CS 880: Advanced Complexity Theory 3/5/2008

Lecture 17: Hardness of Approximation

Instructor: Dieter van Melkebeek Scribe: David Malec

In this and the next few lectures we revisit the PCP Theorem with the aim of establishing tight inapproximability results for some natural NP-hard optimization problems. Earlier, we discussed the use of harmonic analysis in the derivation of the PCP Theorem. More specifically, in lecture 5 we devised tests for the Hadamard code and the long code, which play a critical role in the construction of PCPs. In order to obtain the tight inapproximability results we aim for, we will need to delve again into the harmonic analysis underlying the PCP Theorem.

1 Inapproximability Results

In order to demonstrate inapproximability results, we can view a PCP as a game, where the prover is trying to maximize the probability that its purported proof is accepted. The idea then is to reduce the optimization of this game to natural optimization problems. As long as our reductions preserve the gap between the completeness and soundness conditions of the PCP, we can conclude that surpassing some approximation threshold is as hard as distinguishing between the two cases of the PCP Theorem, which is NP-hard.

The next few lectures develop tight inapproximability results for several problems along these lines. In some cases we need the PCP games to have some more struture for the reduction to work, namely that of so-called unique games. As a result, we can only conclude that certain approximation factors are UG-hard rather than NP-hard to realize, where UG refers to Unique Games. The problems which we consider are the following.

MAX-3-SAT: Here, we are given a 3-CNF formula ϕ, and want to find an assignment of variables which maximizes the number of simultaneously satisfied clauses in ϕ. If we know that each clause contains exactly three literals, all distinct, then we can find a 7/8-approximation by making a random assignment to the variables; the constant follows since only one of the eight possible assignments to any given clause fails to satisfy it, and our assumption that all literals are distinct ensures that independent assignments to the variables make each of these outcomes equally likely. The algorithm can easily be derandomized. If we don't have exactly three distinct literals in each clause, we can still effect this result, but it requires more machinery. Despite the simplicity of achieving this 7/8-approximation, we show that finding a $(7/8 + \epsilon)$-approximation to MAX-3-SAT is NP-Hard, even when all the clauses contain exactly three distinct literals.

MAX-3-LIN: In this problem, we are given a linear system of equations over GF(2), where each equation contains three variables, and we are asked to find a setting of the variables maximizing the number of satisfied equations. Using random assignment gives us a 1/2-approximation, since each nontrivial linear equation over GF(2) is equally likely to be satisfied or not under a random assignment. Once again, this type of algorithm yields the best possible approximation in some sense, since achieving a factor of $(1/2 + \epsilon)$ is NP-hard.

MAX-CUT: This problem asks for the maximum cut of agraph, where the value of a cut is the sum of the weights of the edges crossing it. While not simple, an algorithm exists which yields an ρ-approximation for $\rho \approx .878$. In fact, finding a $(\rho + \epsilon)$-approximation is UG-hard for any constant $\epsilon > 0$. While the constant ρ in this result might seem surprising, there is a geometric interpretation which gives some intuition as to how it arises; another way of interpreting it comes from its relation to the noise sensitivity of Majority.

VERTEX-COVER: This problem asks us to find a minimal vertex cover for a graph. A trivial algorithm exists which gives a 2-approximation; finding a $(2 - \epsilon)$-approximation, however, is UG-Hard.

Note that the inapproximability results we give for MAX-CUT and VERTEX-COVER have stronger conditions, namely that the so-called Unique Games Conjecture holds rather than just that P $\neq$ NP.

2 PCPs and Constraint Graph Games

In order to derive our inapproximability results we use a reduction from PCPs to constraint graph games.

2.1 Basic Definitions

Before addressing the reduction itself, we first review the PCP Theorem, and present the definition of a constraint graph game (CGG).

Theorem 1 (PCP Theorem)**.** *There exists a polynomial-time function f : 3-CNF $\rightarrow$ 3-CNF, and a constant $0 < \rho < 1$ such that for all $\phi \in$ 3-CNF:*

$$\phi \in SAT \Rightarrow MAX\text{-}SAT(f(\phi)) = 1; \text{ and}$$
$$\phi \notin SAT \Rightarrow MAX\text{-}SAT(f(\phi)) \leq \rho,$$

where we use MAX-SAT$(f(\phi))$ to indicate the maximum percentage of clauses in $f(\phi)$ which may be simultaneously satisfied.

We do not go into details of the PCP theorem here; we simply apply the result. We need one more component before we can proceed with our reduction – specifically, we need the notion of a constraint graph game, which we now define.

