COSC 544 Probabilistic Proof Systems 10/19/17

PCPs and Succinct Arguments

Lecturer: Justin Thaler

1 PCPs: Definitions and Relationship to MIPs

In an MIP, if a prover is asked multiple questions by the verifier, then the prover can behave adaptively, which means that the prover's responses to any question can depend on the earlier questions asked by the verifier. This adaptivity was potentially good for efficiency (since it means that the prover only has to "think about" its answers to questions that are actually asked), but potentially bad for soundness, since the ability to behave adaptive makes it harder to "pin down" the prover(s) in a lie.

In contrast, Probabilistically Checkable Proofs (PCPs) have non-adaptivity baked directly into the definition, by considering a verifier $\mathcal{V}$ who is given oracle access to a static proof string π. Since π is static, $\mathcal{V}$ can ask several queries to π, and π's response to any query q_i can depend only on q_i, and not on q_j for $j \neq i$.

Definition 1.1. A probabilistically checkable proof system (PCP) for a language $\mathcal{L} \subseteq \{0,1\}^*$ consists of a probabilistic polynomial time verifier $\mathcal{V}$ who is given access to an input x, and oracle access to a proof string $\pi \in \Sigma^\ell$. The PCP has completeness error δ_c and soundness error δ_s if the following two properties hold.

1. *(Completeness)* For every $x \in L$, there exists a proof string $\pi \in \Sigma^\ell$ such that $\Pr[\mathcal{V}^\pi(x) = \text{accept}] \geq 1 - \delta_c$.

2. *(Soundness)* For every $x \notin L$ and every proof string $\pi \in \Sigma^\ell$, $\Pr[\mathcal{V}^\pi(x) = \text{accept}] \leq \delta_s$.

ℓ is referred to as the *size* of the proof, Σ as the alphabet used for the proof. We think of all of these parameters as functions of the input size n. We refer to the time required to *generate* the honest proof string π as the prover time of the PCP.

Remark 1. The PCP model was introduced by Fortnow, Rompel, and Sipser [FRS88], who referred to it as the "oracle" model (we used this terminology in Lemma 1.2 of Lecture 13). The term Probability Checkable Proofs was coined by Arora and Safra [AS98].

Remark 2. Traditionally, the notation $\mathbf{PCP}_{\delta_c,\delta_s}[r,q]_\Sigma$ is used to denote the class of languages that have a PCP verifier with completeness error δ_c, soundness error δ_s, and in which the verifier uses at most r random bits, and makes at most q queries to a proof string π over alphabet Σ. This notation is motivated in part by the importance of the parameters r, q, and Σ in applications to hardness of approximation. In the setting of verifiable computing, the most important parameters are the verifier's runtime and the prover's runtime. Note however that the proof size ℓ is a *lower bound* on the prover's runtime in any PCP system (since it takes time at least ℓ to write down a proof of length ℓ). Hence, obtaining a PCP with a small proof size is necessary, but not sufficient, for developing a PCP system with an efficient prover.

PCPs and MIPs are closely related: any MIP can turned into a PCP, and vice versa. However, both transformations can result in a substantial increase in costs.

The easier direction is turning an MIP into a PCP. This simple transformation dates back to Fortnow, Rompel, and Sipser, who introduced the PCP model, albeit under a different name.

Lemma 1.2. *Suppose $\mathcal{L} \subseteq \{0,1\}^*$ has a k-prover MIP in which $\mathcal{V}$ sends exactly one message to each prover, with each message consisting of at most r_Q bits, and each prover sends at most r_A bits in response to the verifier. Then $\mathcal{L}$ has a k-query PCP system over alphabet $\Sigma = [2^{r_A}]$, where the proof size is $k \cdot 2^{r_Q}$, with the same verifier runtime and soundness and completeness errors as the MIP.*

