CS880: Approximation and Online Algorithms	**Scribe:** Rojin Rezvan, Jialu Bao
Lecture 27: PCP & UGC	**Date:** Dec. 2nd, 2019

27.1 Overview

In this lecture, we will first go through a review on different approximability classes and methods used in them. In section 3, we will discuss optimization problems and gap problem, a tool used to reduce inapproximability to decision problems. In section 4 we will discuss some definitions related to hardness of problems and languages, and in section 5 we will give definition of SAT and 3SAT problem together with providing an approximation algorithm for it. We begin discussing about Probabilistic Checking of Problems (PCP) in chapter 6 and discuss some of its properties and in section 7 we will show how it is used to achieve inapproximability results using 3SAT. In section 8 we will briefly discuss repetition of PCPs. In section 9 we will discuss Hastad's 3bit PCP and how it is used to get inapproximability results and at the very last section we will discuss unique game conjecture, its formulation in 2-prover 1-round format and one of its applications.

27.2 Classes of Approximability

In this section we will briefly summarize approximation algorithms we have seen so far based on their level of approximation and methods used in each category.

For this purpose, we will first start with definition of an approximation problem:

Definition 27.2.1 (Approximation Ratio) *An algorithm for problem P has an approximation ratio α if for instances I, the algorithm produces a solution of cost at most $\alpha.OPT(I)$ if P is a minimization problem and $\alpha \geq 1$ and of cost at least $\alpha.OPT(I)$ if P is a maximization problem and $\alpha \leq 1$.*

We can briefly summarize classes of approximability in below categories:

1. $poly(n)-$approximation
2. constant factor:

   Example 1: $2-$approx algorithm for TSP by considering MST.

   Example 2: $1.5-$approx algorithm for TSP by considering MST and min cost matching.
3. Polynomial Time Approximation Schemes(PTAS) $(1+\epsilon)$ or $(1-\epsilon)$ approximation algorithms with run time polynomial with parameters n (problem input size) and $\frac{1}{\epsilon}$, such as $n^{\frac{1}{\epsilon}}$.

   In such problems there are two general approaches:

   - Type 1: Round down input numbers to limit their possible set of choices, then use dynamic programming.

Example: Knapsack, rounding down the values to the nearest multiple of $\frac{cc}{n}$ where c is the maximum value and n is the number of objects.

- Type 2: Define set of simple solutions, find minimum cost of simple solutions in polynomial time, show and arbitrary solution can be changed to a simple one without losing a lot.

  Example: Euclidean TSP, square network solution.

To name some of the general methods used in approximation algorithms we can mention linear programming, randomized rounding and semi-definite programming.

27.3 Optimization and gap problems

Problems in general can be divided into two categories: The ones in which the aim is to compute a certain value, or decision problems in which on every input we decide whether to output a YES or NO.

Definition 27.3.1 (Gap problem) *For a maximization problem I and $A < B \in \mathbb{R}^+$, the corresponding $[A, B]-$gap problem is the following decision problem:*

$$YES = \{x | OPT(x) \geq B\}$$

$$NO = \{x | OPT(x) < A\}$$

Gap problem is used to transform an optimization problem to a decision problem. Below we will see how hardness of approximation of a maximization problem can be related to hardness of gap problem:

Theorem 27.3.2 *If the $[A, B]-$gap version of a maximization problem is NP-hard, then it is NP-hard to approximate the maximization problem to within a factor $\frac{A}{B}$, assuming $P \neq NP$.*

Proof: Assume by contradiction that there is algorithm C to approximate the problem to within a factor $\frac{A}{B}$ in polynomial time. Use C to make an algorithm for gap problem in the below manner:

If $C(x) \geq A$, return YES. Otherwise return NO.

We will now show that this algorithm yields a working algorithm for gap problem and hence it is a contradiction.

If x is a YES instance of gap, then $OPT(x) \geq B$ and we get:

$$C(x) \geq \frac{A}{B} OPT(x) \geq A$$

If x is a NO instance of gap, then $OPT(x) < A$ and we get:

$$C(x) \leq OPT(x) < A$$

So we built a polynomial time algorithm for $[A - B]-$gap problem using the polynomial time algorithm C that we assumed existed. This contradicts the $P \neq NP$ assumption. ■

27.4 Some definitions

In this section we will discuss some definitions which will be used in the following sections.

