Definition 2 (NP)

A language $L \in \text{NP}$ if there exists a polynomial time, deterministic verifier V (a Turing machine), s.t.

[$x \in L$] There exists a proof string y, $|y| = \text{poly}(|x|)$, s.t. $V(x, y) =$ "accept".

[$x \notin L$] For any proof string y, $V(x, y) =$ "reject".

Note that requiring $|y| = \text{poly}(|x|)$ for $x \notin L$ does not make a difference (**why?**).

Definition 2 (NP)

A language $L \in \mathrm{NP}$ if there exists a polynomial time, deterministic verifier V (a Turing machine), s.t.

[$x \in L$] There exists a proof string y, $|y| = \mathrm{poly}(|x|)$, s.t. $V(x, y) =$ "accept".

[$x \notin L$] For any proof string y, $V(x, y) =$ "reject".

Note that requiring $|y| = \mathrm{poly}(|x|)$ for $x \notin L$ does not make a difference (**why?**).

Probabilistic Proof Verification

Definition 3 (IP)

In an interactive proof system a randomized polynomial-time verifier V (with private coin tosses) interacts with an all powerful prover P in polynomially many rounds. $L \in \mathrm{IP}$ if

[$x \in L$] There exists a strategy for P s.t. V accepts with probability 1.

[$x \notin L$] Regardless of P's strategy V accepts with probability at most $1/2$.

Probabilistic Checkable Proofs

Definition 4 (PCP)

A language $L \in \mathrm{PCP}_{c(n),s(n)}(r(n), q(n))$ if there exists a polynomial time, non-adaptive, randomized verifier V (an Oracle Turing Machine), s.t.

[$x \in L$] There exists a proof string y, s.t. $V^{\pi_y}(x) =$ "accept" with proability $\geq c(n)$.

[$x \notin L$] For any proof string y, $V^{\pi_y}(x) =$ "accept" with probability $\leq s(n)$.

The verifier uses at most $r(n)$ random bits and makes at most $q(n)$ oracle queries.

Probabilistic Checkable Proofs

An Oracle Turing Machine M is a Turing machine that has access to an oracle.

Such an oracle allows M to solve some problem in a single step.

For example having access to a TSP-oracle π_{TSP} would allow M to write a TSP-instance x on a special oracle tape and obtain the answer (yes or no) in a single step.

For such TMs one looks in addition to running time also at query complexity, i.e., how often the machine queries the oracle.

For a proof string y, π_y is an oracle that upon given an index i returns the i-th character y_i of y.

$c(n)$ is called the completeness. If not specified otw. $c(n) = 1$. Probability of accepting a correct proof.

$s(n) < c(n)$ is called the soundness. If not specified otw. $s(n) = 1/2$. Probability of accepting a wrong proof.

$r(n)$ is called the randomness complexity, i.e., how many random bits the (randomized) verifier uses.

$q(n)$ is the query complexity of the verifier.

$$\mathrm{IP} \subseteq \mathrm{PCP}_{1,1/2}(\mathrm{poly}(n), \mathrm{poly}(n))$$

We can view non-adadpative $\mathrm{PCP}_{1,1/2}(\mathrm{poly}(n), \mathrm{poly}(n))$ as the version of IP in which the prover has written down his answers to all possible queries (beforehand).

This makes it harder for the prover to cheat.

The non-cheating prover does not loose power.

Note that the above is not a proof!

- $\mathrm{PCP}(0, 0) = \mathrm{P}$
- $\mathrm{PCP}(\mathcal{O}(\log n), 0) = \mathrm{P}$
- $\mathrm{PCP}(0, \mathcal{O}(\log n)) = \mathrm{P}$
- $\mathrm{PCP}(0, \mathcal{O}(\mathrm{poly}(n))) = \mathrm{NP}$
- $\mathrm{PCP}(\mathcal{O}(\log n), \mathcal{O}(\mathrm{poly}(n))) = \mathrm{NP}$
- $\mathrm{PCP}(\mathcal{O}(\mathrm{poly}(n)), 0) = \mathrm{coRP}$
  randomized polynomial time with one sided error (positive probability of accepting a false statement)
- $\mathrm{PCP}(\mathcal{O}(\log n), \mathcal{O}(1)) = \mathrm{NP}$ (the PCP theorem)

- $\mathrm{PCP}(0,0) = \mathrm{P}$
- $\mathrm{PCP}(\mathcal{O}(\log n), 0) = \mathrm{P}$
- $\mathrm{PCP}(0, \mathcal{O}(\log n)) = \mathrm{P}$
- $\mathrm{PCP}(0, \mathcal{O}(\mathrm{poly}(n))) = \mathrm{NP}$
- $\mathrm{PCP}(\mathcal{O}(\log n), \mathcal{O}(\mathrm{poly}(n))) = \mathrm{NP}$
- $\mathrm{PCP}(\mathcal{O}(\mathrm{poly}(n)), 0) = \mathrm{coRP}$
  randomized polynomial time with one sided error (positive probability of accepting a false statement)
- $\mathrm{PCP}(\mathcal{O}(\log n), \mathcal{O}(1)) = \mathrm{NP}$ (the PCP theorem)