Sketch. For every prover $\mathcal{P}_i : i \in k$ in the MIP, the PCP proof has an entry for every possible message that $\mathcal{V}$ might send to $\mathcal{P}_i$. The entry is equal to the the prover's response that message from $\mathcal{V}$'s. The PCP verifier simulates the MIP verifier, treating the proof entries as prover answers in the MIP. □

Remark 3. It is also straightforward to obtain a PCP from a k-prover MIP in which $\mathcal{V}$ sends multiple messages to each prover: if prover $\mathcal{P}_i$ is sent m messages in the MIP, obtain a new MIP by replacing $\mathcal{P}_i$ with m provers $\mathcal{P}_{i,1}, \cdots, \mathcal{P}_{i,m}$ who are each sent one message (the message to $\mathcal{P}_{i,j}$ being the concatenation of the first j messages sent to $\mathcal{P}_i$). Then apply Lemma 1.2 to the resulting $(m \cdot k)$-prover MIP.

Lemma 1.2 highlights a fundamental difference between MIPs and PCPs: in a PCP, the prover must pre-compute a response for every possible query of the verifier, which will result in a very large prover runtime unless the number of possible queries from the verifier is small. Whereas in an MIP, the provers only need to compute responses "on demand", ignoring any queries that the verifier *might* have asked, but did not. Hence, the MIP $\Longrightarrow$ PCP transformation of Lemma 1.2 may cause a huge blowup in prover runtime.

Lemma 1.2 of Lecture 13 gave a transformation from a PCP to a 2-prover MIP, but this transformation was also expensive. In summary, the tasks of constructing efficient MIPs and PCPs are incomparable. On the one hand, PCP provers are inherently non-adaptive, but they must pre-compute the answers to all possible queries of the verifier. MIP provers only need to compute answers "on demand", but they can behave adaptively, and while there are generic techniques to force them to behave non-adaptively, these techniques are expensive.

2 A First PCP, From an MIP

In light of Lemma 1.2, it is reasonable to ask whether the MIP of Lecture 14 can be transformed into a useful PCP for non-deterministic circuit evaluation. The answer is yes, but the costs of the PCP are not quite optimal.

Suppose we are give an instance $(\mathcal{C}, x, y)$ of arithmetic circuit satisfiability. Recall that in the MIP of Lecture 14, the verifier used the first prover to run the sum-check protocol to a certain $O(\log S)$-variate polynomial $h_{x,y,Z}$ over $\mathbb{F}$. The verifier used the second prover to apply a low-degree test to the $O(\log S)$-variate polynomial Z. which required the verifier to send $\mathcal{P}_2$ a random plane in $\mathbb{F}^{\log S}$. In order to achieve a soundness error of, say, $1/\log(n)$, it was sufficient to work over a field $\mathbb{F}$ of size $\text{polylog}(S)$.

The total number of bits that the verifier sent to the each prover was $r_Q = \Theta(\log S \log |\mathbb{F}|) = \Theta(\log S \log\log S)$, since the verifier had to send a field element for each variable over which $h_{x,y,Z}$ was defined. Applying Lemma 1.2 to transform this MIP into a PCP, we obtain a PCP of length $2^{r_Q} = S^{O(\log\log S)}$. Assuming $S = \text{poly}(n)$, we obtain a PCP in which the proof string can be generated in $n^{O(\log\log n)}$ time, which is slightly superpolynomial. On the plus side, the verifier runs in linear time $O(n)$.

However, by tweaking the parameters used within the MIP itself, we can reduce r_Q from $O(\log S \log |\mathbb{F}|)$ to $O(\log S)$. Recall that within the PCP, each gate in $\mathcal{C}$ was assigned a *binary* label, and defined the functions add_i, mult_i, io, I, and W to take as input $O(\log S)$ binary variables representing the binary labels of one or more gates. The polynomial $h_{x,y,Z}$ was then defined in terms of the *multilinear* extensions of these functions. This led to a highly efficient MIP, in which the provers' runtime was $O(S \log S)$. But by defining

Communication	Queries	$\mathcal{V}$ time	$\mathcal{P}$ time
polylog(S) bits	$O(\log S/\log\log S)$	$O(n+\text{polylog}(S))$	poly(S)

Table 1: Costs of PCP of Section 2 (obtained from the MIP of Lecture 14), when run on a non-deterministic circuit $\mathcal{C}$ of size S. The stated bound on $\mathcal{P}$'s time assumes $\mathcal{P}$ knows a witness w for $\mathcal{C}$.

the polynomials to be over $\Omega(\log S)$ many variables, r_Q became slightly super logarithmic, resulting in a PCP of superpolynomial size.

