18.408 Topics in Theoretical Computer Science Fall 2022
Lecture 1

Dor Minzer

1 Introduction

The main topic of this course is Probabilistically Checkable Proofs (PCP), an active area of TCS which emerged in the early 90's and has had a tremendous impact on many other areas in TCS. One of the most prominent applications of PCPs is to hardness of approximation, wherein the theory of NP-hardness is extended to approximation problems. In an undergraduate complexity course, one often sees NP-hardness results from the 70's asserting that many famous combinatorial optimization problems (3SAT, Clique, Set-Cover, etc.) are NP-hard. It turns out that even approximating the optimal solution to these problems is still NP-hard, but to prove such results one needs an analog of the Cook-Levin theorem for approximation problems. This result is called the PCP theorem, and the first half of the course will be focused on the proof of this result. Aside from that, one then needs to develop Karp-type reductions that respect the notion of approximation to leverage the PCP theorem to prove NP-hardness results for other optimization problems (similar to the classical theory of NP-completeness). The second half of the course will focus on that.

So, what is a PCP? To begin this discussion, let us first view the standard NP-hardness of the 3-SAT problem in several equivalent ways. Recall that a 3-CNF formula is a formula of the form $\phi(x_1, \ldots, x_n) = C_1 \wedge C_2 \wedge \ldots \cdot C_m$, where each C_i is a clause of the form $(\alpha \vee \beta \vee \gamma)$ where each one of α, β, γ is a literal (i.e. one of the input variables x_i or its negation). The computational problem 3-SAT is the problem in which one is given, as an input, a 3-CNF formula $\phi(x_1, \ldots, x_n)$, and the goal is to decide if it is satisfiable or not. Namely, to decide if there is an assignment $A\colon \{x_1, \ldots, x_n\} \to \{0, 1\}$ that satisfies each one of the clauses of ϕ. Thus, in terms of languages, we write

$$\text{3-SAT} = \{\, \phi \mid \phi \text{ is a satisfiable 3-CNF formula}\} \,.$$

The Cook-Levin theorem is one of the most basic results in complexity theory, asserting that 3-SAT is NP-hard. That is, if 3-SAT can be solved in polynomial time, then P $=$ NP. Another formulation of this theorem (which is a bit stronger), is that given any language $L \in$ NP, checking membership in L can be efficiently reduced to checking membership in 3-SAT. Namely, there exists a reduction map $f\colon \{0, 1\}^* \to \{0, 1\}^*$ such that: if $z \in L$, then $f(z) = \phi_z \in$ 3-SAT, and if $z \notin L$, then $f(z) = \phi_z \notin$ 3-SAT. In other words, an instance z to problem L can be encoded as a Boolean formula $\phi_z = f(z)$ so that "witnesses" to satisfiability of ϕ_z encode witnesses to the instance z being in L.

What is so special about 3-SAT, and what makes this useful? To illustrate that and draw our attention to several parameters of interest, we consider an NP-verifier for the 3-SAT language. Recall that an NP-verifier is a polynomial time Turing Machine that gets as input a formula ϕ, and a witness w. We say the verifier accepts ϕ if there is some witness w that makes it accept, and we say the verifier rejects if it rejects no matter what witness w the verifier is given. In the case of 3-SAT, the verifier is quite simple; the witness w is simply a supposed satisfying assignment for ϕ, and the verifier checks that it indeed satisfies the formula ϕ.

1.1 Probabilistic verifiers

So far, there is nothing probabilistic about this, so we adopt a somewhat different view of this classical verifier. The witness w is still a supposed satisfying assignment, but now the verifier picks a random clause C_i from ϕ and checks whether it is satisfied or not. Below we consider a few notions that will be important for us throughout this course, and explain them through this example.

1. **Completeness:** the completeness parameter measures what is the probability the verifier accepts provided the input ϕ is in the language, and the verifier is given a correct witness. Throughout most of this course, we will be discussing perfect completeness, i.e. the case this probability is 1; this is indeed the case for the verifier above.