Definition 1. *A constraint graph game G is specified as $G = (L, R, E, [l], [r], C)$, where: $L \cup R$ form the nodes of a bipartite graph with edges $E \subseteq L \times R$; $[l]$ and $[r]$ are sets of labels that may be applied to the nodes in L and R, respectively; and*

$$C : E \rightarrow \{c : [l] \times [r] \rightarrow \{0, 1\}\}$$

maps each edge $e = (u, v)$ to a constraint c_e on the labels of u and v.

Given a constraint graph game G, we use $\nu(G)$ to denote the maximum fractions of the constraints c_e on G that may be simultaneously satisfied by assignments of labels from $[l]$ to the vertices in L, and labels from $[r]$ to the vertices in R. There are two special types of constraint graph games that prove to be of particular interest to us. Their definitions are as follows.

Function-Type: The constraints can be derived from a function on the labels $[l]$; that is to say,

$$\exists \pi_e : [l] \to [r] \text{ such that } c_e(i,j) = 1 \Leftrightarrow j = \pi_e(i), \tag{1}$$

where we use the notation π_e to indicate that we view this as a projection.

Unique-Type: This is a further specialization of function-type constraint graph games; the condition here is the same as (1), with the exception that we further require that π_e be a permutation. In particular, this requires $l = r$.

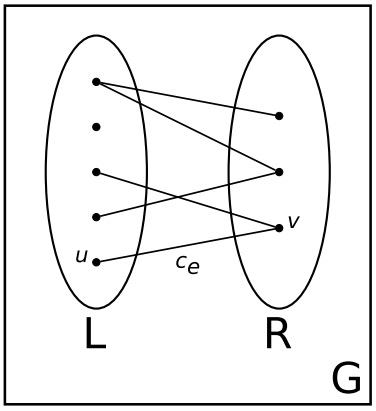

Figure 1: Example of a constraint graph game. Here, c_e enforces the constraint on the edge $e = (u, v)$, and u and v have labels drawn from $[l]$ and $[r]$, respectively.

2.2 Reduction from PCPs to CGGs

Our basic procedure for obtaining inapproximability results is to reduce a PCP to a constraint graph game G; one critical point is that our reduction must be gap-preserving.

Given an f and ρ as described in Theorem 1, and any 3-CNF ϕ, we construct a constraint graph game $G = (L, R, E, [l], [r], C)$ as follows. We form the nodes and edges

$$\begin{aligned} L &= \{\text{Clauses of } f(\phi)\}, \\ R &= \{\text{Variables of } f(\phi)\}, \text{ and} \\ E &= \{(u, v) \in L \times R : \text{ clause } u \text{ contains the variable } v\}. \end{aligned}$$

Our labels for L represent all possible assignments to the variables of the corresponding clause that satisfy that clause. Since a clause with at most 3 literals can have at most 7 distinct satisfying assignments to the variables involved, this means we can set $l = 7$. Note that there are no labels that correspond to assignments that falsify the clause. The labels for R represent the possible truth values of the corresponding variable, so we set $r = 2$. Lastly, for each edge $e = (u, v)$, the constraint c_e represents a check for consistency between the values assigned to u and v, namely that the satisfying assignment determined by the label of u sets the variable v according to v's label. Note that the constraint graph game we have constructed is of function-type, as the label of u dictates the assignment to each of the variables in the corresponding clause, and therefore the label of each v connected to u.

Under this reduction, we get that

$$\phi \in SAT \Rightarrow \nu(G) = 1 \tag{2}$$
$$\phi \notin SAT \Rightarrow \nu(G) \leq s, \tag{3}$$

where $s = \rho + (1 - \rho) \cdot 2/3$ is a constant less than 1.

We can see that (2) follows because a satisfying assignment to $f(\phi)$ induces a labeling G that satisfies all constraints. Condition (3) follows since any assignment to the variables of $f(\phi)$ has to violate at least a fraction $1 - \rho$) of $f(\phi)$. This means that for every labeling of R there has to be a fraction at least $1 - \rho$ of L whose label (which always satisfies the corresponding clause) is inconsistent with at least one of the at most three variables it is connected to.

2.3 Strong PCP Theorem

In order to obtain our tight inapproximability results, we need to boost the soundness level in (3). A natural approach is to repeat the game for multiple trials. If we wish to preserve our overall game as being a constraint graph game, however, we cannot perform sequential trials. Instead, we perform repeated trials in parallel, i.e., both players get the inputs for all trials at once. While it is not obvious that this works as a method for boosting soundness, it does. This is the contents of the Parallel Repetition Theorem.

