Miranda Christ
Rohan Jha

PCP

CSCI 2951M
October 16, 2019

Introduction

We hope to provide some motivation and intuition for the proof of the PCP theorem. We first introduce the theorem itself, outline the proof, then examine the gap amplification and composition steps specifically.

Statement of theorem

We'll start by talking about verifiers in general terms. We consider a verifier V, a language L, an x that might be in L, and a corresponding proof p. Very informally, V does the following: (1) takes in some random bits; (2) looks at x; (3) chooses some bits to query from p; (4) decides the bit strings for which it will accept x; (5) queries the bits from p and either accepts of rejects x.

Now, we say that a language L is in PCP[r, q] if there's a verifier V, that takes in $O(r)$ random bits in (3) and reads $O(q)$ bits from the proof in (5), such that the following holds:

1. (Completeness) If $x \in L$, there's some proof p such that V will always accept x.
2. (Soundness) If $x \notin L$, there's no proof p for which V will accept x with more than $\frac{1}{2}$ probability.

Having defined PCP, we can cleanly state the PCP theorem, as follows.

Theorem (PCP 1)**.** *NP ⊂ PCP[log n, 1]*

In other words, for every language in NP, there's a verifier that needs only read a constant number of bits from a proof.

Equivalence with inapproximability

The PCP Theorem was found to be equivalent to statements about the inapproximability of some problems. Therefore, we'll prove the theorem by showing that one of these problems, constraint satisfication (CSP), is inapproximable.

We'll start by introducing the problem. Assume that we're given a set of variables V, some alphabet of values Σ that the variables can take, and a set of constraints $\mathcal{C}$ on k-tuples of variables. These constraints, for example, might take the form: (x, y) must be $(1, 2)$ or $(2, 3)$, where $x, y \in V$ and $1, 2, 3 \in \Sigma$. The CSP problem is to determine whether all of the constraints in $\mathcal{C}$ can be satisfied by an assignment of V to Σ. We note that this is NP-Hard (it's straightforward to reduce from k-colorability).

Now, to talk about the approximability of the problem, we define $UNSAT(\mathcal{C})$ as the minimum (over assignments) of the proportion of violated constraints in $\mathcal{C}$. For instance, if all the

constraints in $\mathcal{C}$ were satisfiable by some assignment, then $UNSAT(\mathcal{C})$ would be 0. We also define $UNSAT_\sigma(\mathcal{C})$ as the proportion of violated constraints with assignment σ.

We make the claim that the following statement is equivalent to the PCP theorem.

Theorem (PCP 2)**.** *It's NP-Hard to determine whether* $UNSAT(\mathcal{C}) = 0$ *or* $UNSAT(\mathcal{C}) \geq \frac{1}{2}$.

We'll discuss why PCP 1 is implied by PCP 2 (this is the direction we need for the proof).

Proof. We'll construct an appropriate verifier $\mathcal{V}$ for approximate constraint satisfaction (Approx-CSP), as described in the above theorem, in order to show that Approx-CSP is in PCP[log n, 1]. We assume the same notation as above, and we note that the proof P for some $\mathcal{C}$ is the assignment of variables. $\mathcal{V}$ (uniformly) samples a single constraint c from $\mathcal{C}$, then queries P for the assignment of the constant number of variables in V that are included in c. $\mathcal{V}$ accepts x if and only if c is satisfied. Let's make sure that $\mathcal{V}$ is complete and sound. If $UNSAT(\mathcal{C}) = 0$, then there's some assignment of variables (proof) such that all constraints are satisfied and V will always accept $\mathcal{C}$. If $UNSAT(\mathcal{C}) \geq \frac{1}{2}$, then the probability of an arbitrary constraint begin violated will be greater than $\frac{1}{2}$ for every assignment of variables (proof), and V will reject $\mathcal{C}$ with probability greater than $\frac{1}{2}$. Therefore, we conclude that Approx-CSP is in PCP[log n, 1], and because Approx-CSP is NP-Hard, this implies PCP 1. □

Again, the upshot of this discussion is that we can prove the PCP Theorem by showing that Approx-CSP is NP-Hard. We'll do this reducing an arbitrary instance $\mathcal{C}$ of CSP (which is NP-Hard) to an instance $\mathcal{C}'$ that can be solved by Approx-CSP. This is equivalent to transforming $\mathcal{C}$ to $\mathcal{C}'$ such that the following holds: if $UNSAT(\mathcal{C}) = 0$ then $UNSAT(\mathcal{C}') = 0$, and if $UNSAT(\mathcal{C}) \neq 0$ then $UNSAT(\mathcal{C}') \geq \frac{1}{2}$. This transformation is the *gap amplification*.

