18.408 Topics in Theoretical Computer Science Fall 2022
Lecture 21

Dor Minzer

Today we present stronger forms of the PCP theorem that are conjectured to hold but not known. In the upcoming lectures, we will discuss some of their implications and recent related results.

1 On the Structure of the Constraints in Label Cover

1.1 The PCP + Parallel Repetition + Fourier framework

Recall that in the label-cover problem, an instance Ψ consists of a graph $G = (L \cup R, E)$, alphabets Σ_L and Σ_R and a collection of projection constraints on the edges $\Phi = \{\Phi_e\}_{e \in E}$. That is, for each $e \in E$ the constraint Φ_e are defined by a projection map $\phi_e \colon \Sigma_L \to \Sigma_R$ as

$$\Phi_e = \{ (\sigma, \phi_e(\sigma)) \mid \sigma \in \Sigma_L \}.$$

In the first half of the course, we saw that gap-Label-cover$[1, 1-\varepsilon]$ is NP-hard for some absolute constant $\varepsilon > 0$ on alphabets $|\Sigma_L| = k$, $|\Sigma_R| = 2$ where k is an absolute constant. We then used the parallel repetition to improve upon the soundness of the PCP, and in the last lecture we saw how one may use Fourier analysis to get optimal hardness of approximation results for some problems. This framework, that is, the combination of the PCP Theorem, the Parallel Repetition Theorem and Fourier analysis has been very fruitful in the decade or so following the proof of the PCP Theorem, but for some reason there were some (in fact, many) problems for which this approach did not seem to give optimal hardness of approximation results.

To see this, we dissect the hardness result we saw for 3Lin and explain one of the challenges there that we fortunately managed to overcome. As a result of parallel repetition, the alphabet of the sides L and R grows exponentially to k^ℓ and 2^ℓ where ℓ is the number of repetition. Thus, the alphabet of the left side is much larger than that of the right side, hence there are many more points in $\{-1,1\}^{\Sigma_L}$ compared to $\{-1,1\}^{\Sigma_R}$. When proving the hardness result for 3Lin, this point presented some difficulties; more precisely, when we took $x \in \{-1,1\}^{\Sigma_R}$ uniformly and then looked at the pull-back point $y = \phi_{u,v}^{-1}(x) \in \{-1,1\}^{\Sigma_L}$, the point y is very much not random looking and we cannot directly look at the value of the long-code of u there (as it is very cheap to corrupt all of these points). Fortunately, in the case of 3Lin we were able to overcome this issue by using local correction.

There are cases, however, where we cannot use local correction. Suppose that we were trying to prove a hardness result for 2Lin instead of 3Lin (which is the same problem except that in each equation we have two variables), or the very related problem of Max-Cut. In that case we cannot afford to perform local correction: if we wish to perform local correction, we are already investing 2 in that, so we cannot query anything else to compare this value against. This barrier, and other manifestations of it, present themselves in numerous problems wherein one has a candidate construction for a dictatorship test but does not know how to use it for a NP-hardness reduction.

1.2 Circumventing the Local Correction Barrier

Thus, to make progress on problems in which this barrier exists, researchers had to come up with rather ad-hoc solutions. This includes inspecting other parameters of PCP that can be used, on top of the parameters we've discussed in this course, for the purpose of a specific problem. This also includes more ingenious ways of combining the above ingredients to circumvent this barrier (smooth PCPs, randomized noise rates and so on), but even these ideas only led to progress on a limited class of problems. Some notable problems for which these techniques did not work very well are the Vertex Cover problem, the Max-Cut problem and the 2SAT problem.

This situation naturally sparks the question of whether there are yet other, stronger forms of the PCP theorem that would allow one to circumvent these issues altogether. And while we do not know the answer to that, in 2002 a conjecture regarding the existence of such PCP theorem has been made, which we discuss in the next section.