- $\mathrm{PCP}(0,0) = \mathrm{P}$
- $\mathrm{PCP}(\mathcal{O}(\log n), 0) = \mathrm{P}$
- $\mathrm{PCP}(0, \mathcal{O}(\log n)) = \mathrm{P}$
- $\mathrm{PCP}(0, \mathcal{O}(\mathrm{poly}(n))) = \mathrm{NP}$
- $\mathrm{PCP}(\mathcal{O}(\log n), \mathcal{O}(\mathrm{poly}(n))) = \mathrm{NP}$
- $\mathrm{PCP}(\mathcal{O}(\mathrm{poly}(n)), 0) = \mathrm{coRP}$
  randomized polynomial time with one sided error (positive probability of accepting a false statement)
- $\mathrm{PCP}(\mathcal{O}(\log n), \mathcal{O}(1)) = \mathrm{NP}$ (the PCP theorem)

- $\mathrm{PCP}(0,0) = \mathrm{P}$
- $\mathrm{PCP}(\mathcal{O}(\log n), 0) = \mathrm{P}$
- $\mathrm{PCP}(0, \mathcal{O}(\log n)) = \mathrm{P}$
- $\mathrm{PCP}(0, \mathcal{O}(\mathrm{poly}(n))) = \mathrm{NP}$
- $\mathrm{PCP}(\mathcal{O}(\log n), \mathcal{O}(\mathrm{poly}(n))) = \mathrm{NP}$
- $\mathrm{PCP}(\mathcal{O}(\mathrm{poly}(n)), 0) = \mathrm{coRP}$
  randomized polynomial time with one sided error (positive probability of accepting a false statement)
- $\mathrm{PCP}(\mathcal{O}(\log n), \mathcal{O}(1)) = \mathrm{NP}$ (the PCP theorem)

- $\mathrm{PCP}(0,0) = \mathrm{P}$
- $\mathrm{PCP}(\mathcal{O}(\log n), 0) = \mathrm{P}$
- $\mathrm{PCP}(0, \mathcal{O}(\log n)) = \mathrm{P}$
- $\mathrm{PCP}(0, \mathcal{O}(\mathrm{poly}(n))) = \mathrm{NP}$
- $\mathrm{PCP}(\mathcal{O}(\log n), \mathcal{O}(\mathrm{poly}(n))) = \mathrm{NP}$
- $\mathrm{PCP}(\mathcal{O}(\mathrm{poly}(n)), 0) = \mathrm{coRP}$
  randomized polynomial time with one sided error (positive probability of accepting a false statement)
- $\mathrm{PCP}(\mathcal{O}(\log n), \mathcal{O}(1)) = \mathrm{NP}$ (the PCP theorem)

- $\mathrm{PCP}(0,0) = \mathrm{P}$
- $\mathrm{PCP}(\mathcal{O}(\log n),0) = \mathrm{P}$
- $\mathrm{PCP}(0,\mathcal{O}(\log n)) = \mathrm{P}$
- $\mathrm{PCP}(0,\mathcal{O}(\mathrm{poly}(n))) = \mathrm{NP}$
- $\mathrm{PCP}(\mathcal{O}(\log n),\mathcal{O}(\mathrm{poly}(n))) = \mathrm{NP}$
- $\mathrm{PCP}(\mathcal{O}(\mathrm{poly}(n)),0) = \mathrm{coRP}$
  randomized polynomial time with one sided error (positive probability of accepting a false statement)
- $\mathrm{PCP}(\mathcal{O}(\log n),\mathcal{O}(1)) = \mathrm{NP}$ (the PCP theorem)

- $\mathrm{PCP}(0,0) = \mathrm{P}$
- $\mathrm{PCP}(\mathcal{O}(\log n),0) = \mathrm{P}$
- $\mathrm{PCP}(0,\mathcal{O}(\log n)) = \mathrm{P}$
- $\mathrm{PCP}(0,\mathcal{O}(\mathrm{poly}(n))) = \mathrm{NP}$
- $\mathrm{PCP}(\mathcal{O}(\log n),\mathcal{O}(\mathrm{poly}(n))) = \mathrm{NP}$
- $\mathrm{PCP}(\mathcal{O}(\mathrm{poly}(n)),0) = \mathrm{coRP}$
  randomized polynomial time with one sided error (positive probability of accepting a false statement)
- $\mathrm{PCP}(\mathcal{O}(\log n),\mathcal{O}(1)) = \mathrm{NP}$ (the PCP theorem)

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

PCP(poly(n), 1) means that we have a potentially exponentially long proof but we only read a constant number of bits from the proof.

The idea is to encode an NP-witness/proof (e.g. a satisfying assignment (say n bits)) by a code whose code-words have 2^n bits.

