Computational complexity

lecture 13

PCP

The **PCP** theorem gives another, interesting definition of the **NP** class, as the set of languages that have a “locally checkable” proofs of belonging to the language.

In effect, we obtain hardness of approximation for many **NP**-complete problems.

PCP = “probabilistically checkable proof”

PCP

The **PCP** theorem gives another, interesting definition of the **NP** class, as the set of languages that have a “locally checkable” proofs of belonging to the language.

In effect, we obtain hardness of approximation for many **NP**-complete problems.

- The question whether **P≠NP** is not only an important theoretical question. It is also important from the practical point of view, because of many real-life problems that are **NP**-hard.
- In practice, in many applications it is not necessary to find the (completely) best solution, it is enought to have a solution close to the best one (approximation)
- In effect, the **PCP** theorem (hardness of approximation) is important from the practical point of view: it shows for many problems that even their approximation is **NP**-hard

PCP

The **PCP** theorem gives another, interesting definition of the **NP** class, as the set of languages that have a “locally checkable” proofs of belonging to the language. Somehow similarly to the theorem saying that **IP**=**PSPACE**. The idea:

- Suppose that someone wants to convince us that a Boolean formula is satisfiable.
- He can show us a standard witness, that is, a valuation. In order to check it, we substitute it to the formula. In order to do this, though, we have to read the whole witness.
- The **PCP** theorem gives us an interesting alternative: the “prover” can write his witness (his proof) in such a way, that we can check its correctness by randomly choosing only a constant number of letters to be read (it is enough to read only 3 bits).
- A correct witness will be always accepted.
- If a formula is not satisfiable, with high probability we will reject every proposed witness with high probability.

PCP

Example: non-isomorphism of graphs G_1 and G_2

- An **IP** approach: V picks $i \in \{1,2\}$ at random, creates a graph H permuting randomly nodes of G_i, and asks P: "is H jest isomorphic to G_1 or to G_2?"
- A **PCP** approach: Now P provides a huge witness (of exponential size), which for every graph H says: to which graph G_i is the graph H isomorphic. Having this witness, V picks $i \in \{1,2\}$ at random, creates a graph H permuting randomly nodes of G_i, and reads from the proof to which graph is H isomorphic. To this end, V needs $poly(n)$ random bits, but he reads only 1 bit of the proof.

PCP

Definition: **PCP**($r(n)$,$q(n)$)-verifier for a language L – a randomized machine V, working in polynomial time (wrt. the length of the input word), which:

- on a word w of length n, having access to a word π (a proof / a witness), uses $r(n)$ random bits, and reads $q(n)$ positions of π
- we assume that V writes numbers of positions to be read on a special tape, and then in a single step he receives bits written on these positions
- in particular V is not adaptive: consecutive questions do not depend on answers to previous questions (we ask all questions at once)
- for $w \in L$ there exists π such that V always accepts
- for $w \notin L$, for every π V accepts with probability ≤1/2

The language L is in the class **PCP**($r(n)$,$q(n)$) if there exist constants c, d such that there exists a **PCP**($c \cdot r(n)$, $d \cdot q(n)$)-verifier for L

PCP

- Fact: amplification – the number *1/2* in the definition of **PCP** can be replaced by any number from the interval *(0,1)* (simple exercise)

PCP

- Fact: amplification – the number $1/2$ in the definition of **PCP** can be replaced by any number from the interval $(0,1)$ (simple exercise)
- Fact: we can assume that a **PCP**($r(n), q(n)$)-verifier receives a proof of length at most $q(n)2^{r(n)}$, because anyway he is able to check only this number of positions
- For example, if $r(n)=O(\log n)$, then we can restrict ourselves to proofs of polynomial length

PCP

- <u>Fact</u>: amplification – the number $1/2$ in the definition of **PCP** can be replaced by any number from the interval $(0,1)$ (simple exercise)
- <u>Fact</u>: we can assume that a **PCP**($r(n), q(n)$)-verifier receives a proof of length at most $q(n)2^{r(n)}$, because anyway he is able to check only this number of positions
- For example, if $r(n)=O(\log n)$, then we can restrict ourselves to proofs of polynomial length
- Trivial cases: **PCP**($poly(n), 0$)=**coRP**, **PCP**($0, poly(n)$)=**NP**
- Tutorials: **PCP**($\log n, poly(n)$)=**NP**