2 The d-to-1 and the Unique-Games Conjectures

2.1 The Statements of the Conjectures

An instance of the d-to-1 Games problem is special type of instance of the Label-cover problem, wherein each constraint is a d-to-1 constraint. Formally:

Definition 2.1. *An instance of d-to-1-Games is $\Psi = (G = (L \cup R, E), \Sigma_L, \Sigma_R, \Phi = \{\Phi_e\}_{e \in E})$, wherein G is a bi-regular bipartite graph, Σ_L and Σ_R are finite alphabet with $|\Sigma_L| = d\,|\Sigma_R|$, and for each $e \in E$, the constraint Φ_e is a d-to-1 constraint. By that, we mean that there is a d-to-1 map $\phi_e \colon \Sigma_L \to \Sigma_R$ such that*

$$\Phi_e = \{\, (\sigma, \phi_e(\sigma)) \mid \sigma \in \Sigma \}\,.$$

In the context of d-to-1-Games, d should be thought of as a small constant, say $d = 2$. The smaller the d, the better. For the smallest possible d, that is, for $d = 1$, d-to-1-Games take the more well-known name *Unique-Games*.

Definition 2.2. *The Unique-Games problem is the d-to-1-Games problem for $d = 1$.*

The d-to-1-Games Conjecture and the Unique-Games Conjecture assert, morally speaking, that the statement of the PCP theorem holds for d-to-1 Games and Unique-Games respectively. More precisely, the d-to-1 Games Conjecture states that

Conjecture 2.3. *For all $d \geqslant 2$ and for all $\varepsilon > 0$, there is $k \in \mathbb{N}$ such that gap-d-to-1-Games$[1, \varepsilon]$ is NP-hard on instances with alphabet sizes at most k.*

For $d = 1$, the situation is a bit more delicate. Given an instance of Unique-Games which is promised to be fully satisfiable, it is possible to efficiently find a satisfying assignment. Indeed, one takes some vertex $u \in L$, guess their label σ_u and then use the 1-to-1 constraints to propagate this and get the labels for all other vertices in the graph. In words, fully satisfiable instances of Unique-Games are easy to solve. The statement of the Unique-Games Conjecture states that, except for that, Unique-Games are just as hard as Label-cover instances:

Conjecture 2.4. *For all $\varepsilon, \delta > 0$, there is $k \in \mathbb{N}$ such that gap-Unique-Games$[1 - \varepsilon, \delta]$ is NP-hard on instances with alphabet sizes at most k.*

2.2 Proving Via Parallel Repetition?

d-to-1 Games. One may attempt to prove Conjecture 2.3 for some $d \in \mathbb{N}$ via a strategy similar to the one we've seen in this course. Namely, start off with a basic result saying that gap-d-to-1-Games$[1, 1-\varepsilon]$ is NP-hard for some $\varepsilon > 0$ and $d \in \mathbb{N}$, and then perform parallel repetition to amplify the gap. And indeed, while the first step works (that is, one can get a basic result of this form), the parallel repetition step does not. Indeed, applying t-fold parallel repetition on a d-to-1-Games instance leads to d^t-to-1-Games instance, so the parallel repetition operation does not preserve d-to-1-ness.

Unique-Games. Ok, but we can still try to prove Conjecture 2.4 for some $d \in \mathbb{N}$ via this strategy, since parallel repetition does preserve uniqueness. That is, we want to start off with a basic result stating that gap-Unique-Games$[1-\varepsilon, 1-\varepsilon']$ is NP-hard for some $\varepsilon < \varepsilon'$, and then perform parallel repetition to amplify the gap. If we had an "ideal" parallel repetition theorem which states that $\mathsf{val}(\Psi^{\otimes t}) = \mathsf{val}(\Psi)^t$ and ε was arbitrarily smaller than ε' (say, $\varepsilon' = \varepsilon^{0.99}$), then this approach could be in fact made to work. Alas, such "ideal" parallel repetition theorem is not known — in fact it is false. Moreover, the basic PCP result for Unique-Games one wants to amplify, is also not known.

In other words, it is unclear how to go about proving Conjectures 2.3 and 2.4. And indeed, despite being proposed in 2002 until recently there hasn't been much progress towards a proof of these results.

2.3 Implications of Conjectures 2.3 and 2.4

In contrast to the lack of progress towards a proof of Conjectures 2.3 and 2.4, there has been much progress in understanding their power and their implications. In the same paper that suggested these conjectures, it was shown that they imply improved inapproximability results for the vertex cover and 2SAT problems that bypassed the best known hardness result that can be achieved by existing PCP techniques.