Instead, we can assign each gate in $\mathcal{C}$ a label in base b, for $b = \log(S)/\log\log(S)$. Each gate label consists of only b digits in base b, since $b^b = S$. Hence, we can redefine the functions add_i, mult_i, io, I, and W to take as input $O(b) = O(\log(S)/\log\log(S))$ variables representing the b-ary labels of one or more gates, rather than having them take as input $O(\log S)$ variables.

We can then define $h_{x,y,Z}$ exactly as before, except higher-degree extensions of add_i, mult_i, io, I, and W must be used in the definition, since the multilinear extensions of the functions may not exist. Specifically, the functions each have a suitable extension of degree at most b in each variable. Compared to the MIP, the use of the higher-degree extensions increases the degrees of all of the polynomials exchanged in the sum-check protocol and in the low-degree test by polylog(S) factors, but soundness still holds if a $\mathbb{F}$ of size polylog(S) is used, for a sufficiently large polynomial in $\log S$. Hence, $r_Q = O(b \cdot \log|\mathbb{F}|) = O((\log(S)/\log\log(S)) \cdot \log\log S) = O(\log S)$.

This yields a PCP for non-deterministic circuit evaluation in which the prover's runtime is poly(S), the verifier's is $O(n)$, the number of queries the verifier makes to the proof oracle is $O(\log(S)/\log\log(S))$. The alphabet size is $|\Sigma| = 2^{\text{polylog}(S)}$, which means that each symbol of the alphabet requires polylog(S) bits to represent. Hence all of the answers to the verifier's queries can be communicated in polylog(S) bits in total.

Remark 4. To clarify, the use of labels in base $b = 2$ rather than base $b = \log(S)/\log\log(S)$ is the superior choice in interactive settings such as IPs and MIPs. The reason is that binary labels allows $\mathcal{P}$ to manipulate and send polynomials of degree $O(1)$ in each round. In the context of the GKR protocol, the MIP described in the previous section, and other interactive protocols that appear in the literature [BC12], the difference between the two parameter settings is a factor of at least b, and sometimes b^2, in both the provers' runtime and the total communication costs.

To recap, we have obtained a PCP for arithmetic circuit satisfiability with a linear time verifier, and a prover who can generate the proof in time polynomial in the size of the circuit. But to get a PCP that has any hope of being practical, we really need the prover time to be close to linear in the size of the circuit. Obtaining such PCPs is quite challenging, and we defer treatment of how to achieve this until a later lecture.

3 Compiling a PCP Into a Succinct Argument

We saw in Lecture 11 that one can turn the GKR interactive proof circuit *evaluation* into a succinct argument for circuit *satisfiability*: at the start of the argument, the prover sends a cryptographic commitment to the multilinear extension $\tilde{w}$ of a witness w; the prover and verifier then run the GKR protocol to check that $\mathcal{C}(x,w) = y$, and the verifier checks the prover's final claim about $\tilde{w}(r)$ for a random point r in the GKR protocol against the corresponding claim derived from the commitment.

The polynomial commitment scheme described in Lecture 11 consisted of two pieces; a *set-commitment* scheme using a Merkle tree, which allowed the prover to commit to *some* fixed function claim to equal $\tilde{w}$,

and a low-degree test, which allowed the verifier to check that the function committed to was indeed (close to) a low-degree polynomial.

