X of the original boolean circuit Φ. Then*

- *If $L = \mathsf{Had}(a)$ and $Q = \mathsf{QF}(a)$ and $P_j(a) = 0$ for all $1 \leq j \leq m$ then the test accepts with probability 1.*
- *Let $\delta \leq 1/28$. If the assignment a_X is δ-far from every satisfying assignment to boolean circuit Φ, then the test rejects with probability at least $\frac{\delta}{8}$ irrespective of the contents of tables L and Q.*

Proof: The completeness part of the proof is trivial, since each of the steps in the assignment tester, has completeness 1.

Suppose L or Q is δ-far from the nearest Hadamard codeword, then from Lemma 4.1 with $\delta_1 = \delta$, Step 1 rejects with probability at least δ. Since with probability $\frac{1}{4}$ Step 1 is performed, the test rejects with probability at least $\frac{\delta}{4}$. Without loss of generality we assume that L and Q are δ-close to their nearest Hadamard codewords.

Further if L and Q do not 'refer' to the same assignment, by applying Lemma 4.3 with $\delta_1 = \delta$, Step 2 rejects with probability at least $\frac{1}{4} - 6\delta$. Observe that for $\delta < \frac{1}{28}$, $\frac{1}{4} - 6\delta \geq \delta$. As Step 2 is chosen with probability $\frac{1}{4}$, the test rejects with probability at least $\frac{\delta}{4}$. So without loss of generality, we can assume that L and Q also refer to the same assignment c.

Suppose c is not a satisfying assignment for $\mathcal{P}$, then by applying Lemma 4.4 with $\delta_1 = \delta$, we know that Step 3 rejects with probability at least $\frac{1}{2} - 4\delta$. Since $\delta < \frac{1}{28}$, Step 3 rejects with probability at least $\frac{1}{2} - 4\delta > \delta$. As Step 3 is chosen with probability at least $\frac{1}{4}$, the test rejects with probability at least $\frac{\delta}{4}$. So without loss of generality we can assume that c is a satisfying assignment for $\mathcal{P}$.

We know that a_X is δ-far from every satisfying assignment to Φ, so in particular it is δ-far from c_X (since c being a satisfying assignment for $\mathcal{P}$ implies that c_X satisfies the circuit Φ). By the property of Self Correction, and that L is δ-close to $\mathsf{Had}(c)$, we know that $SelfCorrect(L, e_i) = c_i$ with probability at least $1 - 2\delta$. Therefore with probability at least $1 - 2\delta$, Step 4 is testing if $a_i = c_i$. Since a is at least δ-far from c, $a_i \neq c_i$ with probability δ over the random choice of i in $\{1, \ldots, |X|\}$. Therefore Step 4 rejects with probability at least $\delta(1 - 2\delta) \geq \frac{\delta}{2}$. As Step 4 is chosen with probability at least $\frac{1}{4}$, the test rejects with probability at least $\frac{\delta}{8}$.

□

4.1 Remarks

- Observe that by end of Step 3 in the assignment tester, the verfier is convinced that there is some satisfying assignment to $\mathcal{P}$ and L and Q are referring to it. Therefore, steps $1, 2, 3$ already form a PCP system albeit with exponential size proofs. Step 4 just tests if the given assignment a is close to the assignment, that L and Q are referring to. This is the extension to get the Assignment Tester property.

- We have presented the assignment tester in a Prover-Verifier description. This description can be readily converted to suit the original definition of assignment tester in terms of constraints. The auxilary variables of the assignment tester, are

  - The variables a_i corresponding to the gates in the original boolean circuit Φ.
  - The entries in tables L and Q.

  Thus there are $m + 2^N + 2^{N^2}$ auxilary variables in total, all over the alphabet $\mathbb{F}_2$. All the constraints have at most six variables each, and are linear or quadratic equations over $\mathbb{F}_2$.

5 PCP Proof - Another look

With the construction of a Assignment Tester, we have completed the proof of the PCP theorem. Considering the importance of the theorem, and the variety of tools used in its proof, it is worth taking another look at the proof.

PCP theorem as it is stated normally, implies that for every language in NP there is a polynomial-size proof, that can be checked by random verifier using very few queries. However, as we showed in the first class, the PCP theorem can also be stated as a NP-hardness result. Towards this, we observed that PCP theorem is equivalent to the fact that $GAP - CG_{1,s}$ is NP-hard for some constant $s < 1$.

In order to show that $GAP - CG_{1,s}$ is NP-hard, we need a polynomial time reduction from a known NP-complete problem to it. But we also know that Constraint Graph satisfaction is NP-hard. Therefore, a possible proof of the PCP theorem would be to reduce a Constraint graph satisfaction problem to $GAP - CG_{1,s}$. Observe that Constraint Graph satisfaction problem is a special case of $GAP - CG_{1,s}$ for $s = 1 - \frac{1}{|E|}$ where $|E|$ is the number of constraints in the constraint graph. Let us define the "gap" to be the unsatisfiability value of a constraint graph (i.e., 1 minus the fraction of constraints satisfied by the best assignment). So the PCP theorem guarantees a reduction from instances with gap $\frac{1}{|E|}$ to instances with gap $1 - s = \Omega(1)$. Therefore PCP theorem can be viewed as a Gap Producing or Amplifying Reduction.

The proof that we presented, creates the gap slowly and persistently, a little increase each time, keeping the other parameters under control. In the original proof of the PCP theorem, a large gap was created in one-single step and the other parameters had to be remedied later.

The Gap producing reduction consisted of several iterations. In each iteration, there were four steps that modified the parameters of the constraint graph appropriately. At the end of each iteration, the gap of the graph doubled, with a accompanying constant factor increase in its size. Therefore, by the end of $O(\log n)$ iterations, the gap increases to a constant, while the size is still polynomial in the original size of the graph.

The following table , sums up the central ideas and tools used in each of the four steps.

Step	Main Ideas	Effects	Proof Techniques
Degree Reduce	Split every vertex in to many vertices,and introduce an Expander cloud with equality constraints among the split vertices.	Size $\uparrow$ a $O(d)$ factor, Gap decreases by a constant factor, Alphabet remains same	Basic expansion property of expanders
Expanderize	Superimpose a constant degree expander with trivial constraints, on to the constraint graph G	Size $\uparrow$ a factor of 2 to 3, Gap decreases by a constant factor, Alphabet remains same	Existence of constant degree expanders and Property that Expander + Graph gives an expander.
Gap-Amplification	Each vertex's value is its opinion,on the values of vertices at a distance $< t$,Add edges corresponding to consistency on random walks	Size $\uparrow$ by a large constant factor ,Gap increases by $O(t)$, Alphabet size becomes $\|\Sigma\|^{O(d^t)}$	Properties of random walks on the graph
Alphabet-Reduce	Encode the assignment with error correcting codes, Build a circuit that checks if assignment satisfies and is a valid codeword, Use an assignment tester for the circuit	Size $\uparrow$ a constant factor, Gap decreases by a constant factor,Alphabet size reduced to 2^6	Hadamard codes, Linearity Testing, Fourier Analysis

Table 1: Proof of PCP