Definition 27.4.1 (NP-Problems) *Nondeterministic Polynomial time, or NP problems, are a set of decision problems for which the problem instances where the answer is yes have proofs verifiable in polynomial time. For no instances, there exists no proofs.*

In other words, a language L is in NP if there exists a polynomial time verifier V and a constant c such that for an input x of length n, the following two properties hold:

1. *(Completeness) If $x \in L$ then there exists a witness (proof) y of length $O(n^c)$ such that V accepts the pair $< x, y >$.*
2. *(Soundness) If $x \notin L$ then for all witnesses y of length $O(n^c)$, V rejects $< x, y >$.*

Definition 27.4.2 (NP-hard) *A problem H is NP-hard when every problem L in NP can be reduced in polynomial time to H: assuming a solution for H takes 1 unit time, H'¼ solution can be used to solve L in polynomial time.*

Definition 27.4.3 (NP-complete) *A problem is NP-complete when it is in NP and it is also NP-hard.*

Definition 27.4.4 ($NTIME(f(n))$) *The set of decision problems that can be solved by a non-deterministic Turing machine which runs in time $O(f(n))$ is called $NTIME(f(n))$.*

27.5 3SAT and MAX-3SAT Problem

Definition 27.5.1 (3CNF form) *We say a formula is in 3CNF form when it is in the below form:*

$$(x_{11} \vee x_{12} \vee x_{13}) \wedge \ldots \wedge (x_{m1} \vee x_{m2} \vee x_{m3})$$

In the above formula, the variables named differently can be the same. We denote n as the number of variables and m as the number of clauses (each of the formulas in the parenthesis is called a clause.)

Definition 27.5.2 (3SAT Problem) *Can we find values for x_{i1}, x_{i2}, x_{i3} for all $1 \geq i \geq m$ to satisfy a given 3CNF formula? (By satisfying we mean that all the clauses have to return true value.)*

Definition 27.5.3 (Max-3SAT) *What is the maximum number of clauses that can be satisfied simultaneously?*

Theorem 27.5.4 *There exists an assignment that satisfies at least $\frac{7}{8}$ fraction of clauses in expectation.*

Proof: Consider an answer in which we randomly assign each of the n variables 0 or 1 values with same probability. Let $Y_i = 1$ if clause i is satisfied with such answer and $Y_i = 0$ otherwise. We have:

$$\mathbf{E}[Y_i] = 0.\frac{1}{8} + 1.\frac{7}{8}$$

Now if we have $Y = \Sigma_{i=1}^{m} Y_i$ then we get:

$$\mathbf{E}[Y] = \Sigma_{i=1}^{m} \mathbf{E}[Y_i] = \frac{7}{8}m$$

Since the expected value of Y over all possible random settings of the n variables is $\frac{7}{8}m$, there has to be at least one assignment in which the number of satisfied clauses is at least $\frac{7}{8}m$. ■

Theorem 27.5.5 *There exists an approximation algorithm with factor $\frac{7}{8}$ that works in polynomial time in expectation.*

Proof: As done in 27.5.4, keep generating randomly until you get an assignment with at least $\frac{7}{8}$th of its clauses satisfied.

Lemma 27.5.6 *If a trial can success with probability p, the expected nuber of trials ecessary until success is $\frac{1}{p}$*

Proof: If X is a random variable that denotes the number of trials necessary until success, we get:

$$\mathbf{E}[X] = \Sigma_{j=1}^{\infty} jp(1-p)^{j-1} = \frac{1}{p}$$

■

Define p_j as the probability of having j satisfied clauses by an assignment when we randomly assign the variables 0 and 1 with same probability. Moreover, let p be the probability of having at least $\frac{7}{8}$ fraction of the clauses satisfied. Then we get:

$$\begin{aligned} \frac{7}{8}m = \mathbf{E}[Y] =& \Sigma_{1 \le j \le \frac{7}{8}m} j p_j + \Sigma_{\frac{7}{8}m < j \le m} j p_j \\ \le& (\frac{7}{8}m - \frac{1}{8}) \Sigma_{1 \le j \le \frac{7}{8}m} p_j + m \Sigma_{\frac{7}{8}m < j \le m} p_j \\ \le& (\frac{7}{8}m - \frac{1}{8}.1) + kp \\ \Longrightarrow p \ge& \frac{1}{8m} \end{aligned} \tag{27.5.1}$$

Now using 27.5.6 we get:

$$\mathbf{E}[\text{Number of variable samplings necessary to get a good assignment}] = \frac{1}{p} = 8m = Poly(m)$$

So we proved that this algorithm gives a $\frac{7}{8}$-approximation in expected polynomial time.

■

Remark 27.5.7 *3SAT is in NP because given a formula and a proof (assignment of values to variables) satisfiability of all clauses is checkable in polynomial time.*

With the method mentioned above, it is easy to check satisfiability of every given 3CNF formula. However, in order to do so we need to check all clauses, because it might be the case that the very

last clause we check is not satisfied by the given assignment. Now what if we had a way to check only a constant part of proof? This brings us the idea of local checking of proofs. In this context, we are going to allow queries to be chosen in a randomized manner (because otherwise there would clearly exist a polynomial time verifier). Moreover, as we want our checker to work in polynomial time, we allow using at most logarithmic number of random bits. In this way, verifier can check all possible randomly produced values in polynomial time and simulate a random producer. In doing so, we are ready to give up on some of the constraints we had in an NP problem. More precisely, we will give up completeness to probability of at least c, and we will give up soundness to probability of at most s.

