Lecture B.10

Applications of PCPs

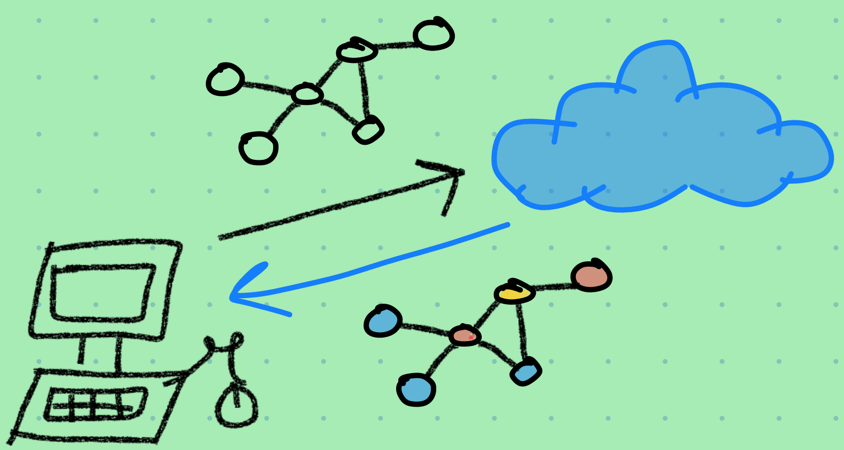
Summer Graduate School on
Foundations and Frontiers of Probabilistic Proofs
2021.08.05

Applications of PCPs

Two main directions:

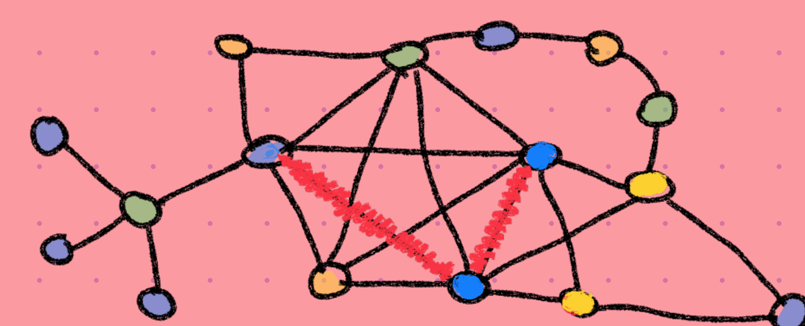
Delegation of computation

outsource your problem
& efficently verify the solution!



Hardness of approximation

which problems remain hard
even if we only require
an approximate solution?

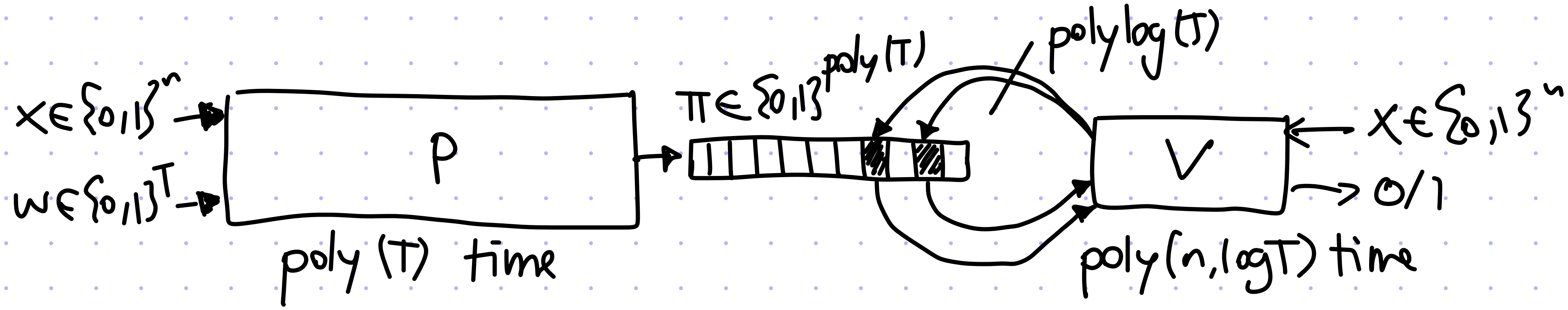


Delegation of Computation via PCPs

We showed the following result:

<u>theorem:</u> Every language $L \in \mathrm{NTIME}(T)$ has a PCP where:

proof length $\ell = \mathrm{poly}(T)$ query complexity $q = \mathrm{polylog}(T)$
prover time $pt = \mathrm{poly}(T)$ verifier time $vt = \mathrm{poly}(n, \log T)$



In this setup, a single reliable PC can monitor the operation of a herd of supercomputers working with possibly extremely powerful but unreliable software and untested hardware.

But how to use this "setup"?

Checking Computations in Polylogarithmic Time

László Babai[1]
Univ. of Chicago[6] and
Eötvös Univ., Budapest

Lance Fortnow[2]
Dept. Comp. Sci.
Univ. of Chicago[6]

Leonid A. Levin[3]
Dept. Comp. Sci.
Boston University[4]

Mario Szegedy[5]
Dept. Comp. Sci.
Univ. of Chicago[6]

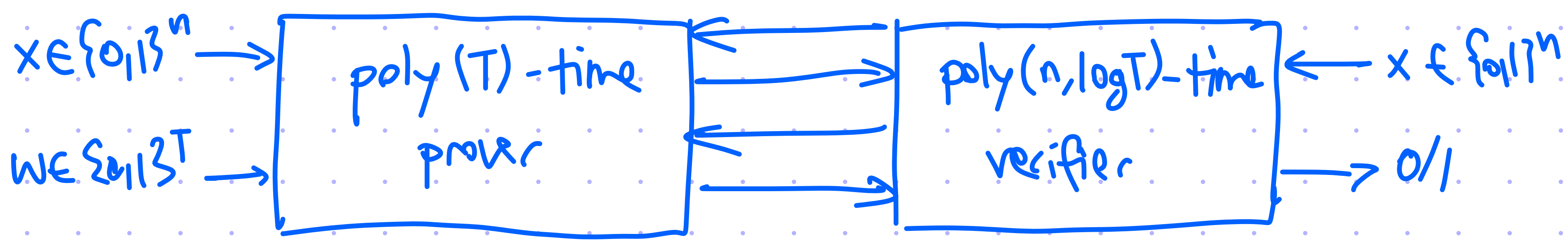
A Crypto Interlude: From PCP to Interactive Arguments

<u>theorem</u> [informal]

Suppose L has a PCP with prover time pt, verifier time vt, query complexity q.
Then by using cryptography we can construct an interactive "proof" for L s.t.

prover time $O(pt)$, verifier time $O(vt)$, communication $O(q)$.

If we apply this to PCPs in prior slide, we get a powerful result:



[does NOT contradict limitations of IPs with small communication!]

<u>Proof attempt:</u>

$P(x,w)$		$V(x)$
	$\xleftarrow{\rho}$	sample PCP randomness ρ
• produce PCP string: $\Pi := P_{PCP}(x,w)$ • deduce query set Q in $V_{PCP}^{\Pi}(x;\rho)$	$\xrightarrow{\Pi\vert_Q}$	$V_{PCP}^{\Pi\vert_Q}(x;\rho) \stackrel{?}{=} 1$

<u>Problem:</u> prover can pick Π based on Q.

[Also, where is the crypto??]

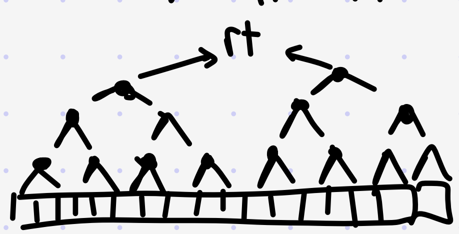
A Crypto Interlude: Kilian's Protocol

Idea: commit to PCP string first then locally open locations of it

Def: A function family $H_\lambda = \{h_\lambda : \{0,1\}^{2\lambda} \to \{0,1\}^{\lambda}\}$ is collision-resistant if $\forall$ efficient adversary A

$$\Pr_{h \leftarrow H_\lambda}\left[\tilde{A}(h) \text{ outputs } x \neq y \text{ s.t. } h(x) = h(y)\right] \text{ is negligible in } \lambda.$$

