CS 354: Unfulfilled Algorithmic Fantasies April 22, 2019

Lecture 7: Introduction to Unique Games Conjecture

Lecturer: Aviad Rubinstein *Scribe: Wenzheng Li*

In this lecture, we will show the background and motivation of the Unique Games Conjecture, and we will see how to really use it in the next few lectures.

1 Max-Cut

As a motivating example for the Unique Games Conjecture, consider the Max-Cut problem: Given an undirected graph $G = (V, E)$, the goal is to find a subset of vertices $S \subseteq V$ to maximize $|E \cap (S, \overline{S})|$, i.e. the number of edges crossing the cut induced by S.

There are a few algorithms for Max-Cut. One algorithm we have mentioned is an $1/2$ approximation for unconstrained non-monotone submodular maximization.We know that in general $1/2$ is tight for unconstrained non-monotone submodular maximization. But as we saw in the homework, we cannot hope to prove this query complexity lower bound for Max-Cut since we can find the entire graph with polynomial queries.

There is a much simpler algorithm: greedy. Add the vertices one by one. When a vertex arrives, you assign it to the side with fewer degrees of it. This also gives a $1/2$ approximation.

An even simpler algorithm is to simply pick a cut uniformly at random. Each edge crosses the cut with probability $1/2$, so this also gives a $1/2$-approximation

Something else we can do is writing a linear programing relaxation as follows. We can solve this linear programming efficiently and round the fraction solution to an integer solution. The approximation ratio is still $1/2$.

$$\begin{aligned} \max \quad & \sum_{(u,v)\in E} z_{(u,v)} & \\ s.t. \quad & 0 \le x_u \le 1 & \forall\ u \in V \\ & 0 \le z_{(u,v)} \le \min\{2 - x_u - x_v, x_u + x_v\} & \forall\ (u,v) \in E \end{aligned}$$

These was the-state-of-art for decades and then amazingly Goemans and Williamson [1] used the following semidefinite programming relaxation to give a ≈ 0.878 approximation for Max-Cut. Instead of relaxing the integer constraint for each vertex variable x_u to be $x_u \in [0, 1]$, they relax it to be a vector in the unit sphere, $||x_u||_2 = 1$.

$$\begin{aligned} \max \quad & \sum_{(u,v)\in E} \frac{1 - x_u \cdot x_v}{2} & \\ s.t. \quad & ||x_u||_2 = 1 & \forall\ u \in V \end{aligned}$$

To see how to use semidefinite programming, we first solve this semidefinite program and get some vectors in the unit sphere. To round those vectors into a cut for the original problem, we

Table 1: Approximation algorithms for Max-Cut

Algorithm	approximation factor
Non-monotone submodular maximization	1/2
Greedy	1/2
Random	1/2
LP relaxation	1/2
SDP relaxation	≈ 0.878

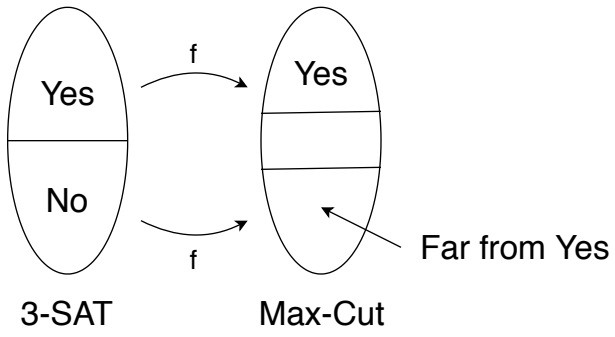
pick a random hyperplane and set of vertices in the same side of the hyperplane to be on the same side of the cut. The crucial observation is that for any edge $(u, v) \in E$, the probability of (u, v) being separated by the cut is $2\arccos(x_u \cdot x_v)/\pi$, which is at least $0.878 \cdot \frac{1 - x_u \cdot x_v}{2}$. Since $\sum_{(u,v)\in E} \frac{1 - x_u \cdot x_v}{2}$ is at least the value of the maximum cut, this algorithm gives a 0.878 approximation for Max-Cut problem.

Looking back on the history of the problem, conjecturing that 1/2 is optimal was natural, but the constant 0.878 (which comes from a numeric solution to a maximization problem involving a trigonometric function) seems arbitrary. Morally, this should not be optimal! But we are going to prove that assuming the Unique Games Conjecture, this is actually the best we can do [2]. And more amazingly, the Unique Games Conjecture implies tight bounds for a bunch of other problems.