27.6 PCP

Definition 27.6.1 *A language L is in $PCP_{c,s}(r,q)$ if there exists a polynomial time verifier V that given an input x of length n and a purported proof y, runs in time poly(n) bits of randomness, makes $q(n)$ queries to y and satisfies the following two properties:*

1. *(Completeness): If $x \in L$ then there exists a y such that V accepts the pair $< x, y >$ with probability at least c.*
2. *(Soundness): If $x \notin L$, then for all y, V accepts $< x, y >$ with probability at most s.*

Theorem 27.6.2 *$NP = PCP_{1,0}(0, poly(n))$: excellent completeness and soundness, many queries*

Proof: This PCP precisely follows the definition of NP. In definition of NP we have perfect completeness ($c = 1$), perfect soundness (no false positives, $s = 0$), no randomization and polynomial access to proof. (proportional to size of proof) ■

Theorem 27.6.3 $NP = PCP_{\frac{1}{2^{O(n)}},0}(poly(n), 0)$

Proof: We will show that:

1. $NP \subseteq PCP_{\frac{1}{2^{O(n)}},0}(poly(n), 0)$
2. $PCP_{\frac{1}{2^{O(n)}},0}(poly(n), 0) \subseteq NP$

First, we will show 1. Consider a problem $Q \in NP$. We should design a PCP verifier for it that would give us completeness of $\frac{1}{2^{O(n)}}$ and soundness of 0, using $poly(n)$ random bits and reading nothing from the proof. Keep in mind that we already have a verifier W for Q for NP.

The verifier is as follows: The verifier randomly produces a witness w of length $poly(n)$ using this many random bits that it has. Then it gives w to W and returns result of W. Now if $x \in Q$, then there exists at least one witness for x, and so we will guess it randomly with probability at least $\frac{1}{2^{O(n)}}$ and hence accept it. If $x \notin Q$, there is no w that W would accept. So W will always return false and we will get soundness 0. We have used $poly(n)$ random digits to produce the witness and read nothing of the given proof.

Now let's prove 2. So we will consider a problem $Q \in PCP_{\frac{1}{2^{O(n)}},0}(poly(n), 0)$ and assume a verifier W for it. Now we know that W accepts a correct witness with probability $\frac{1}{2^{O(n)}}$. So now we can assume that the proof given to the NP verifier by adversary is a random witness that we produce via W, give it to W and return the result. Basically the verifier for NP, say V, uses how W works but does not use it's random generator. As soundness of W is 0, we know no problem that is not in the language would be accepted. ■

This PCP has the benefit of perfect soundness. However, the completeness factor is very bad. We would want to interpolate between all these factors. In the following theorem, we will do that using 3SAT problem. We note that we can do this, as 3SAT is an NP-complete problem.

Theorem 27.6.4 $NP = PCP_{1,1-\frac{1}{n^{O(1)}}}(O(logn), 3)$

Proof: We need to prove inclusion from both side in the above equality. For $NP \supseteq PCP_{1,1-\frac{1}{n^{O(1)}}}(O(logn), 3)$ consider the below lemma.

Lemma 27.6.5 $PCP_{c,s}(r(n), q(n)) \subseteq NTIME(2^{O(r(n))}q(n)n^{O(1)})$

Proof: In order to prove this, we will show that running the PCP verifier deterministically will take $O(2^{O(r(n))}q(n)n^{O(1)})$ time. This is because we have $r(n)$ random bits, by which we can produce $2^{r(n)}$ different strings which we will use for picking queries. Moreover, for each random string we can pick $q(n)$ queries and each of the queries will take $O(n)$ time to look up. Moreover, simulating such strings will take time $O(2^n)$. This will give us $O(2^{r(n)}q(n)O(n))$. So we can use the verifier for PCP in the following manner to get a verifier for NP:

1. Produce all the $2^{r(n)}$ possible random strings and consider the $q(n)$ queries each will point to. (In total we will get $2^{r(n)}.q(n)$ locations of witness to check.)
2. Each look up will take $O(n)$ time.
3. Simulating each of the random bits will take $O(2^{r(n)})$ time.
4. Doing these steps we can find the explicit possibility of acceptance of an instance. If this probability was at least c, accept, otherwise reject.

■

Now using the above lemma, it suffices to set $r(n) = logn$ and $q(n) = O(1)$. Then we will have $O(2^{r(n)}q(n)O(n)) = O(n^c)$ for some constant c. And we know that $O(n^c) = NP$.

