ECS 189A Sublinear Algorithms for Big Data **Fall 2024**

Lecture 9: The PCP Theorem

Lecturer: Jasper Lee *Scribe: Kim Calabrese*

1 Preliminaries & Complexity Theory Speedrun

In previous lectures, we were interested in property testing: given a set of objects $\mathcal{C}$, how efficiently can we decide the contents of some subset $\mathcal{P}$. The objects studied in complexity theory are similar. The class of objects we study are binary *strings* over the "alphabet" $\{0,1\}$. We denote the set of all binary strings of arbitrary length as $\mathscr{L} = \{0,1\}^*$.

A *language* $L \subseteq \mathscr{L}$ is a set of strings. In the context of property testing, this can be considered a property. Given a language L, the *decision problem* for L is given $x \in \mathscr{L}$, decide if $x \in L$ or $x \notin L$. We often think of the language L as the decision problem itself, and will sometimes refer to L as a "problem". If an algorithm A solves the decision problem for L, that is A halts and outputs 1 (yes) when $x \in L$ and 0 (no) when $x \notin L$, we say that A *decides* L. We refer to a set of languages as a *class*. A particularly important class is the class of languages decidable in polynomial time.

Definition 9.1 (The class **P**)**.** A language L is contained in **P** if and only if there exists a deterministic algorithm that decides L, which always halts in time $\text{poly}(|x|)$, where x is the input.

A difference between this (more standard) setting and property testing (which we have been focusing in class) is that here we are always allowed to read the entire input, and then perform our computations. Some examples of problems in **P** are

- Bipartiteness of a graph.
- Existence of an Eulerian path in a graph.
- Deciding if two vertices are connected in a graph.

Another important class is the class of problems verifiable in polynomial time.

Definition 9.2 (The class **NP**)**.** A language L is in **NP** if there is a deterministic algorithm $V(x,w)$ and polynomials p and q such that

1. $V(x,w)$ halts in time $p(|x| + |w|)$.
2. For all $x \in L$, there exists $w \in \mathscr{L}$ of length $q(|x|)$ such that $V(x,w) = 1$.
3. For all $x \notin L$, each $w \in \mathscr{L}$ satisfies $V(x,w) = 0$.

We refer to V as a *verifier* for L and w as a witness for $x \in L$. Intuitively, a string x is contained in L if and only if there a "short" string w that provides proof of x's membership in L which can be quickly verified by V. Some examples of languages in **NP** are

- 3-SAT (Given a boolean formula φ in conjunctive normal form, does φ have a satisfying assignment?)

- 3-Colorability of a graph (Given a graph G, can you label the nodes with a color in {0,1,2} such that no two adjacent nodes have the same label?)

For a boolean formula φ, a witness for φ would be a satisfying assignment for the variables in φ. For a graph G, a witness that G is 3-colorable is a 3-coloring of the graph G.

We also define the notion of a *polynomial time reduction*.

Definition 9.3 (Polytime reduction)**.** Let $L_A, L_B \subseteq \mathscr{L}$. We say L_A *reduces to* L_B, written as $L_A \leq_p L_B$, if and only if there exists a polytime computable function $f : \mathscr{L} \to \mathscr{L}$ such that $a \in L_A$ if and only if $f(a) \in L_B$.

Reductions are meant to capture the idea of how hard a problem is relative to another problem. The intuition is that if $L_A \leq L_B$, then L_B is in some sense "harder" than L_A as any algorithm solving L_B implies an algorithm solving L_A.

Definition 9.4 (**NP**-Hard)**.** $L \subseteq \mathscr{L}$ is **NP**-hard if for all $A \in \mathbf{NP}$, $A \leq_P L$.

That is L is **NP**-hard if it is at least as difficult to solve as all other problems in **NP**. A **NP**-hard problem need not be contained in **NP**. [1]

Definition 9.5 (**NP**-complete)**.** L is **NP**-complete if $L \in \mathbf{NP} \cap \mathbf{NP}$-hard.

Proposition 9.6 ($\mathbf{P} \subseteq \mathbf{NP}$)**.** *Proof.* If $L \in \mathbf{P}$, then some deterministic algorithm A decides L in polynomial time. A is then a polynomial time verifier for L where the witness for each $x \in L$ is ε, the empty string. □

The million dollar question: $\mathbf{NP} \subseteq \mathbf{P}$? The question of $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ is essentially asking if finding a witness is as easy as verifying it.

1.1 NP languages as theorems with proofs.

Given $L \in \mathbf{NP}$:

- Interpret the instance $x \in \{0,1\}^*$ as the (potentially false) theorem statement $x \in L$.
- For all $x \in L$, there exists a proof w (of length polynomial in $|x|$) such that $V(x, w) = 1$.
- For all $x \notin L$, no such proof exists.

This is not necessarily a different *definition* of **NP**, but rather a different framing. For example, in this context 3-colorability may be understood as

Theorem: "the input graph G is 3-colorable."

