CS 252, Lecture 9: Probabilistically Checkable Proofs and Hardness of Approximation

1 Introduction

In this lecture, we introduce the notion of Probabilistically Checkable Proofs (PCPs), and explain their connection to proving bounds on the existence of approximation algorithms.

Recall the complexity class **NP**: it is the set of languages whose membership can be efficiently certified. More formally, A language $L \subseteq \{0,1\}^*$ is in the class **NP** if there exists a deterministic poly-time verifier V such that for every string $x \in \{0,1\}^*$,

1. if $x \in L$, there exists a "proof" string y such that $|y|$ is poly($|x|$) and $V(x,y)$ accepts.
2. if $x \notin L$, then for every string y, $V(x,y)$ rejects.

For example, consider the problem 3-coloring of graphs. Here, given a string x containing the description of the graph G, if it is 3-colorable, we can certify this by simply presenting y to be the valid 3-coloring of the graph G. The verifier V simply reads x, y and checks if the coloring in y is a valid one for the graph G. Note that if the graph is not 3-colorable, the verifier will reject every proof string y. The first part, where we accept all the valid strings, is usually referred to as the *completeness* of the proof system. The second part, where we reject all the invalid strings, is instead called *soundness*.

An issue with the above proof system is that the verifier needs to read the proof entirely. Can we verify the proof without reading the whole proof? i.e. can we "spot check" the proof? A priori it may seem impossible - we might be able to encode an incorrect proof that has error only in one bit. For example, in the graph coloring case, even though the graph is not 3-colorable, there might be a 3-coloring that violates only one edge in the graph, and if we don't read the colors assigned to every vertex, we might incorrectly accept the proof.

Furthermore, if we require the verifier to be a deterministic algorithm, checking fewer bits of the proof essentially means asking if there exists a short proof for every language in **NP**, which is unlikely. Thus, we restrict ourselves to randomized verifier. However, the previous issue remains: if we have a coloring that violates only one edge, we can't find it by sampling only a few edges. Hence, the goal is to develop a notion of robust proof system such that true proofs are always accepted and for false proofs, there are bugs everywhere. The main question is the following: is there a proof system where we can query only a few bits in the proof, and decide whether it is correct or not? The famous PCP Theorem answers this question in affirmative.

Theorem 1. *(PCP Theorem) There exists an absolute constant q such that the following holds: for every language L in **NP**, there exists a randomized poly-time verifier V such that for every string $x \in \{0,1\}^*$,*

1. *If $x \in L$, there exists y such that $|y|$ is poly($|x|$) such that $V(x,y)$ accepts.*
2. *If $x \notin L$, for every string y, $V(x,y)$ rejects with probability at least $\frac{1}{3}$.*

Furthermore, the verifier V only reads q (randomly chosen) bits in the proof.

Note that the error probability $\frac{1}{3}$ can be replaced with any constant < 1 as we can always amplify the probability by running the verifier algorithm multiple times. Also, the constant q is independent of $|x|$ or $|y|$! In fact, we can take q to be 3 in the above formulation i.e. to get $\frac{1}{3}$ rejection probability.

PCP Theorem is widely considered as one of the key results in theoretical computer science. It was originally proved in 1992 [AS92, ALM+92]. The original proof is quite involved, and a new insightful proof was presented in 2005 [Din07]. Even though it seems to be a statement about just interactive proofs, it has widespread applications in many other areas. One of the main connections is in proving impossibility results on approximation algorithms.

2 Connection to hardness of approximation

Recall that 3-SAT is NP-complete. That is, given a satisfiable formula, no polynomial time algorithm can find a satisfying assignment. Can we find an approximation algorithm - one that given any satisfying formula, outputs an assignment that satisfies 99% of the clauses? Simply picking the assignment at random satisfies $\frac{7}{8}$ fraction of the constraints. However, it seems unlikely that there exists an algorithm that can find an assignment satisfying 99% of clauses. How do we prove such a hardness? Note that proving such a thing unconditionally would imply $\mathbf{P} \neq \mathbf{NP}$, which we are quite far from proving. Instead, we prove that approximating these is **NP**-hard as well.

The way we achieve this is by so called "gap preserving" reductions. For example, we are aiming for the following type of reduction: given an instance C of CIRCUIT-SAT problem[1], we output a 3-SAT instance ϕ such that

1. If C is satisfiable, ϕ is satisfiable as well.
2. If C is not satisfiable, there is no assignment satisfying 99% of constraints in ϕ.

Recall that CIRCUIT-SAT problem is **NP**-complete. Thus, if we indeed have a reduction of the above kind, we can prove that finding an assignment satisfying 99% of a satisfiable 3-SAT instance is **NP**-hard. There is a very simple way to reduce CIRCUIT-SAT to a 3-SAT instance: for every gate, we assign a variable, and enforce the condition of the gate using 3-SAT clauses. For example, if we have an AND gate, and the two inputs are labeled as g_1, g_2 and the output is labeled as g_3, we can write this as $(g_1 \vee g_2 \vee \overline{g_3}) \wedge (g_1 \vee \overline{g_2} \vee \overline{g_3}) \wedge (\overline{g_1} \vee g_2 \vee \overline{g_3}) \vee (\overline{g_1} \vee \overline{g_2} \vee g_3)$.