A wrong proof is either

- a code-word whose pre-image does not correspond to a satisfying assignment
- or, a sequence of bits that does not correspond to a code-word

We can detect both cases by querying a few positions.

$\mathrm{NP} \subseteq \mathrm{PCP}(\mathrm{poly}(n), 1)$

$\mathrm{PCP}(\mathrm{poly}(n), 1)$ means that we have a potentially exponentially long proof but we only read a constant number of bits from the proof.

The idea is to encode an NP-witness/proof (e.g. a satisfying assignment (say n bits)) by a code whose code-words have 2^n bits.

A wrong proof is either

- a code-word whose pre-image does not correspond to a satisfying assignment
- or, a sequence of bits that does not correspond to a code-word

We can detect both cases by querying a few positions.

$\text{NP} \subseteq \text{PCP}(\text{poly}(n), 1)$

$\text{PCP}(\text{poly}(n), 1)$ means that we have a potentially exponentially long proof but we only read a constant number of bits from the proof.

The idea is to encode an NP-witness/proof (e.g. a satisfying assignment (say n bits)) by a code whose code-words have 2^n bits.

A wrong proof is either

- a code-word whose pre-image does not correspond to a satisfying assignment
- or, a sequence of bits that does not correspond to a code-word

We can detect both cases by querying a few positions.

The Code

$u \in \{0,1\}^n$ (satisfying assignment)

Walsh-Hadamard Code:
$\mathrm{WH}_u : \{0,1\}^n \to \{0,1\}, x \mapsto x^T u$ (over $\mathrm{GF}(2)$)

The code-word for u is WH_u. We identify this function by a bit-vector of length 2^n.

The Code

Lemma 5
If $u \neq u'$ then WH_u and $\mathrm{WH}_{u'}$ differ in at least 2^{n-1} bits.

Suppose that $u - u' \neq 0$. Then

$$\mathrm{WH}_u(x) \neq \mathrm{WH}_{u'}(x) \iff (u - u')^T x \neq 0$$

This holds for 2^{n-1} different vectors x.

The Code

Lemma 5
If $u \neq u'$ *then* WH_u *and* $\mathrm{WH}_{u'}$ *differ in at least* 2^{n-1} *bits.*

Suppose that $u - u' \neq 0$. Then

$$\mathrm{WH}_u(x) \neq \mathrm{WH}_{u'}(x) \iff (u - u')^T x \neq 0$$

This holds for 2^{n-1} different vectors x.

The Code

Suppose we are given access to a function $f : \{0,1\}^n \to \{0,1\}$ and want to check whether it is a codeword.

Since the set of codewords is the set of all linear functions $\{0,1\}^n$ to $\{0,1\}$ we can check

$$f(x+y) = f(x) + f(y)$$

for all 2^{2n} pairs x, y. But that's not very efficient.

The Code

Suppose we are given access to a function $f : \{0,1\}^n \rightarrow \{0,1\}$ and want to check whether it is a codeword.

Since the set of codewords is the set of all linear functions $\{0,1\}^n$ to $\{0,1\}$ we can check

$$f(x+y) = f(x) + f(y)$$

for all 2^{2n} pairs x, y. But that's not very efficient.

Can we just check a constant number of positions?

Definition 6

Let $\rho \in [0,1]$. We say that $f, g : \{0,1\}^n \to \{0,1\}$ are ρ-close if

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

Theorem 7

Let $f : \{0,1\}^n \to \{0,1\}$ with

$$\Pr_{x,y \in \{0,1\}^n}\Big[f(x) + f(y) = f(x+y)\Big] \geq \rho > \frac{1}{2} \ .$$

Then there is a linear function $\tilde{f}$ such that f and $\tilde{f}$ are ρ-close.

Definition 6

Let $\rho \in [0,1]$. We say that $f, g : \{0,1\}^n \to \{0,1\}$ are ρ-close if

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

Theorem 7

Let $f : \{0,1\}^n \to \{0,1\}$ with

$$\Pr_{x,y\in\{0,1\}^n}\Big[f(x) + f(y) = f(x+y)\Big] \geq \rho > \frac{1}{2} \ .$$

Then there is a linear function $\tilde{f}$ such that f and $\tilde{f}$ are ρ-close.

We need $\mathcal{O}(1/\delta)$ trials to be sure that f is $(1-\delta)$-close to a linear function with (arbitrary) constant probability.

Suppose for $\delta < 1/4$ f is $(1-\delta)$-close to some linear function $\tilde{f}$.

$\tilde{f}$ is uniquely defined by f, since linear functions differ on at least half their inputs.

Suppose we are given $x \in \{0,1\}^n$ and access to f. Can we compute $\tilde{f}(x)$ using only constant number of queries?

Suppose for $\delta < 1/4$ f is $(1-\delta)$-close to some linear function $\tilde{f}$.

$\tilde{f}$ is uniquely defined by f, since linear functions differ on at least half their inputs.