PCP

- <u>Fact</u>: amplification – the number $1/2$ in the definition of **PCP** can be replaced by any number from the interval $(0,1)$ (simple exercise)
- <u>Fact</u>: we can assume that a **PCP**($r(n), q(n)$)-verifier receives a proof of length at most $q(n)2^{r(n)}$, because anyway he is able to check only this number of positions
- For example, if $r(n)=O(\log n)$, then we can restrict ourselves to proofs of polynomial length
- Trivial cases: **PCP**($poly(n), 0$)=**coRP**, **PCP**($0, poly(n)$)=**NP**
- Tutorials: **PCP**($\log n, poly(n)$)=**NP**
- <u>The **PCP** Theomem</u> (Arora, Lund, Motwani, Safra, Sudan, Szegedy 1992):
  PCP($\log n, 1$)=**NP**

PCP

- Fact: amplification – the number $1/2$ in the definition of **PCP** can be replaced by any number from the interval $(0,1)$ (simple exercise)
- Fact: we can assume that a **PCP**($r(n), q(n)$)-verifier receives a proof of length at most $q(n)2^{r(n)}$, because anyway he is able to check only this number of positions
- For example, if $r(n)=O(\log n)$, then we can restrict ourselves to proofs of polynomial length
- Trivial cases: **PCP**($poly(n), 0$)=**coRP**, **PCP**($0, poly(n)$)=**NP**
- Tutorials: **PCP**($\log n, poly(n)$)=**NP**
- The **PCP** Theomem (Arora, Lund, Motwani, Safra, Sudan, Szegedy 1992):
  PCP($\log n, 1$)=**NP**
- The verifier reads a constant number of bits. How many?
  - This does not depend on the choice of the language (reductions)
  - The original theorem: about 10^6
  - [1998] It is enough to read 3 bits, for error $1/2+\varepsilon$
    (and reading 2 bits is not sufficient)

PCP

<u>The **PCP** Theomem</u> (Arora, Lund, Motwani, Safra, Sudan, Szegedy 1992): **PCP**($log\ n, 1$)=**NP**

- Inclusion **PCP**($log\ n, 1$)⊆**NP** obvious: a proof is of polynomial length, so it can serve as a witness, and in polynomial time we can check all possible sequences of $O(log\ n)$ random bits
- We remark that verifiers tossing less than $O(log\ n)$ random bits do not make too much sense, since some parts of proofs (of polynomial length) will be never read by such verifiers

PCP

The **PCP** Theomem (Arora, Lund, Motwani, Safra, Sudan, Szegedy 1992): **PCP**($log\ n, 1$)=**NP**

This means that for every problem in **NP**, there is a verifier s.t.

- given an input word, it expects a proof of polynomial size
- tossing $log\ n$ random bits it checks a contant number of bits of the proof
- basing on this, it certainly accepts all correct words, and with high probability it rejects incorrect words

This is a strange theorem. Consider, e.g., 3-colorability of a graph, where a coloring serves as a proof. If the coloring is incorrect in a single place, it is difficult to find this place (more-or-less, the whole coloring has to be read). The **PCP** theorem says that the coloring can be written in such a way that every error is visible in many places.

Important! we should reject with high probability in two cases:

- when we have a (correct) encoding of an incorrect coloring,
- when the proof is not a correct encoding of any coloring.
  (ensuring the latter seems much more difficult)

PCP

<u>The **PCP** Theomem</u> (Arora, Lund, Motwani, Safra, Sudan, Szegedy 1992): **PCP**($log\ n, 1$)=**NP**

Consider another problem: does a given mathematical theorem ϕ have a proof of length n, where n is given in unary?