2 Hardness of Approximation – "PCP Theorem"

Remember in the last week, the standard hardness reduction approach we used is to map all YES instances of 3-SAT to some YES[1] instances of Max-Cut and all NO instances of 3-SAT to some NO instances of Max-Cut. Now if we want prove the hardness of approximation, we need to map all NO instances to not only NO instances but also instances far away from YES in Max-Cut as following.

[1]Here YES instances for Max-Cut means instance with cut value larger than some threshold. And far away from YES means the cut value is no even close to the threshold.



Here are some interesting observations about the reduction. Let φ_Y and φ_N be some YES and NO instance of 3-SAT. The reduction guarantees that $f(\varphi_Y)$ and $f(\varphi_N)$ are far from each other. But the point is the reduction doesn't know which instance is YES because distinguishing between YES and NO is NP-hard and the reduction runs in polynomial time. So intuitively every two instances in 3-SAT should be mapped to instances in Max-Cut that are far away from each other. In other word, the reduction should be some kind of error-correcting code. And indeed almost every hardness of approximation reduction uses error-correcting codes.

3 Error-Correcting Code

An error-correcting code C is a subset of all strings over some alphabet Σ and length n. It has the special property that every two codewords $x \neq y \in C$ are far from each other.

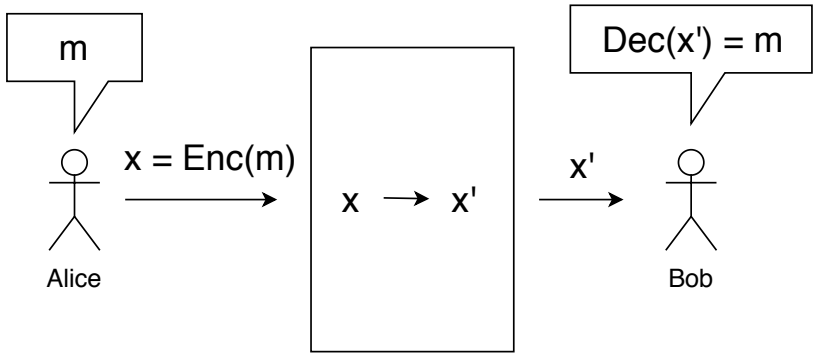
In the classical application, the error-correcting code comes with two functions: $Enc : \{0,1\}^{\log_2(|C|)} \to C$ that encodes messages as codewords; and a decoding function $Dec : \Sigma^n \to \{0,1\}^{\log_2(|C|)}$ that decodes any possible string (not necessarily codeword) back into a message. The typical story of the error-correcting code is that Alice want to communicate with Bob but someone is corrupting their channel. So if Alice want to send message m to Bob, she will send $x = Enc(m)$ instead. The corruption step will take x to x' which is close to x. Then Bob receives x' and can decode x' to be $m = Dec(x')$ because all other codes are far from x and thus also far from x'.

3.1 Example: Low degree polynomials

One example of error-correcting codes is low-degree polynomials (also called the Reed-Solomon Code). For a message $m \in Z_q^{t+1}$, first we map it to a polynomial with degree t over F_q, $m(z) = \sum_{0 \leq i \leq t} m_i z^i$. The codeword encoding m will be evaluating this polynomial everywhere

$$Enc(m) = (m(0), m(1), \cdots, m(q-1)) .$$

In order to show that this code is a good error-correcting code, we need to show that for any two different degree t polynomials over F_q, x and y, their value vector are far from each other. Notice that $y - x$ is a non-zero degree t polynomials over F_q, which means it can only have at most t roots in F_q. Thus the value vector of x and y must differ in at least $q - t$ places.



3.2 The Long Code

The code we will use is called Long Code. We will think of our message m as just one number. And we will map messages to functions. More precisely, for any message $m \in [q]$, we will encode m to be a function $f_m : \{-1,1\}^q \to \{-1,1\}$ s.t. $f_m(x) = x_m$. To prove the distance between two codewords are large, consider two different messages m and m',

$$\Pr_x[f_m(x) \neq f_{m'}(x)] = \Pr_x[x_m \neq x_{m'}] = \frac{1}{2},$$

which is actually almost the best we can hope to get.