It took a while longer, but it was later realized that, if true, Conjecture 2.4 in fact implies *tight inapproximability* results for all constraint satisfaction problems. This result, known as Raghavendra's theorem, is a beautiful culmination of many ideas that were developed in UGC based reduction, among which are connections to Fourier analysis, Gaussian geometry and Semi-definite Programming relaxation.

In this course, we will not prove Raghavendra's theorem or even state it, and instead focus on predecessors of it which got the UGC train started. Namely, we are going to discuss the Max-cut and Vertex-cover problems.

3 The Max-cut Problem

Recall that given a graph $G = (V, E)$, a cut in G is a set of vertices $S \subseteq V$. Denoting by $E(S, \bar{S})$ the set of edges that go from S to its complement, the size of the cut defined by S is $E(S, \bar{S})$, and the fractional size of the cut defined by S is $\frac{|E(S,\bar{S})|}{|E|}$.

In an undergraduate algorithm course, one often sees that the Min-cut problem, which asks, given a graph G, to find the smallest cut in it. This is a well known problem in the class P, and a typical textbook way of showing that is by using LP-duality to establish the Min-cut Max-flow algorithm, and then solving the Max-flow problem by one of the many existing polynomial time algorithms for it. What about the maximization version of the cut problem, though?

In the Max-cut problem, the input is again a graph $G = (V, E)$, and the task is to find a cut of maximum size. This problem is another well-known NP-hard problem, and we will care about the approximation version of it. Here, for $\alpha \in (0, 1)$, an α-approximation algorithm for Max-cut is an algorithm that on a graph G outputs a cut whose size is as least $\alpha\mathsf{MC}(G)$, where $\mathsf{MC}(G)$ denotes the size of the maximum cut in G. How well can one approximate Max-cut?

Theorem 3.1. *There is a polynomial time $\frac{1}{2}$-approximation for Max-cut.*

Proof. Given a graph $G = (V, E)$, we choose a cut $S \subseteq V$ randomly, by including each $v \in V$ in S with probability $1/2$. Note that for any edge $e = (u, v) \in E$, the probability it belong to the cut S is $1/2$, so denoting by Z_e the event that e crosses the cut, we get that the expected size of the cut of S is

$$\mathbb{E}\left[\sum_{e\in E} Z_e\right] = \sum_{e\in E} \mathbb{E}\left[Z_e\right] = \sum_{e\in E} \frac{1}{2} = \frac{|E|}{2}.$$

Hence, in expectation the cut S has size $|E|/2$, and by standard techniques again one can de-randomize this algorithm. □

In light of Theorem 3.1 and previous lectures, one may expect that the next result would state that achieving a better approximation ratio than $1/2$ is NP-hard (or maybe UGC-hard since we talked about UGC before). However, here the plot thickens:

Theorem 3.2. *For $\alpha_{\mathsf{GW}} \approx 0.878$, there is a polynomial time α_{GW}-approximation for Max-cut.*

The rest of this lecture is devoted to the proof of Theorem 3.2. Our approach will be to first phrase the Max-cut problem as an integer program, which by itself is not very useful since integer programming is NP-hard. We will then consider a convex relaxation of this program which is known as the Semi-definite Programming relaxation. The benefit of this is that, unlike integer programs, such convex optimization problems can be solved by polynomial time algorithm. The down-side is, though, that we will get a solution to the relaxed version of the problem, which does not give us a cut in the graph. The final step in our approach will be a rounding phase, wherein we will turn the solution of the relaxed program into a Max-cut by a rounding algorithm.

3.1 The Integer Program Formulation

First, we phrase the Max-cut problem as an integer program. For each vertex $v \in V$ we create a variable x_v that is supposed to be assigned a value from $\{-1, 1\}$. The idea is that $x_v = 1$ will represent that v is on the left side, and $x_v = -1$ will represent that v is on the right side. Thus, for $(u, v) \in E$, $x_u x_v = -1$ if and only if (u, v) crosses the cut, and otherwise $x_u x_v = 1$. Therefore, the following program is a formulation of the Max-cut problem over G:

$$\begin{aligned} \max \quad & \tfrac{1}{2} \sum_{(u,v)\in E} 1 - x_u x_v \\ \text{subject to} \quad & x_v \in \{-1, 1\} \qquad \forall v \in V. \end{aligned}$$

However, integer programming is NP-hard in general, so this formulation does not get us anywhere. That being said, this formulation does motivate us to look at a higher dimensional, *Semi-definite Program* (SDP in short) formulation of the problem.