Proof: "A proper 3-coloring of G"

In our previous framework of linear versus sublinear algorithms, this type of proof system could be considered a *linear* algorithm. To decide if a proof w for an instance x is valid, we must read all of w and all of x to decide if w indeed proves x.

The notion of probabilistically checkable proofs is a surprising result showing that we can in fact verify proofs *sublinearly*.

[1] For example, it is quite easy to show that the Halting problem is **NP**-hard, but clearly not in **NP**.

2 Probabilistically checkable proofs

Definition 9.7 ($\mathbf{PCP}_{c,s}[r,q]$)**.** [2] The complexity class $\mathbf{PCP}_{c,s}[r,q]$ consists of languages L with a *non-adaptive*, randomized poly-time verifier $V(x,\pi)$ satisfying

1. If $x \in L$, there exists a poly-sized π such that $\mathbb{P}(V(x,\pi)=1) \geq c$.
2. If $x \notin L$, $\mathbb{P}(V(x,\pi)=0) \leq s$ for every "potential proof" π.
3. V uses at most $r(|x|)$ bits of randomness.
4. V makes at most $q(|x|)$ non-adaptive queries into π.

We think of the constant c, our completeness variable as close to one, and the constant s, our soundness variable, as close(ish) to zero. r is a function (not necessarily a polynomial!!) that determines the number of random bits V is allowed to use (how many times are we allowed to flip coins) and q represents the query complexity of V with respect to π.

Definition 9.8 (Canonical **PCP** settings)**.** By convention, we set

- $\mathbf{PCP}[r,q] = \mathbf{PCP}_{1,\frac{1}{2}}[r,q]$
- $\mathbf{PCP} = \mathbf{PCP}[O(\log(n)), O(1)]$

Proposition 9.9 ($\mathbf{PCP} \subseteq \mathbf{NP}$)**.** *Proof.* Let $L \in \mathbf{PCP}$. For each $x \in L$, there exists a PCP proof π for x. We will use π as our **NP** witness for x. As the **PCP** verifier $V_{\mathbf{PCP}}(x,\pi)$ uses at most $O(\log(|x|))$ random bits, we may construct a deterministic verifier $V_{\mathbf{NP}}(x,\pi)$ by enumerating all of the $2^{O(\log(|x|)} = \text{poly}(|x|)$ possible random bit strings and running $V_{\mathbf{PCP}}(x,\pi)$ on each fixed bit string. $V_{\mathbf{NP}}$ accepts if all of the runs of $V_{\mathbf{PCP}}$ accept. □

Theorem 9.10 (**PCP** theorem)**.** $\mathbf{NP} \subseteq \mathbf{PCP}$*. That is, every* $L \in \mathbf{NP}$ *has a probabilistically checkable proof.*

The proof of this theorem is quite long - so we omit it. The reason the **PCP** theorem is so surprising (and beautiful) is that it tells us that for every $x \in L$, the witness w (which required linear time to check) may be encoded into a **PCP** proof π (still of length poly($|x|$) but longer) for which it only takes a *constant* number of (random non-adaptive) queries to verify. In principle, this constant could be huge, resulting in impractical **PCP** systems. Absolutely horrible! However, with some slight relaxations, we only need *three* queries to verify a proof.

Theorem 9.11 (Håstad's 3-bit PCP)**.** *For any* $\epsilon, \delta > 0$, $\mathbf{NP} = \mathbf{PCP}_{1-\epsilon,\frac{1}{2}+\delta}[O(\log n), 3]$.

[2] The astute reader may recognize that PCP could also be an abbreviation for Phenylcyclohexyl Piperidine, a recreational dissociative drug. According to Muli Safra, a researcher who participated in proving one of the first versions of the PCP theorem, the name commemorates an incident in which a police force mistakenly suspected that Safra and his friends were running a PCP lab. Safra thought that the least they could do after being falsely accused is to actually run a PCP lab (albeit of a different nature).

3 PCP $\iff$ Gap/Inapproximability.

We have seen that PCPs can be used to verify membership in a language in sublinear time. In this section, we see how the PCP theorem has a surprising (and strong!) connection to approximation algorithms and approximability of optimization problems.

Definition 9.12 (Constraint)**.** Given:

- A set of variables $\mathcal{V} = \{v_1, \ldots, v_n\}$.
- A finite set of values Σ, called an alphabet.

A $q-$*ary constraint* is a tuple $(C \subseteq \sigma^q, i_1, \ldots, i_q)$, where $i_1, \ldots, i_q$ denotes the variables we are interested in constraining, and C denotes the set of acceptable values of variables. An assignment $a : \mathcal{V} \to \Sigma$ satisfies the constraint if $(a(v_{i_1}), \ldots, a(v_{i_q})) \in C$.