Suppose we are given $x \in \{0,1\}^n$ and access to f. Can we compute $\tilde{f}(x)$ using only constant number of queries?

Suppose for $\delta < 1/4$ f is $(1-\delta)$-close to some linear function $\tilde{f}$.

$\tilde{f}$ is uniquely defined by f, since linear functions differ on at least half their inputs.

Suppose we are given $x \in \{0,1\}^n$ and access to f. Can we compute $\tilde{f}(x)$ using only constant number of queries?

Suppose we are given $x \in \{0,1\}^n$ and access to f. Can we compute $\tilde{f}(x)$ using only constant number of queries?

1. Choose $x' \in \{0,1\}^n$ u.a.r.
2. Set $x'' := x + x'$.
3. Let $y' = f(x')$ and $y'' = f(x'')$.
4. Output $y' + y''$.

x' and x'' are uniformly distributed (albeit dependent). With probability at least $1 - 2\delta$ we have $f(x') = \tilde{f}(x')$ and $f(x'') = \tilde{f}(x'')$.

Then we can compute $\tilde{f}(x)$.

This technique is known as local decoding of the Walsh-Hadamard code.

Suppose we are given $x \in \{0,1\}^n$ and access to f. Can we compute $\tilde{f}(x)$ using only constant number of queries?

1. Choose $x' \in \{0,1\}^n$ u.a.r.
2. Set $x'' := x + x'$.
3. Let $y' = f(x')$ and $y'' = f(x'')$.
4. Output $y' + y''$.

x' and x'' are uniformly distributed (albeit dependent). With probability at least $1 - 2\delta$ we have $f(x') = \tilde{f}(x')$ and $f(x'') = \tilde{f}(x'')$.

Then we can compute $\tilde{f}(x)$.

This technique is known as local decoding of the Walsh-Hadamard code.

Suppose we are given $x \in \{0,1\}^n$ and access to f. Can we compute $\tilde{f}(x)$ using only constant number of queries?

1. Choose $x' \in \{0,1\}^n$ u.a.r.
2. Set $x'' := x + x'$.
3. Let $y' = f(x')$ and $y'' = f(x'')$.
4. Output $y' + y''$.

x' and x'' are uniformly distributed (albeit dependent). With probability at least $1 - 2\delta$ we have $f(x') = \tilde{f}(x')$ and $f(x'') = \tilde{f}(x'')$.

Then we can compute $\tilde{f}(x)$.

This technique is known as local decoding of the Walsh-Hadamard code.

$\mathrm{NP} \subseteq \mathrm{PCP}(\mathrm{poly}(n), 1)$

We show that $\mathrm{QUADEQ} \in \mathrm{PCP}(\mathrm{poly}(n), 1)$. The theorem follows since any PCP-class is closed under polynomial time reductions.

introduce QUADEQ...

prove NP-completeness...

Let A, b be an instance of QUADEQ. Let u be a satisfying assignment.

The correct PCP-proof will be the Walsh-Hadamard encodings of u and $u \otimes u$. The verifier will accept such a proof with probability 1.

We have to make sure that we reject proofs that do not correspond to codewords for vectors of the form u, and $u \otimes u$.

We also have to reject proofs that correspond to codewords for vectors of the form z, and $z \otimes z$, where z is not a satisfying assignment.

Step 1. Linearity Test.

The proof contains $2^n + 2^{n^2}$ bits. This is interpreted as a pair of functions $f : \{0,1\}^n \to \{0,1\}$ and $g : \{0,1\}^{n^2} \to \{0,1\}$.

We do a 0.99-linearity test for both functions (requires a constant number of queries).

We also assume that the remaining constant number of (random) accesses only hit points where $f(x) = \tilde{f}(x)$.

Hence, our proof will only see $\tilde{f}$ and therefore we use f for $\tilde{f}$, in the following (similar for g, $\tilde{g}$).

Step 1. Linearity Test.

The proof contains $2^n + 2^{n^2}$ bits. This is interpreted as a pair of functions $f : \{0,1\}^n \to \{0,1\}$ and $g : \{0,1\}^{n^2} \to \{0,1\}$.

We do a 0.99-linearity test for both functions (requires a constant number of queries).

We also assume that the remaining constant number of (random) accesses only hit points where $f(x) = \tilde{f}(x)$.

Hence, our proof will only see $\tilde{f}$ and therefore we use f for $\tilde{f}$, in the following (similar for g, $\tilde{g}$).

Step 1. Linearity Test.
The proof contains $2^n + 2^{n^2}$ bits. This is interpreted as a pair of functions $f : \{0,1\}^n \to \{0,1\}$ and $g : \{0,1\}^{n^2} \to \{0,1\}$.

We do a 0.99-linearity test for both functions (requires a constant number of queries).

We also assume that the remaining constant number of (random) accesses only hit points where $f(x) = \tilde{f}(x)$.