If our goal is to transform a PCP rather than an interactive proof into a succinct argument, we can use a similar approach, but omit the low-degree test. Specifically, Kilian [Kil92] (see also Micali [Mic00]) showed that any PCP can be combined with Merkle-hashing to yield four-message argument systems for all of NP, assuming that collision-resistant hash functions exist. The prover and verifier runtimes are the same as in the underlying PCP, up to low-order factors, and the total communication cost is just $O(\log^2 n)$ per PCP query, assuming an exponentially hard collision-resistant hash function.

The idea is the following. The argument system consists of two phases: commit and reveal. In the commit phase, the prover writes down the PCP π, but doesn't send it to the verifier. Instead, the prover builds a Merkle tree, with the symbols of the PCP as the leaves, and sends the root hash of the tree to the verifier. This binds the prover to the string π. In the reveal phase, the argument system verifier simulates the PCP verifier to determine which symbols of π need to be examined (call the locations that the PCP verifier queries $q_1, \ldots, q_k$). The verifier sends $q_1, \ldots, q_k$ to the prover to $\mathcal{P}$, and the prover sends back the answers $\pi(q_1), \ldots, \pi(q_k)$, along with their authentication paths.

Completeness can be argued as follows. If the PCP satisfies perfect completeness, and then whenever there exists a w such that $\mathcal{C}(x,w) = y$, there is always some proof π that would convince the PCP verifier to accept. Hence, if the prover commits to π in the argument system, and executes the reveal phase as prescribed, the argument system verifier will also be convinced to accept.

Soundness can be argued as follows. The analysis of Lecture 12 showed that the use of the Merkle tree binds the prover to a fixed string π', in the sense that after the commit phase, for each possible query q_i, there is at most one value $\pi'(q_i)$ that the prover can successfully reveal without finding a collision under the hash function used to build the Merkle tree. Hence, if the argument system prover convinces the argument system verifier to accept, π' would convince the PCP verifier to accept. Soundness of the argument system is then immediate from soundness of the PCP system.

Remark 5. In order to turn a PCP into a succinct argument, we used a Merkle tree, and did not need to use a low-degree test. This is in contrast to Lecture 12, where we turned an interactive proof into a succinct argument by using a polynomial commitment scheme; the polynomial commitment scheme given in Lecture 12 combined a Merkle tree and a low-degree test. The low-degree test tends to be concretely quite expensive; one would not actually want to use the polynomial commitment scheme from Lecture 12 in practice. In later lectures, we will describe new, more efficient approaches to designing polynomial commitment schemes that avoid low-degree tests entirely.

However, the PCP approach to building succinct arguments has not "really" gotten rid of the low-degree test; it has just pushed it out of the commitment scheme and "into" the PCP. That is, short PCPs are themselves typically based on low-degree polynomials, and the PCP itself typically makes use of a low-degree test.

A difference between the low-degree tests that normally go into short PCPs and the low-degree tests we've already seen is that short PCPs are typically based on low-degree *univariate* polynomials (whereas efficient interactive proofs and MIPs are typically based on low-degree multivariate polynomials). So the low-degree tests that go into short PCPs are targeted at univariate rather than multi-variate polynomials. Low-degree univariate polynomials are often referred to as Reed-Solomon codes, which is why many papers on PCPs refer to "Reed-Solomon PCPs".

References

[AS98] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of NP. *J. ACM*, 45(1):70–122, 1998.

[BC12] Nir Bitansky and Alessandro Chiesa. Succinct arguments from multi-prover interactive proofs and their efficiency benefits. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 255–272. Springer, 2012.

[FRS88] Lance Fortnow, John Rompel, and Michael Sipser. On the power of multi-power interactive protocols. In *Structure in Complexity Theory Conference, 1988. Proceedings., Third Annual*, pages 156–161. IEEE, 1988.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *Proceedings of the Twenty-fourth Annual ACM Symposium on Theory of Computing*, STOC '92, pages 723–732, New York, NY, USA, 1992. ACM.

[Mic00] Silvio Micali. Computationally sound proofs. *SIAM J. Comput.*, 30(4):1253–1298, 2000.