601.435/635 Approximation Algorithms **Lecturer:** Michael Dinitz
Topic: More Hardness of Approximation: Label Cover, Parallel Repetition **Date:** 4/25/24
Scribe: Michael Dinitz

26.1 Introduction

Last class we talked about the PCP theorem, the fact that $PCP_{c(n),s(n)}(O(\log n), O(1)) = NP$ for various interesting completeness $c(n)$ and soundness $s(n)$, and how to use this theorem to prove hardness of approximation for various problems (Max-3SAT in particular). Today we're going to continue this and talk about some extensions, most notably a problem called Label Cover which forms the basis of many modern hardness of approximation results. I'm going to present Label Cover differently from the book – they just present it directly, while I'm going to try to present it in a way which has both more historical context and I think shows the connections to complexity theory a bit more (like with the PCP theorem). I'm going to be a little bit informal, but the book has all the important details.

26.2 Two Prover Proof Systems

Last time we proved that Max-3SAT is hard to approximate better than $\frac{7}{8} + \epsilon$ (for arbitrarily small $\epsilon > 0$) by using the version of the theorem which states that $PCP_{1-\epsilon,1/2+\epsilon}(O(\log n), 3) = NP$ and where the PCP verifier only uses even and odd tests. Today we're going to use a slightly different version of the PCP theorem which can also be used to prove the same result, but where the parameters (particularly the completeness) will lead to it being a bit more useful later on.

Theorem 26.2.1 *$PCP_{1,7/8+\epsilon}(O(\log n), 3) = NP$, even when the PCP verifier is only allowed to compute the OR function of the three bits (or their negations).*

Note that this version of the PCP theorem implies the hardness of Max-3SAT directly, since each OR function computed by the PCP verifier can be obviously translated to a single 3CNF clause (unlike last time where we translated each function into four 3CNF clauses). Moreover, this PCP theorem has completeness 1, which will be very useful for us. So, in particular, this PCP theorem gives the following theorem.

Theorem 26.2.2 *Unless $P = NP$, there is no polynomial time algorithm which can distinguish between instances of Max-3SAT where all clauses are satisfiable (YES instances) and instances in which at most $\frac{7}{8} + \epsilon$ of the clauses are satisfiable (NO instances).*

It's not actually necessary, but it's generally easier to think about a special case of 3SAT where we also have an additional bound, on the number of clauses in which any variable appears. Let Max-3SAT-5 be Max-3SAT but only on instances where every variable is in exactly five clauses. There are standard transformations from 3SAT to 3SAT-5 which lose the 7/8 but maintain a (smaller) constant gap

Theorem 26.2.3 *Unless $P = NP$, there is some constant $\epsilon > 0$ such that no polynomial time* *algorithm can distinguish between instances of Max-3SAT-5 where all clauses are satisfiable (YES instances) and instances in which at most $1 - \epsilon$ of the clauses are satisfiable (NO instances).*

Now that we have Theorem 26.2.3, let's take what first might seem like another philosophical detour into another notion of "proof". Last time we briefly mentioned different types of proof systems, like interactive proofs and zero-knowledge proofs, before exploring probabilistically checkable proofs in more depth. Let's consider a new setup for proof systems: "one-round two-prover" proof systems.

Suppose, like last time, you're holding an instance of Max-3SAT and want a proof that it is satisfiable. But instead of just me proving it to you, there are actually *two* provers. These two provers are allowed to talk with each other before we start, but then you're allowed to ask a question of each prover and they have to respond without talking to each other. How can we set up a protocol so that if there is a satisfying assignment you will be convinced, but if the best assignment only satisfies on $7/8 + \epsilon$ of the clauses, you have a reasonable probability of rejecting?

Of course, one obvious thing you could do is ask each prover to just give you an assignment. Then if there is a satisfying assignment, they can give it to you. If there's not, then you can just check whatever they give you and realize that it's bad.

But what if you want everything to be compact, e.g., what if you want each question and response to be very short? Here's a natural protocol:

- Choose a clause C uniformly at random
- Choose one of the three variables x_i in C uniformly at random
- Ask prover 1 the value of x_i, ask prover two which of the seven satisfying assignments for C is the one in the overall solution.
- The answer of prover 2 gives you an assignment to x_i. If it matches with what prover 1 said, then return YES. Otherwise return NO.

This protocol has some nice properties, including good completeness and soundness.

Lemma 26.2.4 *If there is a satisfying assignment, then the provers can get the verifier to return YES with probability* 1.