Hence, our proof will only see $\tilde{f}$ and therefore we use f for $\tilde{f}$, in the following (similar for g, $\tilde{g}$).

Step 1. Linearity Test.
The proof contains $2^n + 2^{n^2}$ bits. This is interpreted as a pair of functions $f : \{0,1\}^n \to \{0,1\}$ and $g : \{0,1\}^{n^2} \to \{0,1\}$.

We do a 0.99-linearity test for both functions (requires a constant number of queries).

We also assume that the remaining constant number of (random) accesses only hit points where $f(x) = \tilde{f}(x)$.

Hence, our proof will only see $\tilde{f}$ and therefore we use f for $\tilde{f}$, in the following (similar for g, $\tilde{g}$).

Step 2. Verify that g encodes $u \otimes u$ where u is string encoded by f.

$f(r) = u^T r$ and $g(z) = w^T z$ since f, g are linear.

- choose r, r' independently, u.a.r. from $\{0,1\}^n$
- if $f(r)f(r') \neq g(r \otimes r')$ reject
- repeat 3 times

A correct proof survives the test

$$f(r) \cdot f(r')$$

A correct proof survives the test

$$f(r) \cdot f(r') = u^T r \cdot u^T r'$$

A correct proof survives the test

$$f(r) \cdot f(r') = u^T r \cdot u^T r' = \Big(\sum_i u_i r_i\Big) \cdot \Big(\sum_j u_j r'_j\Big)$$

A correct proof survives the test

$$f(r) \cdot f(r') = u^T r \cdot u^T r' = \Big(\sum_i u_i r_i\Big) \cdot \Big(\sum_j u_j r'_j\Big)$$

$$= \sum_{ij} u_i u_j r_i r'_j$$

A correct proof survives the test

$$f(r) \cdot f(r') = u^T r \cdot u^T r' = \Big(\sum_i u_i r_i\Big) \cdot \Big(\sum_j u_j r'_j\Big)$$

$$= \sum_{ij} u_i u_j r_i r'_j = (u \otimes u)^T (r \otimes r')$$

A correct proof survives the test

$$
\begin{aligned}
f(r) \cdot f(r') &= u^T r \cdot u^T r' = \Big(\sum_i u_i r_i\Big) \cdot \Big(\sum_j u_j r'_j\Big) \\
&= \sum_{ij} u_i u_j r_i r'_j = (u \otimes u)^T (r \otimes r') = g(r \otimes r')
\end{aligned}
$$

Suppose that the proof is not correct and $w \neq u \otimes u$.

Suppose that the proof is not correct and $w \neq u \otimes u$.

Let W be $n \times n$-matrix with entries from w. Let U be matrix with $U_{ij} = u_i \cdot u_j$ (entries from $u \otimes u$).

Suppose that the proof is not correct and $w \neq u \otimes u$.

Let W be $n \times n$-matrix with entries from w. Let U be matrix with $U_{ij} = u_i \cdot u_j$ (entries from $u \otimes u$).

$$g(r \otimes r')$$

Suppose that the proof is not correct and $w \neq u \otimes u$.

Let W be $n \times n$-matrix with entries from w. Let U be matrix with $U_{ij} = u_i \cdot u_j$ (entries from $u \otimes u$).

$$g(r \otimes r') = w^T(r \otimes r')$$

Suppose that the proof is not correct and $w \neq u \otimes u$.

Let W be $n \times n$-matrix with entries from w. Let U be matrix with $U_{ij} = u_i \cdot u_j$ (entries from $u \otimes u$).

$$g(r \otimes r') = w^T(r \otimes r') = \sum_{ij} w_{ij} r_i r'_j$$

Suppose that the proof is not correct and $w \neq u \otimes u$.

Let W be $n \times n$-matrix with entries from w. Let U be matrix with $U_{ij} = u_i \cdot u_j$ (entries from $u \otimes u$).

$$g(r \otimes r') = w^T(r \otimes r') = \sum_{ij} w_{ij} r_i r'_j = r^T W r'$$

Suppose that the proof is not correct and $w \neq u \otimes u$.

Let W be $n \times n$-matrix with entries from w. Let U be matrix with $U_{ij} = u_i \cdot u_j$ (entries from $u \otimes u$).

$$g(r \otimes r') = w^T(r \otimes r') = \sum_{ij} w_{ij} r_i r'_j = r^T W r'$$

$$f(r)f(r')$$

Suppose that the proof is not correct and $w \neq u \otimes u$.

Let W be $n \times n$-matrix with entries from w. Let U be matrix with $U_{ij} = u_i \cdot u_j$ (entries from $u \otimes u$).

$$g(r \otimes r') = w^T (r \otimes r') = \sum_{ij} w_{ij} r_i r'_j = r^T W r'$$

$$f(r) f(r') = u^T r \cdot u^T r'$$

Suppose that the proof is not correct and $w \neq u \otimes u$.