3.2 The Goemans-Williamson Algorithm for Max-cut

In this section, we show the algorithm that proves Theorem 3.2, which goes by the name the Goemans-Williamson algorithm.

3.2.1 The Semi-definite Programming Relaxation

In the SDP formulation of the problem, we allow each variable x_u to take a value in the unit ball in $\mathbb{R}^r$ (where r may be polynomially large in $n = |V|$).

$$\begin{array}{ll} \max & \frac{1}{2} \sum_{(u,v)\in E} 1 - \langle x_u, x_v \rangle \\ \text{subject to} & \|x_v\|_2 = 1 \qquad\qquad \forall v \in V. \end{array}$$

This optimization problem now can be solved, at least approximately. We will not discuss convex optimization in this course further, but we remark that the point is that this program is convex: this is really an optimization problem over the cone of PSD matrices, where the matrix in question is $|V| \times |V|$ matrix of inner products $J = (\langle x_u, x_v \rangle)_{u,v\in V}$.

Let $\{x_v\}_{v\in V}$ be a vector solution to the above program. Amazingly, we can turn this vector-valued solution into pretty good integral-valued solution, that is, a cut in the graph G!

3.3 The Rounding Procedure

In this section, we show how to turn the vector-valued solution to the above SDP program relaxation into a decent cut in G. Suppose the optimum size of the cut in our graph G is $\rho\,|E|$, where $\rho \in [1/2, 1]$. First, it is clear that the optimum of the SDP program is at least $\rho\,|E|$ (why?), so in particular

$$\frac{1}{2} \sum_{(u,v)\in E} 1 - \langle x_u, x_v \rangle \geqslant \rho\,|E|\,.$$

We now generate a randomized cut from the vector solution. Take a random vector h from the unit ball in $\mathbb{R}^m$, and define

$$L = \{\, v \mid \langle x_v, h \rangle \leqslant 0 \}\,; \qquad\qquad R = \{\, v \mid \langle x_v, h \rangle > 0 \}\,.$$

Our goal is to analyze the expected number of edges that crosses the cur (L, R). Fix an edge $(u, v) \in E$; then the probability that (u, v) is cut is $\theta_{u,v}/\pi$, where $\theta_{u,v}$ is the angle between u and v. Thus, by linearity of expectation the expected size of the cut is

$$\sum_{(u,v)\in E} \frac{\theta_{u,v}}{\pi} = \sum_{(u,v)\in E} \frac{\mathsf{Arccos}(\langle x_u, x_v \rangle)}{\pi} \geqslant \sum_{(u,v)\in E} \alpha_{GW}\,(1 - \langle x_u, x_v \rangle) \geqslant \alpha_{GW}\,\rho\,|E|\,.$$

Here, $\alpha_{GW} = \min_{z\in[-1,1]} \frac{\mathsf{Arccos}(z)/\pi}{(1-z)/2}$. Given this expectation guarantee, one can again use standard tools to design a proper approximation algorithm that achieves this approximation ratio.

3.4 The Goemans-Willaimson algorithm for almost bipartite graphs

With a more careful analysis, one can show that if the original size of the cut was very large, say $\rho = 1 - \varepsilon$ for small ε, then the above analysis could be significantly improve.

Theorem 3.3. *Suppose $G = (V, E)$ has a cut of size $(1 - \varepsilon)\,|E|$. Then the expected size of the cut in the Goemans-Williamson algorithm is at least $\left(1 - \frac{2}{\pi}(1 + o(1))\sqrt{\varepsilon}\right)|E|$.*

3.5 Optimal Algorithms for Max-cut

The algorithmic guarantees given by Theorems 3.2 and 3.3 seem rather bizarre. A-priori, there is no reason to believe that the best approximation ratio achievable for Max-cut is provided by this ad-hoc-ish approach of solving a convex programming relaxation and then rounding it to an integral solution.

It turns out, though, that the algorithm we presented today is the best polynomial time approximation algorithm for Max-cut. At least assuming the Unique-Games Conjecture. In the next lecture, we will present a reduction that proves this last assertion.

18.408 Topics in Theoretical Computer Science: Probabilistically Checkable Proofs
Fall 2022