Proof: This is basically trivial: both provers respond honestly, i.e., prover 1 gives the value of x_i in the satisfying assignment, and prover 2 gives the assignment to variables in clause C in the satisfying assignment. Since it's a satisfying assignment, prover 2's solution satisfies C, and since both provers agreed on the same satisfying assignment, their answers will agree on x_i. Thus the verifier will return YES. ■

Lemma 26.2.5 *If any assignment satisfies at most $1 - \epsilon$ of the clauses, then no matter what the provers do, the probability that the verifier returns YES is at most $1 - \epsilon/3$.*

Proof: Since each prover is deterministic and cannot communicate with each other after the questions are asked, prover 1 always returns the same value whenever it is queried on the same variable x_i. So prover 1 essentially has an assignment of values to variables, which by assumption can only satisfy at most $1 - \epsilon$ of the clauses. So with probability ϵ, the clause that we ask prover

2 is one of the clauses that this assignment does not satisfy. Since prover 2 has to return one of the 7 assignments to C which satisfy it, there must be at least one variable that has a different assignment in prover 2's answer than in what prover 1 would answer. Since we choose which of the three variables in C to ask prover 1 uniformly at random, the total probability of catching prover 1 and prover 2 in an inconsistency (and thus returning NO) is at least $\frac{1}{3}\epsilon$. ■

26.3 Label Cover

Last lecture we transformed a "weird" proof system into a relatively natural problem (a CSP). Now we're going to do the same: we're going to transform this two-prover proof system into a problem known as Label Cover, which is also a CSP, but of a particular kind. Label Cover is defined as follows:

- Input: Bipartite graph $G = (L, R, E)$, alphabet Σ_L, alphabet Σ_R, relation $\pi_e \subseteq \Sigma_L \times \Sigma_R$ for each $e \in E$
- Feasible: assignment $f : L \to \Sigma_L$ and $f : R \to \Sigma_R$
- Objective: maximize the fraction (or number) of edges $e = (u, v)$ where $(f(u), f(v)) \in \pi_e$.

How does this relate to our two-prover proof system for 3SAT-5? Let $L = [n]$, i.e., create a left vertex for every variable in the 3SAT-5 instance. Let R be the clauses, i.e., create a right vertex for every clause in the instance. Create an edge between every clause and variable in the clause, so every left vertex (variable) has degree 5 and every right vertex (clause) has degree 3. Let $\Sigma_L = [2]$ (we think of this as $\{T, F\}$), and let $\Sigma_R = [7]$ (think of this as the seven satisfying assignments for each clause – note that the interpretation of each symbol in the alphabet as an assignment might be different for each vertex). Finally, for each edge $e = (x_i, C)$, we include in the relation the 7 pairs (out of the 14 possible) which correspond to consistent assignment of x_i and C).

So the left corresponds to prover 1, and the right corresponds to prover 2. Since both sides of the graph are regular, choosing a random clause (right vertex) and then a random variable in that clause (left vertex) is equivalent to choosing an edge of this graph uniformly at random. So we can reinterpret our two-prover proof system as follows: prover 1 assigns labels to the left nodes, prover 2 assigns labels to the right nodes, then we choose a random edge and check whether the labels of the two endpoints are in the relation for that edge.

Thus our analysis of the two-prover proof system implies the following:

Theorem 26.3.1 *Unless $P = NP$, there is some constant $\epsilon > 0$ so that no polynomial time algorithm can distinguish between instances with value 1 and instances with value at most $1 - \epsilon$.*

26.4 Parallel Repetition

It turns out that Label Cover is *far* harder than just the small constant inapproximability of Theorem 26.3.1. To prove this, we'll again go through a proof system point of view. We saw that the inapproximability of Label Cover is related to the soundness of the two-prover proof system. So how can we improve that? How can we decrease the probability of fooling the verifier?

One obvious approach is repetition: what if we just repeat the whole protocol multiple times? If the underlying 3SAT-5 insteance is a YES instance, then the provers can respond correctly in every iteration. But if it's a NO instance, then every time we repeat we have an $\epsilon/3$ probability of catching them in a lie and thus returning NO. So if we repeat this x times, the probability of getting fooled goes down from $(1-\epsilon/3)$ to $(1-\epsilon/3)^x \leq e^{-x(\epsilon/3)}$. So, for example, if $x = \Theta(\frac{1}{\epsilon}\ln n)$, the probability of getting fooled is only an inverse polynomial!

This seems like it would give us great hardness results, but unfortunately our relationship between our proof system and Label Cover requires that the proof system only be one round. So we can't just repeat the whole process multiple times. But can we still use this idea of repetition in any way? What about if, instead of repeating serially, we repeated *in parallel*?

Slightly more formally, what if we choose k random clauses $C_1, C_2, \ldots, C_k$ (instead of just one clause) and for each of these k clauses we choose a random variable $x_1, x_2, \ldots, x_k$ from that clause. Then we ask prover one for an assignment to each of these k variables, and we ask prover two for an assignment to each of the k clauses.