The new protocol is as follows:

$P(x,w)$		$V(x)$
	$\xleftarrow{\;h\;}$	$h \leftarrow H_\lambda$
• produce PCP string: $\pi := P_{PCP}(x,w)$ • commit to it: $rt = MT_h(\pi)$	$\xrightarrow{\;rt\;}$	
	$\xleftarrow{\;\rho\;}$	sample PCP randomness ρ
• deduce query set Q in $V_{PCP}^{\pi}(x;\rho)$ • produce auth paths for each answer	$\xrightarrow{\;\pi\vert_Q,\ auth\;}$	$V_{PCP}^{\pi\vert_Q}(x;\rho) \overset{?}{=} 1$ & check auth
$time(P_{PCP}) + O_\lambda(\ell)$	$O_\lambda(q \log \ell)$	$time(V_{PCP}) + O_\lambda(q \log \ell)$

Security analysis involves cryptography and so we will not discuss it.

From IOP to Interactive Argument [1/2]

<u>theorem</u> [informal]

Suppose L has a public-coin IOP with prover time pt, verifier time vt, query complexity q. Then by using cryptography we can construct an interactive argument for L with

prover time $O(pt)$, verifier time $O(vt)$, communication $O(q)$.

<u>proof attempt</u>:

$P(x)$

1. deduce all the oracles:

   $\Pi_1 := P_{IOP}(x)$, $\Pi_2 := P_{IOP}(x, r_1)$, …, $\Pi_k := P_{IOP}(x, r_1, \dots, r_{k-1})$

2. deduce IOP verifier's queries:

   $Q := \text{queries}\left(V_{IOP}^{\Pi_1,\dots,\Pi_k}(x; r_1, \dots, r_k)\right)$

$P \xleftarrow{r_1, \dots, r_k} V$

$P \xrightarrow{[\Pi_1, \dots, \Pi_k]|_Q} V$

$V(x)$

$V_{IOP}^{[\Pi_1,\dots,\Pi_k]|_Q}(x; r_1, \dots, r_k) \stackrel{?}{=} 1$

This is NOT secure because the prover can answer queries based on $r_1, \dots, r_k$!

Idea: extend Kilian's protocol from PCP to IOP by committing to each oracle via a Merkle tree and then locally open the relevant locations.

From IOP to Interactive Argument [2/2]

As in Kilian's protocol, we rely on collision-resistant functions to build Merkle trees.

$P(x)$		$V(x)$
	$\xleftarrow{h}$	sample CRH: $h \leftarrow H_\lambda$
$\pi_1 := P_{IOP}(x)$, $rt_1 := MT_h(\pi_1)$	$\xrightarrow{rt_1}$	
	$\xleftarrow{r_1}$	
$\pi_2 := P_{IOP}(x, r_1)$, $rt_2 := MT_h(\pi_2)$	$\xrightarrow{rt_2}$	
	$\xleftarrow{r_2}$	
	$\vdots$	
$\pi_k := P_{IOP}(x, r_1, \ldots, r_{k-1})$, $rt_k := MT_h(\pi_k)$	$\xrightarrow{rt_k}$	
	$\xleftarrow{r_k}$	
• deduce IOP verifier's queries: $Q := \text{queries}\left(V_{IOP}^{\pi_1,\ldots,\pi_k}(x; r_1, \ldots, r_k)\right)$ • produce auth paths for each answer	$\xrightarrow{\text{ans, paths}}$	$V_{IOP}^{\text{ans}}(x; r_1, \ldots, r_k) \stackrel{?}{=} 1$ & check paths

$\text{time}(P) = \text{time}(P_{IOP}) + O_\lambda(\ell)$

$\underbrace{\qquad\qquad}_{O_\lambda(q \log \ell)}$

$\text{time}(V) = \text{time}(V_{IOP}) + O_\lambda(q \log \ell)$

Security analysis involves cryptography and so we will not discuss it.

In sum, designing efficient IOPs leads to efficient arguments.