The disadvantage about Long Codes is that the ratio is double exponential since the the message is only $\log q$ bits long but the code is 2^q bits long. But Long Codes also have many advantages:

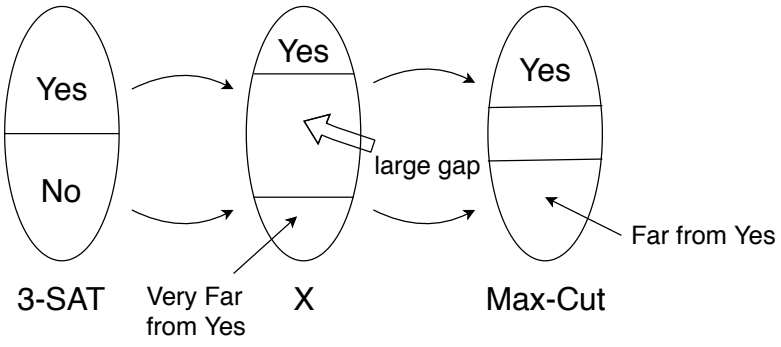
- It is a binary code (i.e. alphabet is $\{-1,1\}$).
- It is 2-query local testable, which roughly means we only need to read two bits of a string to test if it is a codeword or far from the code. [We will discuss this property in more detail in coming lectures.]
- The Long Code is symmetric, which means that permuting $[q]$ won't change the code.
- As we showed above, it has nearly optimal distance compared to random codes.

4 Alphabet Reduction

The PCP Theorem is quite difficult, and proving it with good parameters is even more complicated. Instead of proving it from scratch for each problem we want to show is hard, we consider one good reduction from 3-SAT to a generic intermediate problem X with a huge gap and other desirable properties. Then we can have an "easy" (easier than full PCP theorem) reduction from X to each problem that we care about.

In order to have a large gap, X should have a large alphabet (more on that later). So this last step of reduction from X to our specific problem is often called *alphabet reduction*

We will think of what X should be later (it is a constraint satisfaction problem). For now assume that X is Σ-Coloring problem and we will try to reduce it to Max-Cut using alphabet reduction.



4.1 Constructing the vertices

For each vertex in the coloring instance, we have a corresponding "cloud" with 2^Σ vertices in the cut instance. For vertex u we assume the corresponding u-cloud has vertices $u_0, \cdots, u_{2^\Sigma - 1}$.

Given a cut $S \subset V_{\text{Max-Cut}}$, its intersection with the u-cloud can also be seen as a boolean function $f^u : \{-1,1\}^\Sigma \to \{-1,1\}$, where

$$f^u(x) = \begin{cases} 1 & u_x \in S \\ -1 & u_x \notin S \end{cases}.$$

We denote $f_i : \{-1,1\}^\Sigma \to \{-1,1\}$ as the long code for color $i \in \Sigma$. Intuitively, if vertex u takes color i in the coloring problem, for the reduced instance in the cut problem we want $u_x \in S$ if and only if $f_i(x) = 1$.

4.2 Edges: Attempt 1

We want to add edges between clouds of the new graph such that if there is a coloring satisfying many edges in the original graph then the long code of the assigned colors will induce a large cut in the cloud graph, and on the other hand if there is a large cut of the cloud graph we can extract a coloring to satisfy many edges in the original graph.

For example, suppose that there exist colors $\sigma(u), \sigma(v)$ that satisfy the constraint between X-vertices u, v. Suppose further that for $x, y \in \{-1,1\}^\Sigma$, we have that the Long Code encodings of the colors satisfy

$$\underbrace{[Enc(\sigma(u))]_x}_{=f_{\sigma(u)}(x)} \neq \underbrace{[Enc(\sigma(v))]_y}_{=f_{\sigma(v)}(y)}.$$

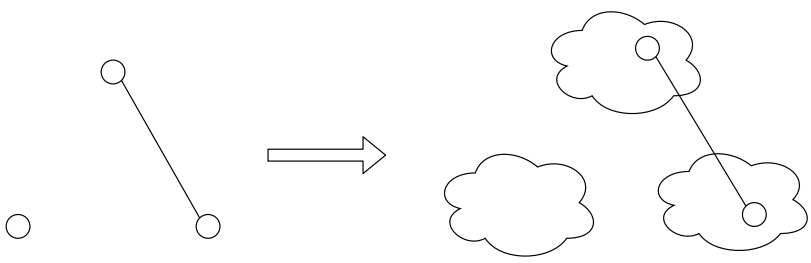
Then, intuitively, we expect Max-Cut-vertices u_x and v_y to be on opposite sides of the cut, so we want to add an edge between.