In order to prove the other side, we will consider the 3SAT problem and show that it belongs to PCP. And doing this is enough to show the theorem because 3SAT is a NP-complete problem. Now we will build a PCP verifier for 3SAT. Assume that the 3SAT has m clauses. We can consider a witness here as an assignment of true/false values to variables. The PCP verifier will do as following: It will randomly pick one clause of the witness. If that clause was satisfied, it will accept, and reject otherwise. Now if an instance formula Φ is satisfiable and such witness is given, the verifier will accept with probability 1. If Φ is not satisfiable, then it contains at least one unsatisfied clause. By picking the clause randomly, we will get that with probability at least $\frac{1}{m}$. So the soundness factor will be at most $1 - \frac{1}{m}$. This will conclude the other side of the equality. ■

In the above theorem, the PCP has the appropriate number of random bits and it has a constant access to query as we were aiming from the beginning. However, its soundness factor is very high. Here we note that in the 3SAT problem, it is possible that all the clauses that we check are satisfied, except for the very last one. This means that we might need to check all clauses of a witness to make sure if it is in our language or not, and so a constant number of queries to witness cannot guarantee we make the right choice. However, if we are aiming to have a low soundness factor, we must be able to solve this somehow. For this purpose, we note that in the normal encodings, it is possible that the error lies only in a constant part of the witness. But we need to have a way of encoding any witness equivalently such that the error is more uniformly spread in the witness, so we have a more chance of catching it when we randomly pick a constant part of the witness to check.

Theorem 27.6.6 (PCP Theorem) $NP = PCP_{1,\frac{1}{2}}(O(logn), O(1))$

Remark 27.6.7 *We proved one side of PCP theorem in the previous theorem. However, the side* $NP \subseteq PCP_{1,\frac{1}{2}}(O(logn), O(1))$ *is the hard side which we will omit here due to its complexity[Arora et al., 1998][Aro*
We must note that this is a very powerful theorem as it gives us a tool to check a proof locally and in polynomial time with perfect completeness and an acceptable soundness.

27.7 PCP and Inapproximability of MAX-3SAT

PCP theorem is mainly used to gain inapproximability results. The following theorem will help us find an inapproximability factor for MAX-3SAT problem.

Theorem 27.7.1 *The following two statements are equivalent:*

1. $NP \subseteq PCP_{1,\frac{1}{2}}(O(logn), O(1))$
2. *There is a mapping reduction from 3SAT to 3SAT that maps a satisfiable formula to a satisfiable formula and maps unsatisfiable formula to formulas for which no assignment satisfies more than a* $1 - \epsilon$ *fraction of the clauses, for some constant* $\epsilon > 0$.

Proof: First we will assume 2 and prove 1.

Suppose such reduction exists. We will build a $PCP_{1,\frac{1}{2}}(O(logn), O(1))$ verifier for 3SAT. For that, we follow these steps:

1. Use reduction to obtain from *Phi* a formula ψ, and assume that the witness is an assignment to ψ.
2. Verifier randomly picks a clause of ψ and query 3 bits regarding that clause to check its satisfiability.
3. If φ is satisfiable, then so is ψ by the reduction, and so the verifier accepts with probability 1.
4. If φ is not satisfiable, then assignment violates at least ϵ fraction of the clauses of ψ. So the verifier accepts with probability at most $1 - \epsilon$.

Now this will give us a $PCP_{1,\epsilon}(O(logn), O(1))$. In order to get to soundness of $\frac{1}{2}$, repeat the above process, and pick the random clause uniformly and independent from the previous choices. After k number of steps, we will get the soundness factor of $(1-\epsilon)^k$, and choosing right constant number k we will get to $\frac{1}{2}$. Note that we can do this here because ϵ is a constant so we need to keep picking new queries only constant number of times. We could not decrease soundness factor in the previous theorem in this manner because we had $1-\frac{1}{n}$ and the number of times this needs to be multiplied to itself to get to a constant is not independent of n.

Now we will assume 1 and prove 2.

Suppose $NP \subseteq PCP_{1,\frac{1}{2}}(O(logn), O(1))$. So there exists a $PCP_{1,\frac{1}{2}}(r,q)$ verifier V for 3SAT in which $r = O(logn)$ and q is a constant. Now this verifier can receive at most 2^q values from q queries. Now we will design the following 3SAT:

-input: φ

-output: ψ

-variables of ψ: all possible bits of proof that can be queried (note that there will be polynomial number of these)

-clauses: For all random strings R of length r

- Determine which bits of the proof will be queried.
- Determine which 2^q assignments of variables would make the verifier accept.
- For each, build a clause with those variables.

The above process will give us at most $2^q 2^r$ clauses, each of length q variables.

Now using the following transformation, we will change such formula to a 3SAT: Consider a clause of length q:

$$(l_1 \vee l_2 \vee \ldots \vee l_q)$$

It is easy to see that the following formula is equivalent to this one using new variables $z_2, \ldots, z_{q-2}$:

$$(l_1 \vee l_2 \vee z_2) \wedge (\overline{z_2} \vee l_3 \vee z_3) \wedge (\overline{z_3} \vee l_4 \vee z_4) \wedge \ldots \wedge (\overline{z_{q-3}} \vee l_{q-2} \vee z_{q-2}) \wedge (\overline{z_{q-2}} \vee l_{q-1} \vee l_q)$$

So we successfully built a 3SAT formula ψ from φ. First we note that this new formula has at most $q2^q2^r = poly(n)$ clauses. Moreover, we have that:

- If φ is satisfiable, then ψ is also satisfiable by assigning variables values according to a proof that makes the verifier accept with probability 1 due to definition of $PCP_{1,\frac{1}{2}}$.