Let W be $n \times n$-matrix with entries from w. Let U be matrix with $U_{ij} = u_i \cdot u_j$ (entries from $u \otimes u$).

$$g(r \otimes r') = w^T(r \otimes r') = \sum_{ij} w_{ij} r_i r'_j = r^T W r'$$

$$f(r)f(r') = u^T r \cdot u^T r' = r^T U r'$$

Suppose that the proof is not correct and $w \neq u \otimes u$.

Let W be $n \times n$-matrix with entries from w. Let U be matrix with $U_{ij} = u_i \cdot u_j$ (entries from $u \otimes u$).

$$g(r \otimes r') = w^T(r \otimes r') = \sum_{ij} w_{ij} r_i r'_j = r^T W r'$$

$$f(r)f(r') = u^T r \cdot u^T r' = r^T U r'$$

If $U \neq W$ then $Wr' \neq Ur'$ with probability at least 1/2. Then $r^T W r' \neq r^T U r'$ with probability at least 1/4.

Step 3. Verify that f encodes satisfying assignment.

We need to check

$$A_k(u \otimes u) = b_k$$

where A_k is the k-th row of the constraint matrix. But the left hand side is just $g(A_k^T)$.

We can handle this by a single query but checking all constraints would take $\mathcal{O}(m)$ steps.

We compute rA, where $r \in_R \{0,1\}^m$. If u is not a satisfying assignment then with probability 1/2 the vector r will hit an odd number of violated constraint.

In this case $rA(u \otimes u) \neq rb_k$. The left hand side is equal to $g(A^T r^T)$.

Step 3. Verify that f encodes satisfying assignment.

We need to check

$$A_k(u \otimes u) = b_k$$

where A_k is the k-th row of the constraint matrix. But the left hand side is just $g(A_k^T)$.

We can handle this by a single query but checking all constraints would take $\mathcal{O}(m)$ steps.

We compute rA, where $r \in_R \{0,1\}^m$. If u is not a satisfying assignment then with probability 1/2 the vector r will hit an odd number of violated constraint.

In this case $rA(u \otimes u) \neq rb_k$. The left hand side is equal to $g(A^T r^T)$.

Step 3. Verify that f encodes satisfying assignment.

We need to check

$$A_k(u \otimes u) = b_k$$

where A_k is the k-th row of the constraint matrix. But the left hand side is just $g(A_k^T)$.

We can handle this by a single query but checking all constraints would take $\mathcal{O}(m)$ steps.

We compute rA, where $r \in_R \{0,1\}^m$. If u is not a satisfying assignment then with probability 1/2 the vector r will hit an odd number of violated constraint.

In this case $rA(u \otimes u) \neq rb_k$. The left hand side is equal to $g(A^T r^T)$.

Step 3. Verify that f encodes satisfying assignment.

We need to check

$$A_k(u \otimes u) = b_k$$

where A_k is the k-th row of the constraint matrix. But the left hand side is just $g(A_k^T)$.

We can handle this by a single query but checking all constraints would take $\mathcal{O}(m)$ steps.

We compute rA, where $r \in_R \{0,1\}^m$. If u is not a satisfying assignment then with probability 1/2 the vector r will hit an odd number of violated constraint.

In this case $rA(u \otimes u) \neq rb_k$. The left hand side is equal to $g(A^T r^T)$.

Theorem 7

Let $f : \{0,1\}^n \to \{0,1\}$ *with*

$$\Pr_{x,y\in\{0,1\}^n}\Big[f(x)+f(y)=f(x+y)\Big] \geq \rho > \frac{1}{2} \enspace .$$

Then there is a linear function $\tilde{f}$ *such that* f *and* $\tilde{f}$ *are* ρ*-close.*

Fourier Transform over $\mathrm{GF}(2)$

In the following we use $\{-1,1\}$ instead of $\{0,1\}$. We map $b \in \{0,1\}$ to $(-1)^b$.

This turns summation into multiplication.

The set of function $f : \{-1,1\} \rightarrow \mathbb{R}$ form a 2^n-dimensional Hilbert space.

Hilbert space

- addition $(f+g)(x) = f(x) + g(x)$
- scalar multiplication $(\alpha f)(x) = \alpha f(x)$
- inner product $\langle f, g\rangle = E_{x\in\{0,1\}^n}[f(x)g(x)]$
  (bilinear, $\langle f, f\rangle \geq 0$, and $\langle f, f\rangle = 0 \Rightarrow f = 0$)
- completeness: any sequence x_k of vectors for which

$$\sum_{k=1}^{\infty} \|x_k\| < \infty \text{ fulfills } \left\| L - \sum_{k=1}^{N} x_k \right\| \to 0$$

for some vector L.

standard basis

$$e_x(y) = \begin{cases} 1 & x = y \\ 0 & \text{otw.} \end{cases}$$

Then, $f(x) = \sum_x \alpha_x e_x$ where $\alpha_x = f(x)$, this means the functions e_x form a basis. This basis is orthonormal.