Ordinarily, in order to check a proof (given in a classic way), it is necessary to read the whole proof, and an error in every single place disqualifies the whole proof. The **PCP** theorem implies that there is such a format for writing proofs, that:

- every error can be detected with high probability, by checking a random fragment
- with high probability, one can also reject proofs which do not follow the format

PCP vs approximation

We will prove that the problem of $1/2$-approximating the size of the largest clique is **NP**-hard

PCP vs approximation

We will prove that the problem of $1/2$-approximating the size of the largest clique is **NP**-hard

<u>What does it mean?</u>

There is a reduction from every problem L in **NP** to the clique problem (i.e., a function converting inputs of problem L to inputs of the clique problem, computable in logarithmic space), such that:

- instances with answer YES are transformed to instances (G,k) such that in G there is a clique of size k
- instances with answer NO are transformed to instances (G,k) such that in G there is no clique of size $k/2$

PCP vs approximation

We will prove that the problem of $1/2$-approximating the size of the largest clique is **NP**-hard

- Fix a language $L \in$ **NP**. There is a **PCP**$(c \cdot log(n), d)$-verifier V for L
- Consider an input word w. Let $q_i(w,r)$ denote the i-th position of the proof read by V for input w and a sequence of random bits r.
- Take $k=2^{clog(n)}$ (the size of a clique). We construct a graph G.
- As nodes we take $(r,a_1,\ldots,a_d)$, where $r \in \{0,1\}^{clog(n)}$, $a_i \in \{0,1\}$, such that if the input is w, random bits are r, and bits read from the proof are $a_1,\ldots,a_d$, then V accepts
- We create an edge between $(r,a_1,\ldots,a_d),(r',b_1,\ldots,b_d)$ if they are consistent, i.e., if $q_i(w,r)=q_j(w,r')$ implies $a_i=b_j$ (edges exist only for $r \neq r'$)

PCP vs approximation

We will prove that the problem of $1/2$-approximating the size of the largest clique is **NP**-hard

- Fix a language $L \in$ **NP**. There is a **PCP**$(c \cdot log(n),d)$-verifier V for L
- Consider an input word w. Let $q_i(w,r)$ denote the i-th position of the proof read by V for input w and a sequence of random bits r.
- Take $k=2^{clog(n)}$ (the size of a clique). We construct a graph G.
- As nodes we take $(r,a_1,...,a_d)$, where $r \in \{0,1\}^{clog(n)}$, $a_i \in \{0,1\}$, such that if the input is w, random bits are r, and bits read from the proof are $a_1,...,a_d$, then V accepts
- We create an edge between $(r,a_1,...,a_d),(r',b_1,...,b_d)$ if they are consistent, i.e., if $q_i(w,r)=q_j(w,r')$ implies $a_i=b_j$ (edges exist only for $r \neq r'$)
- If $w \in L$, then there exists a correct proof π
- For every r we take one node $(r,a_1,...,a_d)$, where as a_i we take the $q_i(w,r)$-th bit of the proof π. They form a clique of size $k=2^{clog(n)}$

PCP vs approximation

We will prove that the problem of $1/2$-approximating the size of the largest clique is **NP**-hard

- Fix a language $L \in$ **NP**. There is a **PCP**$(c \cdot log(n), d)$-verifier V for L.
- Consider an input word w. Let $q_i(w,r)$ denote the i-th position of the proof read by V for input w and a sequence of random bits r.
- Take $k = 2^{clog(n)}$ (the size of a clique). We construct a graph G.
- As nodes we take $(r, a_1, \ldots, a_d)$, where $r \in \{0,1\}^{clog(n)}$, $a_i \in \{0,1\}$, such that if the input is w, random bits are r, and bits read from the proof are $a_1, \ldots, a_d$, then V accepts
- We create an edge between $(r, a_1, \ldots, a_d), (r', b_1, \ldots, b_d)$ if they are consistent, i.e., if $q_i(w,r) = q_j(w,r')$ implies $a_i = b_j$ (edges exist only for $r \neq r'$)
- Every clique of size m defines a proof: if $(r, a_1, \ldots, a_d)$ is in the clique, as the $q_i(w,r)$-th bit of a proof π we take a_i; remaining bits arbitrarily
- V accepts π with probability $\geq m/k \Rightarrow$ for $w \notin L$ we have $m < k/2$

PCP vs approximation

We have proved that the problem of $1/2$-approximating the size of the largest clique is **NP**-hard

- Using amplification for **PCP**, we can prove the same for every constant $c \in (0,1)$ instead of $1/2$
- One can even show that for every constant $c \in (0,1)$, the problem of n^{-c}-approximation is **NP**-hard (i.e., finding a clique of size $best\_size/n^{c}$), by appropriately modifying the resulting graph G, using so-called *expanders*
- This result cannot be stronger: one can always find a clique of size $best\_size/n$ – a single node

PCP vs approximation - 3SAT

MAX3SAT – find a valuation for which the largest number of clauses is satisfied.