- If φ is not satisfiable, consider block of clauses for each random string R. It can only be satisfied by some assignment if the verifier accepts this proof for that R. From $PCP_{1,\frac{1}{2}}$ it follows that at least $\frac{1}{2}$ of the clauses will not be satisfied. So we have at least one unsatisfied clause, and so there are at least $\frac{1}{2}2^r$ unsatisfied clauses in psi. Now it suffices to set $\epsilon = \frac{\frac{1}{2}2^R}{2^r 2^q q} = \frac{1}{22^q q}$. ∎

Theorem 27.7.2 *There is no PTAS for MAX-3SAT unless P=NP.*

Proof: Statement 1 in last theorem holds according to the PCP theorem. Therefore for any 3SAT formula φ there exists an equivalent formula ψ as in the theorem with some $\epsilon > 0$ factor.

Assume a $(1-\epsilon')$-approximation factor algorithm exists ($\epsilon' < \epsilon$) for MAX-3SAT problem. Using this, we will make an exact polynomial time algorithm for 3SAT problem, which is a contradiction to P=NP because this problem is NP-complete. Consider the below algorithm:

As an input it receives a formula φ. Applying the reduction from the previous theorem it gets a new formula ψ. Applying the approximation algorithm, it will get a $(1-\epsilon')$-approx for ψ.

Now if φ is satisfiable, then using the same assignment from the approximation algorithm we can satisfy at least $(1-\epsilon')$ fraction of the clauses. If φ is not satisfiable, then using ψ we can get an assignment in which no more than $(1-\epsilon)$ fraction of clauses are satisfied. So we can have the below classifier for φ, when we consider the result of the approximation algorithm:

- More than $1-\epsilon$ fraction satisfied $\rightarrow$ YES
- Otherwise $\rightarrow$ NO

Note that here we used the fact that since $\epsilon' < \epsilon$, we have $1-\epsilon < 1-\epsilon'$. ■

Remark 27.7.3 *In general we are searching for PCPs with lower soundness and less number of queries. In the next section we will see how to decrease the soundness, and in chapter 9 we will see how we can decrease the number of queries necessary using a theorem as powerful as PCP theorem called Hastad's 3bit PCP.*

Remark 27.7.4 *In all this discussion, when talking about equivalence of NP to some set, we can simply consider 3SAT problem and discuss about that, because 3SAT is an NP-complete problem and can be reduced to any other problem in NP.*

27.8 Reducing Soundness

In the last section we saw how we decreased the soundness to $\frac{1}{2}$ by repeating the verifier process an appropriate number of times and then returning the and of the values. The notion of sequential repetition comes from the same idea:

Definition 27.8.1 (Sequential Repetition) *Whenever we are given a verifier for $PCP_{c,s}(f(n), g(n))$, we can have the verifier repeat the process on separate copies of the proofs and run itself independently for k times. In the end, if result of all copies was 1, we will return 1 and 0 otherwise. In this way, we can improve the soundness down to s^k but our query size will have to increase by a factor of k. We call this process sequential repetition.*

Definition 27.8.2 (Parallel Repetition) *In this form of repetition, the verifier considers k parallel runs of itself. Assume V is the old verifier and we want to build the parallel repetitive verifier W from it. W considers k independent runs of V, then waits for all these copies to ask for their first query, and responds to them all at once. Then it sends the answers all together and waits for all of them to send the second query and answers all at once, and so on. This would require a bigger*

alphabet. If the alphabet for V has been R, W would need R^k alphabet, because it has to receive and send k queries all at once. This means the alphabet size is going to be bigger, however this trade off is sometimes acceptable, for its effect on soundness and the fact that it does not increase number of queries we need, as opposed to sequential repetition.

We will see more about parallel repetition and its equivalent form, 2-Prover 1-Round Games in the last section. Also for more on this we refer to [Raz, 1998].

27.9 Reducing Number of Queries

Theorem 27.9.1 (Hastad's 3-bit PCP) *$NP = PCP_{1-\epsilon,\frac{1}{2}+\epsilon}(O(logn), 3)$ for all constants $\epsilon < 0$. Moreover, the PCP verifier for NP operates by choosing three positions i_1, i_2, i_3 of the proof and a bit b according o some distribution, reading the three values $y_{i_1}, y_{i_2}, y_{i_3}$ and accepting if and only if $y_{i_1} \oplus y_{i_2} \oplus y_{i_3} = b$.*

The proof of this theorem is removed due to complexity [Håstad, 2001b]. Here note that we already had proved a good PCP with soundness of $1 - \frac{1}{n^{O(1)}}$, but the soundness in Hastad's 3-bit PCP is close to $\frac{1}{2}$ which is a big improvement and justifies how this theorem is more complex than the former one. Moreover, a linear structure can be seen in this theorem which calls for definition of the following problem:

Definition 27.9.2 (MAX-3LIN) *Given a system of linear equality constraints over $GF(2)$ where each constraint involves exactly three variables, find a solution that maximizes the number of satisfied constraints.*

Theorem 27.9.3 *The following two statements are equivalent.*

1. *Hastad's 3-bit PCP.*
2. *For all constant $\epsilon > 0$, there is a reduction from 3SAT (or any NP-complete problem) to MAX-3LIN that maps satisfiable formulas to MAX-3LIN instances where a $1 - \epsilon$ fraction of the constraints can be satisfied simultaneously, and maps unsatisfiable formulas to MAX-3LIN instances where no assignment satisfies more than a $\frac{1}{2} + \epsilon$ fraction of constraints.*

Proof: First assuming 2, we prove 1.