2. **Soundness:** the soundness parameter measures the probability the verifier accepts provided the input ϕ is not the language, and the verifier is given any witness (typically thought of the witness that "fools" the verifier the most). In the above example, we know that if ϕ is not satisfiable, then the assignment that w encodes cannot possibly satisfy all of the clauses of ϕ, hence there is some clause C_i that is not satisfied by it. The probability that the verifier picked that clause is $\frac{1}{m}$, and therefore the probability the verifier accepts is at most $1 - \frac{1}{m}$

3. **Locality/ number of queries:** this refers to the number of entries of the witness w the verifier makes access to. In this case, the verifier looks at a clause C_i containing 3 literals, hence has to read 3 locations from the witness w. Hence, we say that the number of queries of the verifier is 3.

4. **Alphabet size:** the alphabet size is the size of each one of the queries the verifier makes. In this case, the verifier just reads off 3 bits from w, hence its alphabet size is 2.

In light of this, we can say that the main features of the Cook-Levin theorem that make it so useful is that it shows that any language in NP admits a probabilistic verifier which is very local, making only 3 queries to the witness whose alphabet is of size 2. Indeed, given any language $L \in$ NP, the verifier would use the reduction f above to reduce an instance z of L to a formula $\phi_z = f(z)$, and think of its witness as an assignment to the formula ϕ_z. The main weakness of this theorem, in this language, is that the soundness guarantee is rather poor — the probability the verifier accepts an input not in the language is still close to 1.

One formulation of the PCP theorem in this language is the following counter-intuitive looking statement. There exists an absolute constant $\varepsilon > 0$, such that any language $L \in$ NP admits a probabilistic verifier V that has completeness 1, soundness at most $1 - \varepsilon$ that makes $O(1)$ queries and has alphabet size $O(1)$. Namely there is a verifier just like before that is able to "catch" cheating witnesses with much higher probability than before! Looking at the previous verifier, it is completely unclear how to do something like that (or if it is even possible), but as we will see in this course, it is possible (though it will take us time).

1.2 A combinatorial point of view of PCP

Next, we present an equivalent combinatorial view of PCP, which is somewhat easier to think about in the context of hardness of approximation. For parameters $0 \leqslant s < c \leqslant 1$, the computational problem gap-3-SAT$[c, s]$ is the promise problem wherein one is given, as an input, a 3-CNF formula ϕ; it is promised that either there exists an assignment to ϕ satisfying at least c fraction of the clauses in it, or that any assignment to ϕ satisfies at most s fraction of the clauses of ϕ. The goal in the problem gap-3-SAT$[c, s]$ is to distinguish between these two cases. That is, to solve gap-3-SAT$[c, s]$ one has to design a Turing Machine that accepts the former type of instances, and rejects the latter type of instances.

In this formulation, the Cook-Levin theorem asserts that gap-3-SAT $\left[1, 1 - \frac{1}{m}\right]$ is NP-hard, and the PCP theorem is equivalent to the following assertion:

Theorem 1.1. *There exists $\varepsilon > 0$ such that gap-3-SAT$[1, 1-\varepsilon]$ is NP-hard.*

Hence, one sees that in gap-3-SAT$[c, s]$, the parameters s and c correspond to the soundness and completeness of the verifier; the alphabet size and the number of queries are already apparent from the form of the definition of 3-SAT.

As we will see in the second half of this course, the above combinatorial view of PCP is useful when doing Karp-style reductions with the goal of showing hardness of approximation results. Indeed, you can prove (try this!) that if gap-3-SAT$[c, s]$ is NP-hard, then it is NP-hard to approximate the maximum number of clauses that can be satisfied in a formula ϕ within factor $\frac{s}{c}$. Hence, to get the best hardness of approximation results one wants to optimize the ratio between the soundness and the completeness parameter. The notion of gap problem carries with it more information though, regarding the "location" of the hardness.

Hence, the PCP theorem implies in particular that there is $\varepsilon > 0$ such that approximating the number of clauses that can be satisfied in a given 3-CNF formula within factor $1 - \varepsilon$ is NP-hard, and a natural question is what is the "right" ε in this result? Is there some $s \in (0, 1)$ such that approximating within factor $s + \delta$ is NP-hard but approximating within factor $s - \delta$ can be done in polynomial time?