We will show that for some constant $\rho \in (0,1)$, the ρ-approximation of this problem (i.e., obtaining $\rho \cdot opt$ satisfied clauses) is **NP**-hard

PCP vs approximation - 3SAT

MAX3SAT – find a valuation for which the largest number of clauses is satisfied.

We will show that for some constant $\rho \in (0,1)$, the ρ-approximation of this problem (i.e., obtaining $\rho \cdot opt$ satisfied clauses) is **NP**-hard

- Fix a language $L \in$ **NP**. There is a **PCP**$(c \cdot log(n), d)$-verifier V for L.
- Consider an input word w. Variables of a created formula describe consecutive bits of a proof (their number = expected proof length).
- For every sequence r of random bits, V reads d bits of a proof. Basing on this, we create a formula ϕ_r (a disjunction of $\leq 2^d$ conjunctions) saying that this bits have values for which V accepts.
- We replace every ϕ_r by a conjunction of $\leq f(d)$ (constant number) of clauses (disjunctions) of length 3, introducing fresh variables.
- We take a conjunction of these formulas over all sequences r.

PCP vs approximation - 3SAT

MAX3SAT – find a valuation for which the largest number of clauses is satisfied.

We will show that for some constant $\rho \in (0,1)$, the ρ-approximation of this problem (i.e., obtaining $\rho \cdot opt$ satisfied clauses) is **NP**-hard

- Fix a language $L \in$ **NP**. There is a **PCP**$(c \cdot log(n),d)$-verifier V for L.
- Consider an input word w. Variables of a created formula describe consecutive bits of a proof (their number = expected proof length).
- For every sequence r of random bits, V reads d bits of a proof. Basing on this, we create a formula ϕ_r (a disjunction of $\leq 2^d$ conjunctions) saying that this bits have values for which V accepts.
- We replace every ϕ_r by a conjunction of $\leq f(d)$ (constant number) of clauses (disjunctions) of length 3, introducing fresh variables.
- We take a conjunction of these formulas over all sequences r.
- Correspondence: proofs $\leftrightarrow$ valuations of variables
- $w \in L \Rightarrow$ exists a correct proof $\Rightarrow$ all clauses satisfied

PCP vs approximation - 3SAT

MAX3SAT – find a valuation for which the largest number of clauses is satisfied.

We will show that for some constant $\rho \in (0,1)$, the ρ-approximation of this problem (i.e., obtaining $\rho \cdot opt$ satisfied clauses) is **NP**-hard

- Fix a language $L \in$ **NP**. There is a **PCP**$(c \cdot log(n),d)$-verifier V for L.
- Consider an input word w. Variables of a created formula describe consecutive bits of a proof (their number = expected proof length).
- For every sequence r of random bits, V reads d bits of a proof. Basing on this, we create a formula ϕ_r (a disjunction of $\leq 2^d$ conjunctions) saying that this bits have values for which V accepts.
- We replace every ϕ_r by a conjunction of $\leq f(d)$ (constant number) of clauses (disjunctions) of length 3, introducing fresh variables.
- We take a conjunction of these formulas over all sequences r.
- Correspondence: proofs $\leftrightarrow$ valuations of variables
- $w \notin L \Rightarrow$ every proof rejected for $>1/2$ sequences $r \Rightarrow$ for these r ≥ 1 false clause $\Rightarrow$ a fraction of $>1/(2f(d))$ false clauses

PCP vs approximation - 3SAT

MAX3SAT – find a valuation for which the largest number of clauses is satisfied.

We have shown that for some constant $\rho \in (0,1)$, the ρ-approximation of this problem (i.e., obtaining $\rho \cdot opt$ satisfied clauses) is **NP**-hard

This can be shown for $\rho = 7/8 + \varepsilon$ (for every $\varepsilon > 0$), using the „3-bits" version of the PCP theorem.

On the other hand, there is an easy algorithm for $7/8$-approximation, because the expected number of clauses satisfied by a random valuation is $7/8$.

PCP vs approximation - 3SAT

MAX3SAT – find a valuation for which the largest number of clauses is satisfied.