Fix a value ϵ. By 2 we know that such described reduction exists. Consider system of $GF(2)$ formed from φ, and an assignment to its variables. The verifier performs as follows: It picks a random constraint, queries 3 bits of its proof, and accepts if and only if the constraint is satisfied.

Now we will have two cases:

- If φ is satisfiable, then there is some assignment for system satisfying at least $(1 - \epsilon)$ fraction of constraints. Prover picks any of such constraints with probability at least $1 - \epsilon$.
- If φ is not satisfiable, then any assignment violates at least $\frac{1}{2} - \epsilon$ fraction of constraints, and so it passes with probability at most $\frac{1}{2} + \epsilon$.

This completes this part of the proof.

Now assume 1 is true, we will prove 2.

Consider verifier V for 3SAT from $PCP_{1-\epsilon,\frac{1}{2}+\epsilon}(O(logn, 3))$. We will build a 3LIN instant as follows:

- variables: All possible bits of the proof
- constraints: For all random strings R formed of $O(logn)$ random bits:
  - Consider 3-bit queries for each R.
  - Consider variable assignments which make V accept.
  - $\rightarrow$ collection of all these constraints forms a MAX-3LIN.

Now if φ is satisfiable, then there exists a proof such that at least $1-\epsilon$ fraction of choices of R would lead to accept, so we will assign these same values to the variables to have the verifier accept in this case.

If φ is not satisfiable, then each assignment yields accept for at most $\frac{1}{2}+\epsilon$ fraction of R strings, so no assignment to the variables of the system satisfies more than a $\frac{1}{2}+\epsilon$ fraction of the constraints.

■

Remark 27.9.4 *In Hastad's theorem, note that having a completeness of 1 would mean P=NP. This is because if so then by the above theorem all NP problems could be reduced to finding out consistency of the set of constraint of the resulting 3LIN instance, and this is doable in polynomial time, for example by Gaussian Elimination.*

Now we can use the above theorem to prove inapproximability of MAX 3SAT.

Theorem 27.9.5 *For all constant $\epsilon' > 0$, MAX-3SAT is NP-hard to approximate within a factor of $\frac{7}{8}+\epsilon'$.*

Remark 27.9.6 *Proof of the above theorem follows the same steps as 27.7.2.*

Remark 27.9.7 *Note that the above theorem together with 27.5.5 would mean the best approximation factor possible for MAX-3SAT problem is $\frac{7}{8}$ unless P=NP.*

27.10 Unique Games Conjecture

27.10.1 Formulation 1: Unique label covering

Unique Games Conjecture has several equivalent formulations. We first consider the formulation based on unique label covering problem.

Definition 27.10.1 (Unique label covering problem) *Unique label covering problem takes inputs graph $G = (V, E)$, a set of k labels L, and bijective functions $\pi_{uv} : L \rightarrow L$ for every $(u, v) \in E$, and output a label assignment $l : V \rightarrow L$.*

Here, each function π_{uv} represents a constraint, which is satisfied if and only if $\pi_{uv}(l(u)) = l(v)$. The goal is to find label assignments that satisfy as much constraints as possible.

For inputs $\mathcal{L} = (G, L, \{\pi_{uv}\})$, it's possible to satisfy all constraints $\{\pi_{uv}\}$, so we say that $\mathcal{L}$ is satisfiable in this case; for some other inputs, at most some fraction of constraints can be satisfied simultaneously. We define the value of an unique label cover instance $\mathcal{L} = (G, L, \{\pi_{uv}\})$ as the maximum fraction of constraints one can satisfy in $\mathcal{L}$ denoted as $value(\mathcal{L})$.

One natural decision problem is to determine whether it is possible to satisfy all constraints, i.e. whether $value(\mathcal{L})$ is 1. It turns out that this could be solved in polynomial time. We provide the following algorithm to decide this problem: (assume the graph is connected without the loss of generality)

Algorithm 1 Decide the satisfiability of Unique label covering

Input: graph $\mathcal{L} = ((V, E), L, \{\pi_{uv}\})$,

Pick a vertex $v \in V$
for $c \in L$ **do**
    Reinitialize l, tentatively let $l(v) = c$
    Propagate labels by having $l(w) = \pi_{uw}(l(u))$ for $(u, w) \in E$
    if l satisfies all constraints π_{uv} **then**
        return Satisfiable
return Not satisfiable

One less clear problem is to efficiently decide whether an instance is almost satisfiable or far from satisfiable.

