Advanced Approximation Algorithms **(CMU 15-854B, Spring 2008)**

Lecture 21: Hardness of Approximation for CSPs

April 1, 2008

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

1 Overview

In the next few lectures we will try to prove the following two theorems, first proved by Hastad in [Has01].

Theorem 1.1. *The* 1 *vs.* $\frac{7}{8} + \epsilon$ *decision problem for Max-E3SAT is* NP*-hard.*

Theorem 1.2. *The* $1 - \epsilon$ *vs.* $\frac{1}{2} + \epsilon$ *decision problem for Max-E3LIN is* NP*-hard. The following two hardness results hold assuming that* $\mathrm{P} \neq \mathrm{NP}$

Two remarks: first, it is easy to see that Theorem 1.2 implies $1 - \epsilon$ vs. $\frac{7}{8} + \epsilon$ hardness for Max-E3SAT, the extra work is in increasing the completeness parameter to 1. Second, both of these results are the best possible, since for Max-E3SAT we can get a $7/8$ approximation based on a random assignment and for Max-E3LIN we can get a $1/2$-approximation by either the all 0 or all 1 assignment.

This class will be mainly a philosophical class. In particular, recall the hardness proofs of Max-Coverage and Max-Ek-Independent-Set from earlier in the class. They were both based on the boolean hypercube $\{0,1\}^k$, and worked by having a gadget for every vertex or edge such that there were k "correct" ways of setting the gadgets, but any other "good" setting would be a constant worse. Also, in both these cases the gadgets were based off dictator functions. In this class, we try to at least begin to answer the following two questions: why do we always use the boolean hypercube, and why are the good solutions always based on the dictator functions?

To see this, let's start trying to prove hardness for Max-E3SAT.

2 Generic Hardness Reduction

As always, we will start with an instance of Label-Cover. Recall that in Label-Cover(K, L) we are given a bipartite graph (X, Y, E) and functions $\pi_{yx} : L \to K$ for each edge (x, y) in the graph. The goal is to give a labeling $f : X \to K, Y \to L$ that maximizes the number of edges that satisfy $\pi_{yx}(f(y)) = f(x)$. We know from parallel repetition that Label-Cover(K, L) is 1 vs. η hard for all $\eta > 0$, where $|K|, |L|$ are polynomial in $1/\eta$. We normally think of η as a small constant.

A polytime reduction from Label-Cover to Max-E3SAT will output list of M OR functions (each of three literals) over the variables $v_1, \ldots, v_N$, and weights $w_1, \ldots, w_M$ that add up to 1. A hardness proof will follow the following outline. First, a completeness proof will show that if there

is some $f : X \to K, Y \to L$ that satisfies all of the Label-Cover constraints, then there is an assignment to $v_1, \ldots, v_N$ satisfying all of the OR constraints. This is usually the easy part. Then a soundness proof will show that if optimum value of the Label-Cover instance is less than η then the optimum value of the Max-E3SAT instance we reduced to is at most $\frac{7}{8} + \epsilon$.

The basic idea is to have a "block" of variables $v_1, \ldots, v_m$ for each vertex in the graph. This block is supposed to "encode" the key/label of that node. Then we place OR constraints on each block to "check" that that v_i's are properly encoding some key/label. This is where we'd like to put a algorithmic gap instance, where there are $|K|$ ways to encode the keys (one for each key) and any other assignment to the variables is somehow bad. Also, we need to place OR constraints across the blocks to ensure consistency, i.e. to actually force the variable assignment to represent a good solution for the original Label-Cover instance. In this lecture we will not discuss this part, instead focusing on how to encode keys as settings of variables. We will now give high-level descriptions of what the completeness and soundness proofs look like.

Completeness: Suppose that there is some $f : X \to K, Y \to L$ that satisfies every edge in the graph. We want to construct a function encode : $K \to \{0,1\}^m$ (and an analogous encode' for labels) that, given a key, outputs an assignment to the m variables in the block. We want this encoding function to have the property that valid encodings pass all of the OR constraints on each block and valid encodings of consistent key/label pairs pass all of the OR constraints between the blocks. While as stated this allows multiple keys to have the same encoding, we're actually going to want the slightly stronger property that there are $|K|$ distinct assignments that satisfy all of the OR constraints.

Soundness: As always, we will prove soundness by contrapositive. Suppose that $v_1, \ldots, v_N$ have an assignment that satisfies more than $\frac{7}{8}+\epsilon$ of the constraints. Then by a simple averaging argument we know that at least an $\epsilon/2$ fraction of the blocks satisfy at least $\frac{7}{8} + \epsilon/2$ of the constraints on that block. Now we want our gap gadget to have the property that if this many constraints on a block are satisfied, then there is some small list $Sugg(x) \subseteq K$ that we can "decode" the assignment to. We'll also want the (right now very vague) property that if there is an edge between two nodes, then their lists of suggested keys/labels are somehow "consistent".