We have shown that for some constant $\rho \in (0,1)$, the ρ-approximation of this problem (i.e., obtaining $\rho \cdot opt$ satisfied clauses) is **NP**-hard

This can be shown for $\rho = 7/8 + \varepsilon$ (for every $\varepsilon > 0$), using the „3-bits" version of the PCP theorem.

On the other hand, there is an easy algorithm for $7/8$-approximation, because the expected number of clauses satisfied by a random valuation is $7/8$.

Remark

Using the formula obtained from the standard proof of **NP**-hardness of 3SAT, we cannot prove hardness of approximation.

Remark 2

Thanks to the **PCP** theorem, for many problems we can very precisely say what is the best factor of approximation.

PCP – idea of a proof

We will prove an easier version of the **PCP** theorem:

NP$\subseteq$**PCP**$(poly(n), 1)$

- even this inclusion is surprising
- the proof of the full **PCP** theorem bases on this inclusion
- it turns out that **PCP**$(poly(n), 1)$=**NEXPTIME**

PCP – idea of a proof

We will prove that **NP**$\subseteq$**PCP**($poly(n), 1$).

- Walsh-Hadamard codes: to a sequence of bits $\boldsymbol{v}\in\{0,1\}^n$ we assign the following function from $\{0,1\}^n$ to $\{0,1\}$:

$$\boldsymbol{x} \rightarrow \boldsymbol{x}\cdot\boldsymbol{v} = x_1\cdot v_1+\ldots+x_n\cdot v_n \ \ (mod\ 2)$$ (scalar product)

  Every such a function can be written as a sequence of length 2^n

PCP – idea of a proof

We will prove that **NP**$\subseteq$**PCP**($poly(n), 1$).

- Walsh-Hadamard codes: to a sequence of bits $\boldsymbol{v}\in\{0,1\}^n$ we assign the following function from $\{0,1\}^n$ to $\{0,1\}$:

  $$\boldsymbol{x} \to \boldsymbol{x}\cdot\boldsymbol{v} = x_1\cdot v_1+\ldots+x_n\cdot v_n \pmod 2$$ (scalar product)

  Every such a function can be written as a sequence of length 2^n
- This is a linear function, i.e., $f(\boldsymbol{x}+\boldsymbol{y})=f(\boldsymbol{x})+f(\boldsymbol{y})$ for all $\boldsymbol{x},\boldsymbol{y}$
- If $\boldsymbol{v}\neq\boldsymbol{w}$, then their encodings differ on exactly half positions
  (because: a nonempty set has the same number of subsets of even size as subsets of odd size)

PCP – idea of a proof

Our first goal: check that a given function $f:\{0,1\}^n \rightarrow \{0,1\}$ (given as a sequence of length 2^n) is a W-H code of some sequence of length n

PCP – idea of a proof

Our first goal: check that a given function $f:\{0,1\}^n \to \{0,1\}$ (given as a sequence of length 2^n) is a W-H code of some sequence of length n

- W-H codes = linear functions: for every linear function it holds

  $$f(\mathbf{x})=f(x_1 \cdot b_1+\ldots+x_n \cdot b_n)=x_1 \cdot f(b_1)+\ldots+x_n \cdot f(b_n)=\mathbf{x}\cdot(f(b_1),\ldots,f(b_n))$$

  where b_i – base vectors
- thus we need to check that a function is linear

PCP – idea of a proof

Our first goal: check that a given function $f:\{0,1\}^n \to \{0,1\}$ (given as a sequence of length 2^n) is a W-H code of some sequence of length n

- W-H codes = linear functions: for every linear function it holds

  $$f(\mathbf{x})=f(x_1 \cdot b_1+\ldots+x_n \cdot b_n)=x_1 \cdot f(b_1)+\ldots+x_n \cdot f(b_n)=\mathbf{x} \cdot (f(b_1),\ldots,f(b_n))$$

  where b_i – base vectors
- thus we need to check that a function is linear
- this can be checked only approximately, if we read only a few bits
- for $\rho \in (0,1)$ we say that a function $f:\{0,1\}^n \to \{0,1\}$ is *ρ-close to a linear function* if there is a linear function g such that

  $$Pr_{\mathbf{x} \in \{0,1\}^n}[f(\mathbf{x})=g(\mathbf{x})] \geq \rho$$

PCP – idea of a proof

Our first goal: check that a given function $f:\{0,1\}^n \to \{0,1\}$ (given as a sequence of length 2^n) is a W-H code of some sequence of length n