This is a different two-prover one-round proof system, but since it still is such a proof system we can turn it into an instance of Label Cover as we did before: L will be the questions that we might ask prover 1 (so $L = [n]^k$), R will be the questions that we might ask prover 2 (so $|R| = m^k$ where m is the number of clauses), the alphabets are the possible answers (so $\Sigma_L = [2]^k$ and $\Sigma_R = [7]^k$), and the relation for an edge is the obvious combination of the associated original relations:

$$\pi_{(x_1,\ldots,x_k),(C_1,\ldots,C_k)} = \{((\alpha_1,\ldots,\alpha_k),(\beta_1,\ldots,\beta_k)) \in [2]^k \times [7]^k : (\alpha_i,\beta_i) \in \pi_{(x_i,C_i)} \text{ for all } i \in [k]\}$$

So now we have the main question: is making queries in parallel "the same" as asking them in series? This might seem intuitive, but it's not obvious – by asking them in parallel, each prover has more information than if they were asked in series. Eventually it was shown that, unfortunately, the answer is no: the provers can actually do intelligent things to increase their chances of fooling us beyond the $(1-\epsilon/3)^k$ that we would get if we asked them in series. However, in a seminal result, Ran Raz showed that parallel repetition is *almost* as good. This has become known as *Raz's Parallel Repetition Lemma*:

Theorem 26.4.1 *If any assignment to the 3SAT-5 instance satisfies at most $1-\epsilon$ of the clauses, then there is some constant $c > 0$ so that for all positive integers k, no matter what the provers do in the parallel repetition protocol, the probability that the verifier returns YES is at most $(1-\epsilon)^{ck}$.*

When phrased in terms of Label Cover, it gives the following:

Theorem 26.4.2 *There is some $\epsilon > 0$ and $c > 0$ so that for all $k \geq 1$, unless $NP \subseteq DTIME(n^{O(k)})$, there is no polynomial-time algorithm which can distinguish between instances of* Label Cover *with value* 1 *and instances with value at most $(1-\epsilon)^{ck}$.*

Note that the complexity assumption has changed from $P \neq NP$ to $NP \not\subseteq DTIME(n^{O(k)})$, which is a weaker assumption if k is superconstant. This is because the size of the graph in our Label Cover instance will be basically n^k (where n is the size of the original Label Cover graph, which in turn is polynomial in the original 3SAT-5 instance). So this reduction takes more than polynomial time if k is more than a constant.

However, note that by definition $P = \cup_{k\in\mathbb{N}} DTIME(n^k)$, i.e., P is the class of languages that can be decided in time n^k for some k (independent of n). Thus we have the following corollary:

Corollary 26.4.3 *Unless $P = NP$, for any constant $\alpha > 0$, there is no polynomial-time α-approximation for* LABEL COVER.

So parallel repetition allows us to rule out any constant approximation. And if we weaken the complexity assumption a bit, it actually allows us to do even more. Suppose that we set $k = \Theta(\log^{(1-\epsilon)/\epsilon} n)$. Then the new graph has size approximately

$$N = n^k = n^{\Theta(\log^{(1-\epsilon)/\epsilon} n)},$$

and so we get that

$$\log N = \Theta(\log^{(1-\epsilon)/\epsilon} n \cdot \log n) = \Theta(\log^{1/\epsilon} n)$$

and so $\log n = \Theta(\log^{\epsilon} N)$. The inapproximability becomes

$$(1-\epsilon)^{c\Theta(\log^{(1-\epsilon)/\epsilon} n)} \leq 2^{-c' \log^{(1-\epsilon)/\epsilon} n} \leq 2^{-c' \frac{\log^{1/\epsilon} n}{\log n}} = 2^{-c'(\log N)/(\log^{\epsilon} N)} \leq 2^{\log^{1-\epsilon} N},$$

where we choose c appropriately.

Recall that *quasipolytime* was running times of the form $n^{\log^c n}$ for some constant c (these are larger than polynomials, but less than exponentials). This is precisely the running time of applying $\log^{(1-\epsilon)/\epsilon}$-parallel repetition, so we have the following corollary

Corollary 26.4.4 *For any $\epsilon > 0$, unless NP has quasipolynomial time algorithms there is no polynomial time algorithm for* LABEL COVER *with approximation ratio better than $2^{-\log^{1-\epsilon} n}$.*

So under this slightly stronger but still widely believed complexity assumption, we end up with hardness that is more than any polylog but less than any polynomial. It is widely believed that the true hardness of LABEL COVER is polynomial, but this is currently unknown.