Definition 9.13 (q-CSP)**.** An *instance* x, is a triple $(\mathcal{V}, \Sigma, \mathcal{C})$ such that $\mathcal{C}$ only contains constraints which are q-ary. **Problem**: Given x, decide if x is satisfiable or not.

Proposition 9.14. *For $q \geq 2$, q-CSP is* **NP** *complete.*

Proof. (Sketch) We can phrase the **NP**-hard problem 3-colorability in terms of q-CSP, showing that q-CSP is **NP**-hard itself. To see that q-CSP is contained in **NP**, verifying if an assignment satisfies x amounts to a polynomial amount of computation. □

Example (3-Colorability as a CSP). Given $G = (V, E)$, our variables are the vertices V, our alphabet is $\Sigma = \{0, 1, 2\}$ and our constraints $\mathcal{C}$ is the edge set E.

Given a constraint set $\mathcal{C}$, it may or may not be satisfiable. We would like a measure (and a reasonable one at that) of just *how* satisfiable (or not) a particular set of constraints is.

Definition 9.15 (**UNSAT**)**.** Given a constraint set $\mathcal{C}$ (technically a instance of CSP $x = (\mathcal{V}, \Sigma, \mathcal{C})$), we define

$$\mathbf{UNSAT}(\mathcal{C}) = \min_{a:\mathcal{V}\to\Sigma} \text{fraction of constraints violated by } a.$$

Clearly, $\mathcal{C}$ is satisfiable if and only if $\mathbf{UNSAT}(\mathcal{C}) = 0$. Further, if $\mathcal{C}$ is unsatisfiable then $\mathbf{UNSAT}(\mathcal{C}) \geq \frac{1}{|\mathcal{C}|}$.

Proposition 9.16. *Given q-CSP $x = (\mathcal{V}, \Sigma, \mathcal{C})$, is is* **NP***-hard to decide if* $\mathbf{UNSAT}(\mathcal{C}) = 0$ *versus* $\mathbf{UNSAT}(\mathcal{C}) \geq \frac{1}{|\mathcal{C}|}$.

In some sense, this result is not extremely surprising, as it makes sense that such a small gap is hard to detect. However, as it turns out even large gaps are hard to differentiate.

Theorem 9.17 (The **PCP** theorem, again)**.** *For some $q \geq 2$, $|\Sigma| > 1$, it is* **NP***-hard to decide*

$$\mathbf{UNSAT}(\mathcal{C}) = 0 \quad \text{versus} \quad \mathbf{UNSAT}(\mathcal{C}) \geq \frac{1}{2}$$

We refer to this instance of q-CSP as *GAP-q-CSP*, where we are promised that either the **UNSAT** value is 0 or it is at least 1/2. What does this have to do with approximation algorithms? To see this, we view a q-CSP instance as an *optimization problem* rather than a satisfiability problem: Given a q-CSP x, maximize the number of constraints satisfied. Theorem 9.17 then implies that it is **NP**-hard to approximate the optimization problem to within a factor of 2!!

Lemma 9.18 (Theorem 9.10 is equivalent to Theorem 9.17)**.** *Proof.* We begin by showing that Theorem 9.10 implies Theorem 9.17. Consider any $L \in \mathbf{NP}$. We want to show $L \leq_P$ GAP-q-CSP. By Theorem 9.10, $L \in \mathbf{PCP}$, so there exists a non-adaptive verifier $V(x, \pi)$ for L. We may modify V into a deterministic verifier $V(x, \pi, \text{rand})$ which accepts an $O(\log(n))$ bits of randomness instead of flipping coins itself. Furthermore, the query complexity of V is a constant q. We construct a new q-CSP instance $(\mathcal{V}, \Sigma, \mathcal{C})$ defined as

- $\mathcal{V} = \{1, 2, \ldots, |\pi|\}$
- $\Sigma = \{0, 1\}$ (Our PCPs are binary strings)
- We construct our constraint set $\mathcal{C}$ via the following procedure: for each of the poly($|x|$) many internal random strings, obtain q queried locations. Add the constraint "V accepts the queries".

The fraction of strings the verifier will reject if x is not in the language is at least $\frac{1}{2}$ since **PCP** requires at least $\frac{1}{2}$ rejection probability. Therefore, for any $\ell \notin L$, the induced CSP has **UNSAT** of at least $\frac{1}{2}$.

To show the converse, assume Theorem 9.17. We will show that every language $L \in \mathbf{NP}$ has a PCP system. Fix $L \in \mathbf{NP}$. Since GAP-q-CSP is **NP** hard, we can map $x \in \{0, 1\}^*$ to a q-CSP instance $\chi = (\mathcal{V}, \Sigma, \mathcal{C})$. To construct the PCP system, consider the proof π being an assignment to $\mathcal{V}$. The verifier V will randomly choose $c \in \mathcal{C}$ and check that it is satisfied by making q queries. If $x \in L$, then we will always be able to satisfy c. Otherwise, we will accept with probability at most $\mathbf{UNSAT}(\mathcal{C})$. □