Definition 27.10.2 *Let $ULC(\delta)$ be the problem of deciding whether an instance $\mathcal{L}$ of label cover with unique constraints has $value(\mathcal{L}) \geq 1 - \delta$ or $value(\mathcal{L}) \leq \delta$. The "uniqueness" refers to the fact that for any edge, the label of one endpoint uniquely defines the valid label of the other endpoint.*

Unique Games Conjecture(UGC) proposes that $ULC(\delta)$ is NP-hard; formally, for sufficiently small constant δ, there exists a constant k depending on δ such that $ULC(\delta)$ with k labels is NP-hard.

The conjecture would have many implications if it holds true. For instance, while PCP implies that it is NP-hard to approximate MAX-CUT better than $\frac{16}{17} \approx 0.941$ [Håstad, 2001b], if UGC is valid, it would upper bound the approximation ratio to $\min_{-1<\rho<0} \frac{2}{\rho} \frac{arccos(\rho)}{1-\rho} \approx 0.87856$. Thus, unless $P = NP$, no polynomial time can achieve better approximation ratio than the one achieved by Goemans-Williamson algorithm using semi-definite programming [Goemans and Williamson, 1995]. In 2008, [Raghavendra, 2008] shows that a more general implication of UGC: If UGC is true, then the best approximation ratio for every constraint satisfication problem is given by some semi-definite programming algorithm. With a wide range of interesting implications, UGC has thus been actively studied since it was first purposed.

27.10.2 Formulation 2: 2-prover 1-round game

One may wonder why the conjecture is named "Unique Games." The name comes from another formulation of the conjecture based on the 2-prover 1-round game (2P1R game) .

Definition 27.10.3 (2-prover 1-round game (2P1R game)) *In a 2P1R game, there are two*

provers, P_1 and P_2 claiming they have a proof for a decision problem, and a polynomial time verifier V querying P_1 and P_2 separately. P_1 and P_2 may collude beforehand but do not communicate during the time responding to V's query. Then, each P_i picks an answer in response to query it gets, without knowledge of the query to the other prover or the other prover's answer. Based on the answers given by two provers, V decides whether to accept or reject.

Let's consider a more concrete example [mad,] :probabilistic verifier V wants to know whether a given graph G is 2-colorable. A proof that could convince V is a valid 2-coloring of G, and provers P_1, P_2 want to convince V that they know a valid 2-coloring - they may be lying. Suppose G is public but so large that V does not want to search a proof itself. V instead checks whether P_1 and P_2 are lying by asking P_i the color of some nodes in the 2-coloring they claim to know. Here the size of answer set is 2 because there are 2 colors to use. V would catch P_1 and P_2 lying if they disagree on the coloring of one node or give same color for two nodes in an edge. P_1, P_2 could collude beforehand but could not adapt its own answer based on other prover's answer. Thus, whether the graph is 2-colorable or not, their best strategy is to agree upon a coloring that has least number of edges with same color endpoints and answer the color of nodes queried by V truthfully.

Definition 27.10.4 (Unique Game) *A unique game is a 2P1R game with extra constraints on pairs of queries such that any pair of queries and the answer of one prover uniquely determines one acceptable answer by the other prover.*

Informally, these uniqueness constraints correspond to constraints $l(w) = \pi_{uw}(l(u))$ in the label covering formulation, where the label of a node uniquely determines a label for each of its neighbors that does not violate constraints. It is natural to reduce an instance of above-mentioned 2P1R game verifying 2-colorability of $G = (V, E)$ to a label covering instance $=((V, E'), L, \pi)$, where

$$
\begin{aligned}
E &= \{(u, v) \mid V \text{ might query } u, v \text{ from } P_1 \text{ and } P_2 \text{ respectively } \} \\
L &= \{\text{color 1, color 2}\} \\
\pi_{uv}(c) &= \begin{cases} \text{color 1} & \text{if } c = \text{color 2 and } u \neq v \\ \text{color 2} & \text{if } c = \text{color 1 and } u \neq v \\ c & \text{if } u = v \end{cases}
\end{aligned}
$$

Thus, the probability of V accepting a proof is reduced to the fraction of satisfied constraints. Analogously, we define the value of unique game as follows:

Definition 27.10.5 *Value of a unique game G is the probability that V accepts when prover P_1, P_2 are playing optimally, i.e.,*

$$\max_{P_1, P_2} \Pr(V \text{ accepts})$$

Then, **Unique games conjecture** can also be formulated as follows: for any sufficiently small constant $\delta > 0$, there exists k depending on δ, such that it's NP-hard decide whether a unique game with answer set size k has value at least $1 - \delta$ or at most δ. This is the formulation appeared in [Khot, 2002] original paper raising the conjecture.

27.10.3 One application: tighter inapproximibility bound of Max-cut

The analysis is based on the following theorem. We omitted its complicated proof using long code and boolean Fourier analysis. Please refer to [Khot, 2002] for more details.