Outline of strategy

We first outline how to represent an instance of CSP with a graph – this is the representation we'll prefer. Given some CSP, we construct a graph G in which the vertices represent variables in V that take on values from our alphabet Σ, and edges correspond to constraints in $\mathcal{C}$ on the two variables at the endpoints.

We'll now give a very broad overview of the reduction.

1. **Preprocess** the graph into a "nicely structured" graph for the powering step. This involves making our graph into an expander, similar to the process in Reingold's paper.
2. **Amplify** the minimum fraction of unsatisfied constraints by powering the graph. This make vertices "care" about the value of vertices that are separated by several edges (whereas originally, we cared only about the values of vertices that were separated by one edge). Crucially, this increases the $UNSAT$ value of the constraint graph by a constant factor.

However, we note that the amplification makes the alphabet exponentially larger; the "values" in the constraints are now a series of values from the original alphabet. The increased alphabet size poses a problem because we wish to show that need only read a *constant* number of bits from the proof. However, if the alphabet grows each time we power the graph, and the number of powerings is logarithmic in the length of the proof, checking a constant number of variable assignments in the powered graph requires reading more than a constant number of bits. This motivates the next step.

3. **Compose** the powered graph with an assignment tester, essentially a smaller PCP. This makes the alphabet a constant size while only decreasing the $UNSAT$ value by a constant factor.

Gap amplification

In this step, we power the constraint graph G by some constant t, which yields G^t. We'll start by talking about the structure of the powered graph: the set of vertices is the same, and we have an edge between u and v in G^t if we can take a t-step walk from u to v in G. The alphabet and the constraints are a bit trickier. A vertex (variable) v in G^t takes an $O(d^t)$-tuple of values in Σ: one for itself, and the remaining for every vertex that can be reached with a $\lceil \frac{t}{2} \rceil$-step walk from v. We think of this latter group of values as v's opinions of its neighbors (we'll be more explicit about their role in a couple sentences). In this sense, the size of the alphabet is blown up considerably (to $|\Sigma|^{O(d^t)}$). We turn to the constraints. We recall that there's a one-to-one mapping from edges in G^t to t-step walks in G. We denote an edge in G^t by E, its endpoints by u and v, and the corresponding walk in G as $(e_1, e_2, \dots e_t)$, where each e_i is an edge in G. The constraints, now, become more 'powerful', in that they enforce that each constraint e_i on the walk in G is satisfied by u and v's opinions of the incident vertices to e_i *and* that the endpoints of the edge in G^t have the same 'opinion' about each vertex on the walk in G.

It's worth thinking about what powering is doing, intuitively. We consider some edge (u, v) in G that's violated by an assignment σ. Assuming the same assignment, this violated edge in G might induce others to be violated in G^t, namely those incident to vertices with opinions about u and v that violate (u, v). We like Paul's metaphor that if u and v are squabbling neighbors in G, then the whole neighborhood is involved in the conflict in G^t.

We'll now state the main lemma about gap amplification and give some intuition about the proof.

Lemma (Gap amplification)**.** *Let λ be a constant with $0 < \lambda < 1$. If G is d-regular, with a self loop on each vertex, and $\lambda(G) \leq \lambda$, then the following holds: $UNSAT(G^t) \geq O(\sqrt{t}) \cdot min(UNSAT(G), \frac{1}{t})$.*

We let σ' be the assignment in G^t that minimizes $UNSAT(G^t)$. In order to relate $UNSAT(G^t)$ with $UNSAT(G)$, we need to relate σ' with an assignment in G because $UNSAT_{\sigma'}(G^t) =$

$UNSAT(G^t)$. Note that we get to pick an assignment σ in G because $UNSAT(G) \leq UNSAT_\sigma(G)$ for any σ. With this in mind, we let σ be the assignment in G where the value of each vertex is the majority opinion of its neighbors in G^t. In broad strokes, this σ makes it easier to analyze the opinions of a vertex's neighbors in G^t. Now, putting everything together, we need only show that $UNSAT_{\sigma'}(G^t) > O(\sqrt{t}) \cdot UNSAT_\sigma(G)$.