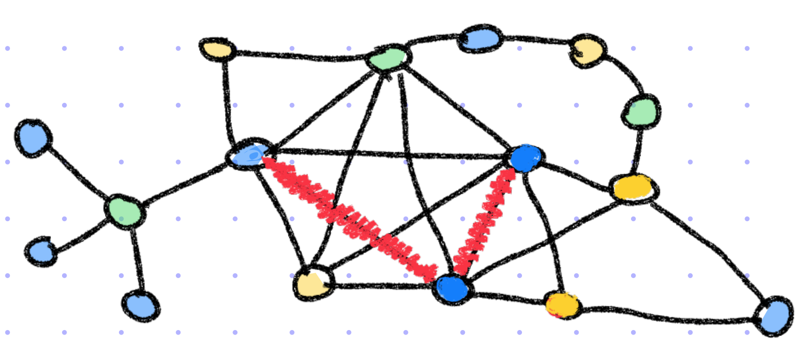
Hardness of approximation

During the 70s and 80s many key optimisation problems were shown to be NP-hard.

E.g., 3SAT, Knapsack, Hamiltonian path, Travelling salesman, 3COL, Clique, Super Mario...

It's natural to seek approximate solutions!

Def: An algorithm A is an α-approx alg for a maximisation problem Π if $\forall x \quad \alpha \cdot \Pi(x) \leq A(x) \leq \Pi(x)$



Example: Max-3SAT

Problem: Given a 3CNF formula, find an assignment that maximises the number of satisfied clauses.

Ex: $\varphi = (x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_2} \vee \overline{x_3} \vee x_1) \wedge (\overline{x_1} \vee \overline{x_4} \vee x_3)$
$\wedge (x_1 \vee \overline{x_2} \vee x_4) \wedge (x_2 \vee x_3 \vee \overline{x_4})$

Claim: There exists a $\frac{7}{8}$-approx poly-time algorithm for Max-3SAT

The main idea is to observe that there must be many $\frac{7}{8}$-sat assignments, so it suffices to guess some at random!

Example: Max-3SAT

Claim: There exists a $\frac{7}{8}$-approx poly-time algorithm for Max-3SAT

Proof: Let c_i be the indicator of whether the i'th clause is sat.

Note that $E[c_i] = Pr[c_i \text{ is sat}] = \frac{7}{8}$.

only one assignment fail to sat. a disjunction

Thus $E\left[\sum_{i=1}^{m} c_i\right] = \sum_{i=1}^{m} E[c_i] = \frac{7}{8}m$.

Hence there exists an assignment satisfying $\frac{7}{8}m$ clauses

Moreover, by Markov, at least $\frac{1}{m+1}$ of the assignment satisfy at least $\frac{7}{8}m$ clauses.

Thus it suffices to choose and check $O(m)$ random assignments ∎

How to show hardness of approximation?

Recall that NP-hardness is defined for decision problems.

Definition: Let Π be an approximation problem.

$\mathrm{Gap}\Pi_{c,s}$ is the problem of deciding

whether $\Pi(x) \geq c$ or $\Pi(x) \leq s$.

Claim: If $\mathrm{Gap}\Pi_{c,s}$ is NP-hard, then it is NP-hard

to approximate Π to $\frac{s}{c}$ precision

Proof: We reduce the gap problem to the approx. problem.

If $\Pi(x) \geq c$, the approx. is at least $\frac{s}{c} \cdot c = s$

If $\Pi(x) \leq s$, the approx is at most s.

PCP <=> hardness of approximation

We'll illustrate the connection via an example.

Theorem: Gap3SAT$(1,s)$ is NP-hard for $s<1$ iff
$3SAT \in PCP_{1,s'}[O(\log n), 3]$ for $s'<1$. $s' = \frac{7}{8} + \varepsilon$

we proved this!

Proof: $\boxed{=>}$ Apply the reduction from 3SAT to Gap3SAT.
The NP proof corr. to the Gap3SAT instance.
Now sample clauses at random.

$\boxed{<=}$ We reduce 3SAT to Gap3SAT.
Express the PCP verifier checks as a 3CNF.
Completeness implies 1-satisfiability.
Soundness implies at most s'-satisfiability.

Concluding words