fourier basis

For $\alpha \subseteq [n]$ define

$$\chi_\alpha(x) = \prod_{i \in \alpha} x_i$$

fourier basis

For $\alpha \subseteq [n]$ define

$$\chi_\alpha(x) = \prod_{i \in \alpha} x_i$$

Note that

$$\langle \chi_\alpha, \chi_\beta \rangle$$

fourier basis

For $\alpha \subseteq [n]$ define

$$\chi_\alpha(x) = \prod_{i \in \alpha} x_i$$

Note that

$$\langle \chi_\alpha, \chi_\beta \rangle = E_x \Big[\chi_\alpha(x) \chi_\beta(x) \Big]$$

fourier basis

For $\alpha \subseteq [n]$ define

$$\chi_\alpha(x) = \prod_{i \in \alpha} x_i$$

Note that

$$\langle \chi_\alpha, \chi_\beta \rangle = E_x\Big[\chi_\alpha(x)\chi_\beta(x)\Big] = E_x\Big[\chi_{\alpha \triangle \beta}(x)\Big]$$

fourier basis

For $\alpha \subseteq [n]$ define

$$\chi_\alpha(x) = \prod_{i \in \alpha} x_i$$

Note that

$$\langle \chi_\alpha, \chi_\beta \rangle = E_x\Big[\chi_\alpha(x)\chi_\beta(x)\Big] = E_x\Big[\chi_{\alpha \triangle \beta}(x)\Big] = \begin{cases} 1 & \alpha = \beta \\ 0 & \text{otw.} \end{cases}$$

fourier basis

For $\alpha \subseteq [n]$ define

$$\chi_\alpha(x) = \prod_{i \in \alpha} x_i$$

Note that

$$\langle \chi_\alpha, \chi_\beta \rangle = E_x\Big[\chi_\alpha(x)\chi_\beta(x)\Big] = E_x\Big[\chi_{\alpha \triangle \beta}(x)\Big] = \begin{cases} 1 & \alpha = \beta \\ 0 & \text{otw.} \end{cases}$$

This means the χ_α's also define an orthonormal basis. (since we have 2^n orthonormal vectors...)

A function χ_α multiplies a set of x_i's. Back in the $\mathrm{GF}(2)$-world this means summing a set of z_i's where $x_i = (-1)^{z_i}$.

This means the function χ_α correspond to linear functions in the $\mathrm{GF}(2)$ world.

We can write any function $f : \{-1,1\}^n \to \mathbb{R}$ as

$$f = \sum_\alpha \hat{f}_\alpha \chi_\alpha$$

We call $\hat{f}_\alpha$ the α^{th} Fourier coefficient.

Lemma 8

1. $\langle f, g\rangle = \sum_\alpha f_\alpha g_\alpha$
2. $\langle f, f\rangle = \sum_\alpha f_\alpha^2$

Note that for Boolean functions $f : \{-1,1\}^n \to \{-1,1\}$, $\langle f, f\rangle = 1$.

Linearity Test

GF(2)

We want to show that if $\Pr_{x,y}[f(x) + f(y) = f(x + y)]$ is large than f has a large agreement with a linear function.

Linearity Test

GF(2)

We want to show that if $\Pr_{x,y}[f(x) + f(y) = f(x+y)]$ is large than f has a large agreement with a linear function.

Hilbert space (we prove)

Suppose that $f : \{+1,-1\}^n \to \{-1,1\}$ satisfies $\Pr_{x,y}[f(x)f(y) = f(xy)] \geq \frac{1}{2} + \epsilon$. Then there is some $\alpha \subseteq [n]$, s.t. $\hat{f}_\alpha \geq 2\epsilon$.

Linearity Test

GF(2)
We want to show that if $\Pr_{x,y}[f(x) + f(y) = f(x+y)]$ is large than f has a large agreement with a linear function.

Hilbert space (we prove)
Suppose that $f : \{+1,-1\}^n \to \{-1,1\}$ satisfies $\Pr_{x,y}[f(x)f(y) = f(xy)] \geq \frac{1}{2} + \epsilon$. Then there is some $\alpha \subseteq [n]$, s.t. $\hat{f}_\alpha \geq 2\epsilon$.

For Boolean functions $\langle f, g \rangle$ is the fraction of inputs on which f, g agree **minus** the fraction of inputs on which they disagree.

Linearity Test

GF(2)
We want to show that if $\Pr_{x,y}[f(x) + f(y) = f(x + y)]$ is large than f has a large agreement with a linear function.

Hilbert space (we prove)
Suppose that $f : \{+1, -1\}^n \to \{-1, 1\}$ satisfies $\Pr_{x,y}[f(x)f(y) = f(xy)] \geq \frac{1}{2} + \epsilon$. Then there is some $\alpha \subseteq [n]$, s.t. $\hat{f}_\alpha \geq 2\epsilon$.