This problem, of creating such encoding and decoding schemes, can be thought of in three ways: as a hard algorithmic gap instance, as a Label-Cover gadget, or as a locally list-decodable code. This last way, which we have not discussed before, has been developed in the coding theory community. At a very high level, a locally list-decodable code is a code with the property that if we're given some word (not necessarily a codeword) that passes many "local" checks, then we can decode to a short list of codewords. Here the locality of the checks basically means that each check is just an OR of three bits of the word.

3 Encoding Scheme

Since we are reducing to Max-E3SAT, we want our constraints to be ORs of three literals. But we can think of this as developing a testing algorithm that is only allowed to interact with the assignment by picking three literals and taking their OR. In particular, suppose we are given M OR constraints, where the ith constraints has weight w_i. Then an assignment satisfies a fraction α of these weighted constraints if and only if the algorithm that chooses a constraint at random according to the weights and then checks whether that one constraint is satisfied succeeds with probability α. We will generally use this view, of the constraints as a testing algorithm.

Note that we really have two components in the generic reduction described above: an encoding scheme that determines how keys/labels should be encoded as variable assignments, and a testing algorithm/constraint system that is supposed to check whether or not the assignment is a valid encoding (when thought of as a testing algorithm) or force the encoding to be valid (when thought of as a constraint system). The strength of the hardness result that we get is basically a function of how good our constraint system is, so we want to use whatever encoding scheme lets us have the strongest testing algorithm/constraint system. In other words, when designing our encoding scheme we want to make the job of the testing algorithm as easy as possible.

So now we have $|K|$ keys, and we want to map them to certain variable assignments. Clearly a variable assignment can be represented as a binary string, so we want to assign each key to a binary string of length m. We want to make it easy to distinguish these special strings from all other strings. Think of these strings as an array:

$$\begin{array}{cccc} 0 & 1 & \dots & 1 \\ 1 & 0 & \dots & 0 \\ & & \vdots & \\ 1 & 0 & \dots & 1 \end{array}$$

where the ith row is the string representing key i. The key idea here is that adding an extra column can only help. Any testing algorithm could just ignore the new column, so it's clearly no worse, and in fact it might contain enough extra information that it allows the testing algorithm to do better. Since this redundancy can only help, we might as well use the most redundant encoding possible, i.e. put in all possible columns.

$$\begin{array}{ccccccc} 0 & 0 & 0 & 0 & \dots & 1 \\ 0 & 0 & 0 & 0 & \dots & 1 \\ \vdots & \vdots & \vdots & \vdots & \dots & \vdots \\ 0 & 0 & 1 & 1 & \dots & 1 \\ 0 & 1 & 0 & 1 & \dots & 1 \end{array}$$

This encoding is known as the "Long Code", and was first developed by Bellare, Goldreich, and Sudan [BGS98]

There are $2^{|K|}$ columns (variables) in this encoding. We can name each column by the contents of that column, which is some string in $\{0,1\}^{|K|}$. So in the column named x (which is some element of $\{0,1\}^{|K|}$, the bit in the ith row is just x_i. While in this encoding it is easy to see what the columns are, what are the rows, i.e. what are the actual codewords? Note that any binary string of length $2^{|K|}$ is just the truth table of some function $f : \{0,1\}^{|K|} \to \{0,1\}$, so we can identify

an assignment to the variables with such a function. It is easy to see that the codeword for key i, which is just the ith row of this array, is the function $f(x) = x_i$ (the ith dictator function). Since we only helped ourselves by adding columns, without loss of generality we might as well always use this encoding.

Note that by adding all of the columns we made the size of the new instance at least $n2^{|K|}$. Since $|K|$ is polynomial in $1/\eta$ and for our purposes η is a constant, this is fine; $2^{|K|}$ is just another constant. But if we wanted to prove super constant hardness bounds, this starts to become a problem, since then we'll generally want η to be subconstant.

So, finally, we have our general hardness reduction technique. Suppose that we are trying to prove c vs. s hardness for Max-E3SAT. Our task is to design a "Dictator-List-Decoding test" which queries three values of an arbitrary function $f : \{0,1\}^{|K|} \to \{0,1\}$ and checks the OR of these values. If f is a dictator, then this test should succeed with probability at least c. If f is some function that passes the test with probability at least $s + \epsilon$, then f should be "decodable" to a list of small size (dependent only on ϵ). In the next lecture we will give such a test, and analyze it using Fourier analysis.

What if we are trying to prove hardness to Max-E3LIN? Then instead of checking the OR of the three queries, we check the XOR. Similarly, for Max-Cut we can make two queries and check that $f(x) \neq f(y)$.

References

[BGS98] Mihir Bellare, Oded Goldreich, and Madhu Sudan. Free bits, PCPs, and nonapproximability—towards tight results. *SIAM J. Comput.*, 27(3):804–915, 1998.

[Has01] Johan Hastad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, 2001.