- W-H codes = linear functions: for every linear function it holds
  $$f(\mathbf{x})=f(x_1 \cdot b_1+\ldots+x_n \cdot b_n)=x_1 \cdot f(b_1)+\ldots+x_n \cdot f(b_n)=\mathbf{x}\cdot(f(b_1),\ldots,f(b_n))$$
  where b_i – base vectors
- thus we need to check that a function is linear
- this can be checked only approximately, if we read only a few bits
- for $\rho \in (0,1)$ we say that a function $f:\{0,1\}^n \to \{0,1\}$ is *ρ-close to a linear function* if there is a linear function g such that
  $$Pr_{\mathbf{x}\in\{0,1\}^n}[f(\mathbf{x})=g(\mathbf{x})]\geq\rho$$
- we use the following theorem: if a function f satisfies
  $$Pr_{\mathbf{x}\in\{0,1\}^n}[f(\mathbf{x}+\mathbf{y})=f(\mathbf{x})+f(\mathbf{y})]\geq\rho$$
  (for some $\rho>1/2$), then it is ρ-close to linear
- checking (a few times) linearity on selected arguments, we can ensure that with probability $\geq\rho$ the function f is ρ-close to a linear function

PCP – idea of a proof

Assume that f is ρ-close to a linear function g, where $\rho > 3/4$.

Then g is determined uniquely (because different linear functions differ for at least half of arguments).

PCP – idea of a proof

Assume that f is ρ-close to a linear function g, where $\rho>3/4$.

Then g is determined uniquely (because different linear functions differ for at least half of arguments). Suppose that we are given an argument $\boldsymbol{x}$, and we want to compute $g(\boldsymbol{x})$ having access to f. For every $\boldsymbol{x}$ we want to succeed with high probability.

- Reading of $f(\boldsymbol{x})$ does not have this properyty: if it happened that $f(\boldsymbol{x})\neq g(\boldsymbol{x})$, then we (always) obtain an incorrect result.

PCP – idea of a proof

Assume that f is ρ-close to a linear function g, where $\rho > 3/4$.

Then g is determined uniquely (because different linear functions differ for at least half of arguments). Suppose that we are given an argument $\mathbf{x}$, and we want to compute $g(\mathbf{x})$ having access to f. For every $\mathbf{x}$ we want to succeed with high probability.

- Reading of $f(\mathbf{x})$ does not have this properyty: if it happened that $f(\mathbf{x}) \neq g(\mathbf{x})$, then we (always) obtain an incorrect result.
- Instead, we randomly choose $\mathbf{y}$, and we return $f(\mathbf{y})+f(\mathbf{x}+\mathbf{y})$
- With high probability $f(\mathbf{y})=g(\mathbf{y})$ and $f(\mathbf{x}+\mathbf{y})=g(\mathbf{x}+\mathbf{y})$, that is, $f(\mathbf{y})+f(\mathbf{x}+\mathbf{y})=g(\mathbf{y})+g(\mathbf{x}+\mathbf{y})=g(\mathbf{x})$

PCP – idea of a proof

We will show a $(poly(n),1)$-verifier for the following **NP**-complete problem: is a given system of quadratic equation over $\mathbb{Z}_2$ satisfiable?

An example system of such equations:

$$x_1x_2+x_3x_4+x_2x_5=1 \quad (mod\ 2)$$

$$x_2x_3+x_4x_5=0 \quad (mod\ 2)$$

$$x_1x_3+x_3x_5+x_3x_4=1 \quad (mod\ 2)$$

(valuation $x_1=x_2=x_3=x_4=x_5=1$ satisfies this system)

PCP – idea of a proof

We will show a $(poly(n),1)$-verifier for the following **NP**-complete problem: is a given system of quadratic equation over $\mathbb{Z}_2$ satisfiable?

An example system of such equations:

$$x_1x_2+x_3x_4+x_2x_5=1 \quad (mod\ 2)$$

$$x_2x_3+x_4x_5=0 \quad (mod\ 2)$$

$$x_1x_3+x_3x_5+x_3x_4=1 \quad (mod\ 2)$$

(valuation $x_1=x_2=x_3=x_4=x_5=1$ satisfies this system)

A system of m equations for n variables can be represented by a matrix A of size $m \times n^2$, and a vector $\boldsymbol{b}$ of length m.