[1] In the CIRCUIT-SAT problem, we are given a Boolean circuit containing AND, OR, NOT gates, and the goal is to determine if there is an assignment to the input gates that outputs True.

However, the above reduction may not be gap preserving i.e. if the input circuit C is not satisfiable, there is no guarantee on the number of clauses violated in ϕ. We only know that ϕ is not satisfiable. However, can we come up with a clever reduction that is in fact gap preserving? Turns out that this question is equivalent to the PCP Theorem!

Theorem 2. *(PCP Theorem, Inapproximability version) There is an absolute constant $c < 1$ such that the following holds: there is a polynomial time algorithm that outputs a 3-SAT formula ϕ given a Boolean circuit C such that*

1. *If C is satisfiable, ϕ is satisfiable as well.*
2. *If C is not satisfiable, no assignment to the variables can satisfy more than c fraction of constraints in ϕ.*

Theorem 2 is equivalent to Theorem 1. We only prove one direction of the implication in this lecture: suppose that Theorem 2 holds. We will prove that a probabilitically checkable proof for **NP** exists. First, since CIRCUIT-SAT is **NP**-complete, if we show the existence of probabilistic verifier for CIRCUIT-SAT, then we have shown the existence for every language in **NP**. (Why? - Do try this as an exercise.) Given an instance C of a CIRCUIT-SAT, our goal is to output a proof of satisfiability/unsatisfiability that can be verified by checking only a few bits. First, using Theorem 2, we find the 3-SAT formula ϕ and the proof string y in our proof system is the formula ϕ together with a purported satisfiable assignment σ to the variables in ϕ that satisfies all the clauses. The verifier then randomly picks a clause and checks if the assignment given satisfies it. We claim that it is a valid Probabilistically Checkable Proof system. Note that the verifier only reads constant many bits from the proof.

1. Completeness: Suppose that C is satisfiable. Then, ϕ is satisfiable as well, and thus the assignment σ satisfies all the clauses in ϕ. Thus, the Verifier outputs yes.
2. Soundness: Suppose that C is not satisfiable. Then, at least $1 - c$ fraction of clauses in ϕ are not satisfied in any assignment to the variables in ϕ. Thus, given any proof string y, the verifier rejects the proof with probability at least $1 - c$.

Now, we can repeat the same verifier algorithm many times, each time independently picking a random clause and checking if the purported assignment satisfies that clause. If we repeat t times this way, the probability that the verifier accepts an incorrect proof is c^t, which can be made as small a constant as possible, since c is a constant as well. The downside of this is that we increase the number of bits queried by the algorithm t times. But it is okay since in Theorem 1, we only claimed that we can read a constant number of bits to ensure that the probability of accepting an incorrect proof is at most $\frac{2}{3}$.

3 Applications

We prove a lot of problems are **NP**-hard by reducing 3-SAT to them. Some examples are graph coloring, maximum independent set in graph, knapsack etc. Similarly, once we have hardness of approximation for

3-SAT, we can use these reductions (and many new ideas!) to prove downstream hardness of approximation results. We demonstrate a concrete example here: Consider the reduction from 3-SAT to CLIQUE[2]. We have a reduction that given a 3-SAT formula ϕ with m clauses, outputs a graph G such that:

1. If ϕ is satisfiable, then G has a clique of size m.
2. Else, there is no clique of size m in G.

If we inspect the reduction, we can actually infer that the maximum clique in G is equal to the maximum number of clauses satisfied by an assignment to the variables. From Theorem 1, we can in fact prove that not only is CLIQUE **NP**-hard, approximating the maximum CLIQUE is **NP** hard as well. That is, there exists a constant $c < 1$ such that given a graph G and a constant k, it is **NP**-hard to distinguish between the following:

1. Completeness: G has a clique of size k.
2. Soundness: There is no clique of size kc in G.

We can in fact amplify this to show that approximating CLIQUE to *any constant* is **NP**-hard.

References

[ALM+92] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and hardness of approximation problems. In *33rd Annual Symposium on Foundations of Computer Science, Pittsburgh, Pennsylvania, USA, 24-27 October 1992*, pages 14–23, 1992.

[AS92] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs; A new characterization of NP. In *33rd Annual Symposium on Foundations of Computer Science, Pittsburgh, Pennsylvania, USA, 24-27 October 1992*, pages 2–13, 1992.

[Din07] Irit Dinur. The PCP theorem by gap amplification. *J. ACM*, 54(3):12, 2007.

[2]It is the same reduction done in 251. See these slides