Theorem 2 (Parallel Repetition Theorem)**.** *For any $s < 1$, l, and r, there exists $s' < 1$ such that for each integer $k \geq 1$,*

$$\nu(G^k) \leq (s')^k,$$

where we use G^k to denote the k-fold parallel repetition of G.

Now, since G^k is the k-fold parallel repetition of G, our label sets for G^k are $[l^k]$ and $[r^k]$. If we want the soundness level to be γ or better, we need that

$$(s')^k \leq \gamma,$$

or equivalently that

$$k \geq \log_{s'} \left(\frac{1}{\gamma} \right). \tag{4}$$

We can see that satisfying (4) tightly gives us the bound

$$l^k, r^k \leq \frac{1}{\text{poly}(\gamma)}. \tag{5}$$

One point of note is that Theorem 2 maintains the function-type property on constraint graph games; so our game retains this property. Combining Theorem 2 with Theorem 1 (and the bound (5)) allows us to make the following stronger formulation of the PCP Theorem, which incorporates our reduction.

Theorem 3 (Strong PCP Theorem)**.** *For any constant $\gamma > 0$, there exist $l, r \leq \frac{1}{\text{poly}(\gamma)}$ such that there is a reduction $f : 3\text{-}CNF \rightarrow CGG$ using that l, r such that for all ϕ:*

$$\phi \in SAT \Rightarrow \nu(f(\phi)) = 1; \text{ and}$$
$$\phi \notin SAT \Rightarrow \nu(f(\phi)) \leq \gamma.$$

3 Reduction to MAX-3-LIN

Throughout the following discussion, we work with a slight variation on the version of MAX-3-LIN previously mentioned; since we want to work over $\{-1, 1\}$ instead of over $\{0, 1\}$, our equations go from having the form $x + y + z = b$, where $b \in \{0, 1\}$, to having the form

$$xyz = \pm 1. \tag{6}$$

In order to reduce a constraint graph game G to a system of equations having the form (6), we look at the long code of the labels which get assigned to the vertices of G, and then test for properties that these encodings must satisfy. If we develop a tester whose acceptance condition can be expressed as an equation of the form (6), then we may generate our system of equations to correspond to all possible executions of our tester. At the end of Lecture 5 we developed a test for any property of a string encoded using the long code. However, in our current situation the property involves two strings u and v, both of which are involved in multiple properties that need to be verified. We could have separate encodings for each $(u, v) \in E$, but then we would need to add consistency checks for joint encodings involving a common u. Instead, we will obviate the need for such consistency checks by encoding the labels for all $u \in L$ and $v \in R$ individually.

The variables in our system represent the bits of the long codes of the labels of all vertices in the graph G. We then generate the system using a tester for long codes and expressing each of the edge conditions as a linear equation. In fact, we'll see that we can do both tests (long code and additional conditions) in one shot.

Consider how we can convert the edge constraints from G into linear equations. Let $e = (u, v)$ be an edge in E. Say we let i and j represent the labels given to vertices u and v, respectively. Assuming f_u and g_v are the correct long encodings of $i \in [l]$ and $j \in [r]$, respectively, we have that

$$f_u(x) = x_i,$$
$$g_v(y) = y_j,$$

since long codes correspond to dictators. Note that $f_u : \{0, 1\}^l \to \{0, 1\}$, and $g_v : \{0, 1\}^r \to \{0, 1\}$.

Now, because the constraint graph game we are working with is of function type, we have that the constraint c_e can be expressed as

$$c_e : \pi_e(i) = j. \tag{7}$$

Recall that $\pi_e : [l] \to [r]$; one way of interpreting this is to think of π_e as associating each index in a string of length l with an index in a string of length r. So, if we have some $y \in \{0, 1\}^r$, we can use this to generate an $x \in \{0, 1\}^l$ by setting position i of x to match position $\pi_e(i)$ of y. We denote a string generated in this fashion by $(y \circ \pi_e)$. So we have that

$$(y \circ \pi_e)_i = y_{\pi_e(i)} \qquad \forall i \in [l],$$

and so we may equivalently write c_e as

$$\forall y \in \{-1, 1\}^r, f_u(y \circ \pi_e) = g_v(y). \tag{8}$$

In the next lecture, we will use the last condition as the starting point for developing a gap-preserving reduction from CGG to MAX-3-LIN.