We ask whether there is a vector $\boldsymbol{v}$ of length n such that $A \cdot (\boldsymbol{v} \otimes \boldsymbol{v}) = \boldsymbol{b}$ (where $\boldsymbol{x} \otimes \boldsymbol{y}$ denotes the tensor product – a vector of length n^2, which on position $n \cdot (i-1)+j$ has x_iy_j)

PCP – idea of a proof

Input: a matrix A of size $m \times n^2$, vector $\boldsymbol{b}$ of length m

Question: is there a vector $\boldsymbol{v}$ of length n such that $A\cdot(\boldsymbol{v} \otimes \boldsymbol{v})=\boldsymbol{b}$?

Verifier V expects a proof of length $2^n+2^{n^2}$, which encodes functions $f\colon \{0,1\}^n \to \{0,1\}$ and $g\colon \{0,1\}^{n^2} \to \{0,1\}$

In a correct proof, f and g are W-H codes of vectors $\boldsymbol{v}$ and $\boldsymbol{v} \otimes \boldsymbol{v}$.

1) V checks that f and g are ρ-close to linear functions.
We have already shown how to read these linear functions having f and g; below for simplicity we assume that f and g are linear.

PCP – idea of a proof

Input: a matrix A of size $m\times n^2$, vector $\boldsymbol{b}$ of length m

Question: is there a vector $\boldsymbol{v}$ of length n such that $A\cdot(\boldsymbol{v}\otimes\boldsymbol{v})=\boldsymbol{b}$?

Verifier V expects a proof of length $2^n+2^{n^2}$, which encodes functions $f\colon \{0,1\}^n\rightarrow\{0,1\}$ and $g\colon \{0,1\}^{n^2}\rightarrow\{0,1\}$

In a correct proof, f and g are W-H codes of vectors $\boldsymbol{v}$ and $\boldsymbol{v}\otimes\boldsymbol{v}$.

1) V checks that f and g are ρ-close to linear functions.

We have already shown how to read these linear functions having f and g; below for simplicity we assume that f and g are linear.

2) V checks that g encodes $\boldsymbol{v}\otimes\boldsymbol{v}$, if f encodes $\boldsymbol{v}$:

- pick randomly $\boldsymbol{x},\boldsymbol{x}'\in\{0,1\}^n$
- reject if $g(\boldsymbol{x}\otimes\boldsymbol{x}')\neq f(\boldsymbol{x})f(\boldsymbol{x}')$
- repeat 10 times

One can see that the equality $g(\boldsymbol{x}\otimes\boldsymbol{x}')=f(\boldsymbol{x})f(\boldsymbol{x}')$ always holds for a correct proof; for an incorrect proof it holds with probability $\leq 3/4$. Thus, after this test, g probably encodes $\boldsymbol{v}\otimes\boldsymbol{v}$

PCP – idea of a proof

3) V checks that $A \cdot (\mathbf{v} \otimes \mathbf{v}) = \mathbf{b}$

- ➔ the i-th equation is $A_i \cdot (\mathbf{v} \otimes \mathbf{v}) = b_i$, where A_i (the i-th row of matrix A) is a vector of length n^2
- ➔ by definition $A_i \cdot (\mathbf{v} \otimes \mathbf{v}) = g(A_i)$, thus it is enough to read $g(A_i)$ and check that $g(A_i) = b_i$

PCP – idea of a proof

3) V checks that $A\cdot(\mathbf{v}\otimes\mathbf{v})=\mathbf{b}$

- the i-th equation is $A_i\cdot(\mathbf{v}\otimes\mathbf{v})=b_i$, where A_i (the i-th row of matrix A) is a vector of length n^2
- by definition $A_i\cdot(\mathbf{v}\otimes\mathbf{v})=g(A_i)$, thus it is enough to read $g(A_i)$ and check that $g(A_i)=b_i$
- difficulty: it is not enough to check a constant number of equations
- solution: pick a random subset of equations, and check that their sum is satisfied (i.e., that $g(A_S)=b_S$, where A_S equals the sum of appropriate vectors A_i, similarly b_S)
- if a system is not satisfied, then with probability $1/2$ the sum of a random subset of equations is not satisfied

THE END