1.3 PCP as a 2-Prover, 1-Round game

PCP can also be used as an interrogation technique. Consider the setting in which we have a computationally weak verifier V, and 2 very powerful provers. The provers try to convince the verifier V that some 3-CNF formula ϕ is satisfiable, but they physically sit in two different rooms with no ability of communicating with each other. Is there some scheme that allows the provers convince the verifier that ϕ is satisfiable if this is indeed the case? Can the provers convince the verifier that ϕ is satisfiable when in actuality it isn't? Of course, V can simply ask one of the provers for a whole assignment to the formula ϕ, and check all clauses are satisfied; V doesn't have time for that though. Another thing V can do is choose a clause C_i randomly, send it to one of the provers, and ask for an assignment to it. This though, fails miserably, as the prover can just cheat and just assign some value to the variables in C_i that satisfies this clause. What should V do then?

Well, V can sample a clause C_i and choose randomly one of the variables in it, say x_j. The verifier V can then send all of the variables of C_i to the first prover, and only the variable x_j to the second prover. The verifier expects to get, as answers from each one of the provers, assignments to the variables they received. Upon getting these answers, V checks that the given assignment satisfies C_i, and that the two provers assigned the same value to x_j (that is, that they are consistent).

The model of 2-prover-1-round protocols can be thought of more generally in the context of language L, not only for 3-SAT. In this formulation, the PCP theorem tells us that any language $L \in$ NP has a 2-prover-1-round protocol that has:

1. Completeness: that is, there are prover strategies that manage to convince V of "true statements" — namely if the protocol is ran on a common input z that is in L, then V always accept.

2. Soundness: that is, if the common input z is not in L, any prover strategy (even a malicious one) makes V accept with probability at most $1/3$.

3. Efficient: the verifier V is very efficient, using $O(\log n)$ randomness and reading only $O(1)$ bits from the answer of each prover.

1.4 Course overview

Since this is the first time this course is ran, the exact material we cover is yet to be determined; below is a tentative plan.

1. **Error-Correcting Codes:** We will begin the course by describing a much simpler, completely combinatorial analog of PCPs, which are also an important building block in PCP constructions. In particular, we will define linear error correcting codes and their various parameters, give examples (Reed-Solomon, Reed-Muller, Hadamard) and describe concatenation of error correcting codes. We will also discuss local testing and local correction of codes.

2. **The algebraic proof of the PCP theorem.** This part of the course will consist of several weeks, and in it we will prove the basic PCP theorem. The way the proof works is by first doing a reduction that gets a large gap between the completeness and soundness parameters, but as a result significantly increases the number of queries and the alphabet size. This will be relatively easy, and the bulk of the work then will be to reduce the number of queries and the alphabet size back to be $O(1)$. For that, we will present the sum-check protocol, proof composition, low-degree testing, aggregation of queries and the local-to-global phenomenon.

3. **Hardness Amplification and the Long-Code Framework.** Next, we will present the tools that are used in conjunction with the PCP theorem in order to establish hardness of approximation results. In particular, we will discuss the parallel repetition technique (an operation that improves the soundness of a PCP while keeping the number of queries the same) and the Long-Code framework. We will see several example of results that can be established this way. In particular, we will prove hardness of approximation results to popular combinatorial optimization problems such as Clique, Vertex-Cover and Linear-Equations over finite fields.

4. **Extreme Forms of the PCP theorem.** Next, we will discuss several improved forms of the PCP theorem that are conjectured to be true (yet are all open). By that, we often refer to pushing some feature of the PCP theorem to "the limit", and asking if there is a PCP construction that achieves that. These include size efficient PCPs, time efficient PCPs, the Sliding Scale Conjecture and the Unique-Games Conjecture.

5. **Advanced Topics.** Towards the end of the course we will discuss more advanced topics. These include the Unique-Games Conjecture (what it implies and the current state of affairs), sub constant soundness PCPs, optimal hardness of Clique and Vertex-Cover.

