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

Definition 1.3. $\text{GAP-CG}(\Sigma)_{c,s}$ *(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* $\text{GAP-CG}(\Sigma_0)_{1,s}$ *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-COLOR problem. The 3-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-COLOR, all of the constraints can be simultaneously satisfied, whereas in the NO case of 3-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-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-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:

- $\mathrm{size}' = O(\mathrm{size})$.
- 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:

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

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 $\mathrm{gap} = 0$ in the YES case and $\mathrm{gap} \geq 1/m$ in the NO case. Suppose we run all four subroutines in order on G, producing some new constraint graph G'. What results is that $\mathrm{size}' = O(\mathrm{size})$, 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 $O(1)^{\log m} = \mathrm{poly}(m)$; 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), $\ldots$, 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. $\square$