We'll just give the intuition. An edge from (u, v) E in G^t will be violated if the following holds: (1) some (u', v') is violated in the corresponding walk in G, (2) u's opinion of u' matches u''s value, and (3) v's opinion of v' matches v''s value. We define this as E being 'hit' by e, and we let N_E be the expected number of hits. We'll proceed by analyzing how often E will be hit (one is enough for E to be violated). There are other cases in which E will be violated, but this turns out to be enough for the analysis.

We need to show that the probability of a hit in G^t is fairly high, relative to the probability of an edge in G being violated. We achieve this by constructing a Chebyshev-like bound on the probability, using the first and second moments of N (this is comparable to showing that N has a high mean and low variance). We consider some vertex v that's incident to a violated edge in G. To estimate the first moment, we use the fact that the v's neighbors are 'likely' to have the same value as v in G^t because v's value was assigned by a majority vote. And to estimate the second moment, we use the fact that G's expansion properties make it unlikely for walks to remain on violated edges. If this were not the case, then N might be much higher for some edges in G^t than others, which means, intuitively, that the variance of N would be higher.

Alphabet reduction

Recall that in this step, we compose the constraint graph with a "small PCP", meaning we turn each constraint into a decision problem and use a PCP to check whether a variable assignment satisfies it. Observe how this would help us: a constraint over the larger alphabet in the powered graph is essentially a Boolean circuit, a decision problem that PCP can solve. Assuming PCP works here, we can read a constant number of bits from a solution (an assignment of variables) to this Boolean circuit and determine whether the constraints are satisfied. We call the "small PCP" an assignment tester:

Definition (Assignment Tester)**.** An assignment tester with alphabet Σ_0 and rejection probability $\epsilon > 0$ is an algorithm $\mathcal{P}$ whose input is a circuit Φ over Boolean variables X, and whose output is a constraint graph $G = \langle (V, E), \Sigma_0, \mathcal{C} \rangle$ such that $V \supset X$, and such that the following hold. Let $V' = V \setminus X$, and let $a : X \to \{0, 1\}$ be an assignment.

- (Completeness) If $a \in \text{SAT}(\Phi)$, there exists $b : V' \to \Sigma_0$ such that $UNSAT_{a \cup b}(G) = 0$
- (Soundness) If $a \notin \text{SAT}(\Phi)$, then for all $b : V' \to \Sigma_0$, $UNSAT_{a \cup b}(G) \geq \epsilon \cdot \text{rdist}(a, \text{SAT}(\Phi))$.

We can make a few observations from the definition, first that the set X of variables in the boolean expression is a subset of the set V of nodes in the resulting constraint graph, so we are creating more variables. Since we care about examining a constant number of constraints, each of which contains q variables (since we use q-ary constraints), increasing the number of variables does not make our job harder. Most importantly, we get a new alphabet Σ_0, which we claim can be of constant size.

Using an assignment tester should feel a bit circular, since it is nearly the same as the PCP construction we're trying to prove works. The key here is that while in our large construction the number of constraints is limited, the constraints we use assignment testers for are constant size, and therefore the assignment testers can have any number of their own constraints without hurting efficiency.

Thus the overall tradeoff is decreasing the alphabet size, while increasing the number of variables and the number of constraints.

A notable difference in the assignment tester compared to our overall construction is the $UNSAT$ value of an invalid variable assignment. In the assignment tester, the $UNSAT$ value is at least $\epsilon \cdot$ rdist(a, SAT(Φ)) as opposed to $\frac{1}{2}$. This is a result of looking at the constraints very locally, which doesn't ensure that a given assignment satisfies all constraints *simultaneously*. For example, variable v could take on value a_1 in constraint c_1 but value a_2 in constraint c_2 and satisfy both.

To remedy this, we want to change the space of assignments we consider so valid assignments are far from each other in terms of relative distance. Then two assignments differing by one variable value cannot both be valid, and we ensure that a single variable does not take on different values in different constraints. Achieving this is nontrivial, and we won't go into depth here. We can verify our intuition with the definition of the assignment tester, though, by seeing that increasing the relative distance between an invalid assignment and the closest valid assignment increases the $UNSAT$ value. This means the number of constraints satisfied by inconsistently valued variables is reduced as desired.