4.3 Analysis: Attempt 1

Completeness

Completeness proves the top arrows in Figure 4, i.e. if there exists a good assignment for X, then there exists a good cut.

If we connected the edges correctly (we didn't...), we expect given a coloring σ for the X-instance, we can construct a cut that crosses a certain fraction of the edges between the clouds of any pair $u, v \in V_X$ such that $(\sigma(u), \sigma(v))$ satisfies the assignment.



Soundness

Soundness proves the bottom arrows in Figure 4, i.e. if there does not exist a good assignment for X, then there does not exist a good cut. Typically this is proven the other way, namely if there is a good cut, then X has a good assignment.

At this point we observe that Σ-coloring was a bad candidate for the intermediate problem X: a random coloring of the X instance satisfies $(1-1/|\Sigma|)$-fraction of the constraints on X. Therefore the corresponding cut will have at least $(1-1/|\Sigma|)$-times as many edges as the cut in the YES case. In other words, Σ-coloring obviously doesn't have a large gap between YES and NO instances.

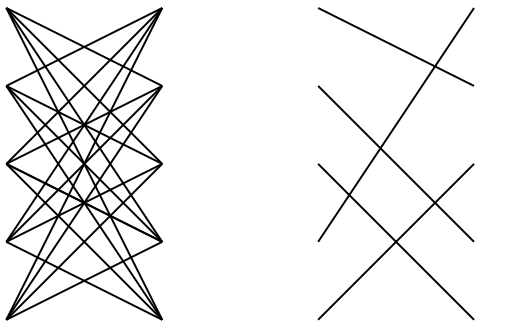
5 Unique Games Conjecture

The discussion above implies that coloring may not be a good candidate for an intermediate problem X. At the very least, we want a good candidate for X to have a very low value for a random assignment. The hardest constraint for random assignments is where each color $\sigma(u) \in \Sigma$ for u can only match to one (*unique*) $\sigma(v) \in \Sigma$ for v. In other words, each constraint is defined by a permutation $\pi_{u,v} : \Sigma \to \Sigma$. This captures the strongest intermediate problem X we can possibly hope for. So starting from such an X makes the reduction to the final problem we care about as easy as we can possibly hope for.

Conjecture 5.1 (Unique Games Conjecture)**.** *For any $\gamma, \epsilon > 0$, there exists an alphabet Σ, graph $G_{ug} = (V_{ug}, E_{ug})$ and a permutation constraint for every edge $(u, v) \in E_{ug}$, $\pi_{uv} : \Sigma \to \Sigma$, such that it is NP-hard to distinguish*

- *(YES) $\exists\ \sigma : V_{ug} \to \Sigma$ to satisfy at least $1 - \gamma$ fraction of constraints,*
- *(NO) $\forall\ \sigma : V_{ug} \to \Sigma$ at most ϵ fraction of constraints are satisfied.*

Note that here we don't have perfect completeness for the YES instances, because that is actually impossible. If we want to test whether a graph with permutation constraints has a perfect assignment to satisfy all the constraints or not, we just need to enumerate the assignment for an arbitrary vertex and compute the rest of the assignment according to constraints. Since each constraint is a permutation, there is at most one possible YES assignment for any initial assignment.



So we just need to verify $|\Sigma|$ many possible assignment to see if there exists a perfect assignment or not.

We don't actually know that the problem in Conjecture 5.1 is NP-hard, or intractable at all. But we also don't know any algorithms that provably solve it. So Unique-Games-hardness (i.e. reductions assuming Conjecture 5.1) at least capture the limitation of all currently known algorithmic techniques.

In the next lecture we will show assuming this conjecture, Goemans and Williamson's SDP algorithm has the optimal approximation ratio for Max-Cut problem.

References

[1] M. X. Goemans and D. P. Williamson. .879-approximation algorithms for max cut and max 2sat. In *Proceedings 26th ACM Symposium on Theory of Computing (STOC)th Annual ACM Symposium on Theory of Computing (STOC*, pages 422–432, 1994.

[2] Subhash Khot, Guy Kindler, Elchanan Mossel, and Ryan O'Donnell. Optimal inapproximability results for max-cut and other 2-variable csps? *SIAM Journal on Computing*, 37(1):319–357, 2007.