Theorem 27.10.6 *Let $MaxCut(G)$ denote the fraction of edges of G in its max cut. For every $\rho \in (-1, 0)$ and $\epsilon > 0$, there exists δ and a polynomial-time algorithm to reduce an instance $\mathcal{L}$ of $ULC(\delta)$ to an instance $G = (V, E)$ of MAX-CUT such that*

- *When $value(\mathcal{L}) \geq 1 - \delta$, $MaxCut(G) \geq \frac{1-\rho}{2} - \epsilon$;*
- *When $value(\mathcal{L}) \leq \delta$, $MaxCut(G) \leq \frac{1}{\pi} arccos(\rho) + \epsilon$*

This theorem does not need to assume unique games conjecture. If we in addition assume Unique Games Conjecture, then we have the following result:

Theorem 27.10.7 *Assuming Unique Games Conjecture, it is NP-hard to approximate MAX-CUT to within any factor larger than*

$$\alpha = \min_{-1 \leq \rho \leq 1} \frac{2}{\pi} \frac{arccos(\rho)}{1 - \rho}$$

Proof Sketch: Assuming Unique Games Conjecture, for any δ, there exists k depending on δ that $ULC(\delta)$ is NP-hard. Let

$$\rho^* = argmin_{-1 \leq \rho \leq 1} \frac{2}{\pi} \frac{arccos(\rho)}{1 - \rho}$$

If there exists algorithm $\mathcal{A}$ that can approximates MAX-CUT within any factor $\alpha + \epsilon$ then it can be used to decide whether $MaxCut(G) \geq \frac{1-\rho^*}{2}$ or $MaxCut(G) < \frac{1}{\pi} arccos(\rho^*) + \rho^* \epsilon$ because

$$\frac{arccos(\rho^*) + \frac{1-\rho^*}{2}\epsilon}{\frac{1-\rho^*}{2}} = \alpha + \epsilon,$$

and by Theorem 3.2, the existence of polynomial time approximation algorithm implies that decision problem is not NP-hard, and then the theorem above, $ULC(\delta)$ is not NP-hard, contradicting the assumption that Unique Games Conjecture is valid.

27.10.4 Recent progress

Theorists hold divided opinions about UGC. In recent years, there are progresses on Unique Games Conjecture from both sides: evidence that supports it might hold and evidence that there might be efficient algorithm to decide $ULC(\delta)$. For the history of recent breakthrough, Quanta magazine has a nice discussion in [qua,].

One progress supporting UGC is that it is proved to be NP-hard to decide whether a 2-to-2 game has value at least $1 - \delta$ or at most δ. [Subhash et al., 2018] 2-to-2 games are set up almost the same as unique (2p1r) games except that any pairs of queries and answer by one prover determines exactly **two** possible answers of the another prover that V would accept. Informally, a unique

game can be viewed as a 2-to-2 game where for each constraint, 1 out of 2 choices that satisfies the constraint is discarded. Thus, the unique game is expected to have value $\frac{1}{2}(1-\delta)$ when its corresponding 2-to-2 game has value $1-\delta$, and have value $\frac{\delta}{2}$ when corresponding 2-to-2 game has value δ. Thus, the NP-hardness of $[\delta, 1-\delta]$-gap decision for 2-to-2 games implies the NP-hardness of $[\delta, (\frac{1}{2}-\delta]$-gap decision for unique games, which proves "half" of UGC and gives people confidence that UGC is valid too.

References

[qua,] First big steps toward proving the unique games conjecture. `https://www.quantamagazine.org/computer-scientists-close-in-on-unique-games-conjecture-proof-20180424/#comments`. Accessed: 2019-12-03.

[mad,] Information theory in computer science lecture 18. `http://people.seas.harvard.edu/~madhusudan/courses/Spring2016/scribe/lect18.pdf`. Accessed: 2019-12-03.

[Arora et al., 1998] Arora, S., Lund, C., Motwani, R., Sudan, M., and Szegedy, M. (1998). Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555.

[Arora and Safra, 1998] Arora, S. and Safra, S. (1998). Probabilistic checking of proofs: A new characterization of np. *J. ACM*, 45(1):70–122.

[Goemans and Williamson, 1995] Goemans, M. X. and Williamson, D. P. (1995). Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42(6):1115–1145.

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

[Håstad, 2001b] Håstad, J. (2001b). Some optimal inapproximability results. *Journal of the ACM (JACM)*, 48(4):798–859.

[Khot, 2002] Khot, S. (2002). On the power of unique 2-prover 1-round games. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 767–775. ACM.

[Raghavendra, 2008] Raghavendra, P. (2008). Optimal algorithms and inapproximability results for every csp? In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 245–254. ACM.

[Raz, 1998] Raz, R. (1998). A parallel repetition theorem. *SIAM J. Comput.*, 27(3):763–803.

[Subhash et al., 2018] Subhash, K., Minzer, D., and Safra, M. (2018). Pseudorandom sets in grassmann graph have near-perfect expansion. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 592–601. IEEE.