1.5 Applications of PCPs

PCPs have a wide array of applications throughout TCS (hardness of approximation, cryptography, interactive protocols and more), as well as some practical ones (most prominently in blockchains), and exploring such applications could be a good direction for a final project in the course. For most of the course though, we will focus on the applications of PCPs to hardness of approximation, and below we give a few examples of results we may see.

Linear equations over finite fields. An instance of the Max-3-Lin$_2$ problem consists of a set of variables $X = \{x_1, \ldots, x_n\}$ and a set of equations $E = \{e_1, \ldots, e_m\}$, wherein each equation $e \in E$ is of the form $x_i + x_j + x_k = b_{i,j,k}$ where $b_{i,j,k} \in \mathbb{F}_2$ is a constant; addition is performed over $(\mathbb{F}_2, + \pmod 2)$. Given an instance (X, E), the goal is to find an assignment to the variables $A\colon X \to \mathbb{F}_2$ that satisfies as many of the equations as possible. What is the complexity of Max-3-Lin$_2$?

Well, if we are given (X, E) that is promised to be fully satisfiable (i.e. that there is an assignment satisfying all of its equations), then we can find a satisfying assignment efficiently by using Gaussian Elimination. In gap notations, this means that gap-Max-3-Lin$_2[1, 1]$ can be solved in polynomial time. What if we relax the promise, and only say that (X, E) is $(1-\varepsilon)$-satisfiable, i.e. that there is an assignment satisfying $1-\varepsilon$ fraction of the equations? Note that one can always satisfy at least half of the equations by trying the "all 0" assignment and the "all 1 assignment". It turns out that beating this trivial algorithm is NP-hard:

Theorem 1.2 (Hastad). *For all $\varepsilon > 0$, gap-Max-3-Lin$_2[1-\varepsilon, 1/2+\varepsilon]$ is NP-hard.*

Minimum Vertex-Cover. A vertex cover in a graph $G = (V, E)$ is a set of vertices $C \subseteq V$ that contains at least one endpoint of each edge. The goal in the Vertex-Cover problem is to find, given a graph $G = (V, E)$, the smallest vertex cover in it; denote the fractional size of it by $\mathsf{VC}(G)$. As we will see, there is an easy 2-approximation algorithm, i.e. an algorithm that given G efficiently finds a set $C \subseteq V$ that is a vertex cover of G and has $|C| \leqslant 2\mathsf{VC}(G)\,|V|$. It is suspected, with strong evidence, that this is essentially the best one can do (and this is related to the Unique-Games Conjecture and all), but this result is not known. The best known result to date is:

Theorem 1.3. *For all $\varepsilon > 0$, approximating the minimum vertex-cover in a graph within factor $\sqrt{2}-\varepsilon$ is NP-hard. Moreover, the promise problem gap-Vertex-Cover$[1-\varepsilon, 1/\sqrt{2}+\varepsilon]$ is NP-hard.*

In words, the "moreover" part of the theorem states it is NP-hard to distinguish between the case that almost all of the vertices of the graph are needed to cover all edges, and the case in which at most $1/\sqrt{2}+o(1)$ fraction of them suffice.

Independent sets in graphs. An instance of the Independent-Set problem consists of a graph $G = (V, E)$, and the goal is to find the largest set of vertices $I \subset V$ that contains no edge from E; such sets are called independent sets. Suppose we are given a graph $G = (V, E)$ which is promised to contain an independent set of fractional size $\frac{1}{2}+\varepsilon$; what is the largest independent set we may find? Well, it turns out that the Vertex-Cover and the Independent-Set problems are very much related, and one can show that in this case, one can use the 2-approximation algorithm from above to find an independent set of fractioanl size 2ε in G. It is again suspected that this is essentially the best one can do (again related to the Unique-Games Conjecture), but the best known result to date is the following:

Theorem 1.4. *For all $\varepsilon > 0$, gap-Independent-Set$[1-1/\sqrt{2}-\varepsilon, \varepsilon]$ is NP-hard.*

In words, given a graph, it is NP-hard to distinguish between the cases it contains an independent set of fractional size $1-1/\sqrt{2}-\varepsilon$, and the case it doesn't even contain an independent set of fractional size ε.