For Boolean functions $\langle f, g \rangle$ is the fraction of inputs on which f, g agree **minus** the fraction of inputs on which they disagree.

$$2\epsilon \leq \hat{f}_\alpha$$

Linearity Test

GF(2)
We want to show that if $\Pr_{x,y}[f(x) + f(y) = f(x + y)]$ is large than f has a large agreement with a linear function.

Hilbert space (we prove)
Suppose that $f : \{+1, -1\}^n \to \{-1, 1\}$ satisfies $\Pr_{x,y}[f(x)f(y) = f(xy)] \geq \frac{1}{2} + \epsilon$. Then there is some $\alpha \subseteq [n]$, s.t. $\hat{f}_\alpha \geq 2\epsilon$.

For Boolean functions $\langle f, g\rangle$ is the fraction of inputs on which f, g agree **minus** the fraction of inputs on which they disagree.

$$2\epsilon \leq \hat{f}_\alpha = \langle f, \chi_\alpha\rangle$$

Linearity Test

GF(2)
We want to show that if $\Pr_{x,y}[f(x) + f(y) = f(x+y)]$ is large than f has a large agreement with a linear function.

Hilbert space (we prove)
Suppose that $f : \{+1,-1\}^n \rightarrow \{-1,1\}$ satisfies $\Pr_{x,y}[f(x)f(y) = f(xy)] \geq \frac{1}{2} + \epsilon$. Then there is some $\alpha \subseteq [n]$, s.t. $\hat{f}_\alpha \geq 2\epsilon$.

For Boolean functions $\langle f, g \rangle$ is the fraction of inputs on which f, g agree **minus** the fraction of inputs on which they disagree.

$$2\epsilon \leq \hat{f}_\alpha = \langle f, \chi_\alpha \rangle = \text{agree} - \text{disagree}$$

Linearity Test

GF(2)
We want to show that if $\Pr_{x,y}[f(x)+f(y)=f(x+y)]$ is large than f has a large agreement with a linear function.

Hilbert space (we prove)
Suppose that $f:\{+1,-1\}^n \rightarrow \{-1,1\}$ satisfies $\Pr_{x,y}[f(x)f(y)=f(xy)] \geq \frac{1}{2}+\epsilon$. Then there is some $\alpha \subseteq [n]$, s.t. $\hat{f}_\alpha \geq 2\epsilon$.

For Boolean functions $\langle f,g\rangle$ is the fraction of inputs on which f,g agree **minus** the fraction of inputs on which they disagree.

$$2\epsilon \leq \hat{f}_\alpha = \langle f,\chi_\alpha\rangle = \text{agree} - \text{disagree} = 2\text{agree} - 1$$

Linearity Test

GF(2)
We want to show that if $\Pr_{x,y}[f(x) + f(y) = f(x + y)]$ is large than f has a large agreement with a linear function.

Hilbert space (we prove)
Suppose that $f : \{+1, -1\}^n \to \{-1, 1\}$ satisfies $\Pr_{x,y}[f(x)f(y) = f(xy)] \geq \frac{1}{2} + \epsilon$. Then there is some $\alpha \subseteq [n]$, s.t. $\hat{f}_\alpha \geq 2\epsilon$.

For Boolean functions $\langle f, g\rangle$ is the fraction of inputs on which f, g agree **minus** the fraction of inputs on which they disagree.

$$2\epsilon \leq \hat{f}_\alpha = \langle f, \chi_\alpha\rangle = \text{agree} - \text{disagree} = 2\text{agree} - 1$$

This gives that the agreement between f and χ_α is at least $\frac{1}{2} + \epsilon$.

Linearity Test

$$\Pr_{x,y}[f(xy) = f(x)f(y)] \geq \frac{1}{2} + \epsilon$$

is equivalent to

$$E_{x,y}[f(xy)f(x)f(y)] = \text{agreement} - \text{disagreement} \geq 2\epsilon$$

$$2\epsilon \le E_{x,y}\left[f(xy)f(x)f(y) \right]$$

$$
\begin{aligned}
2\epsilon &\le E_{x,y}\Big[f(xy)f(x)f(y)\Big] \\
&= E_{x,y}\Big[\Big(\sum\nolimits_\alpha \hat{f}_\alpha \chi_\alpha(xy)\Big)\cdot\Big(\sum\nolimits_\beta \hat{f}_\beta \chi_\beta(x)\Big)\cdot\Big(\sum\nolimits_\gamma \hat{f}_\gamma \chi_\gamma(y)\Big)\Big]
\end{aligned}
$$

$$
\begin{aligned}
2\epsilon &\le E_{x,y}\Big[f(xy)f(x)f(y)\Big] \\
&= E_{x,y}\Big[\Big(\sum_\alpha \hat{f}_\alpha \chi_\alpha(xy)\Big)\cdot\Big(\sum_\beta \hat{f}_\beta \chi_\beta(x)\Big)\cdot\Big(\sum_\gamma \hat{f}_\gamma \chi_\gamma(y)\Big)\Big] \\
&= E_{x,y}\Big[\sum_{\alpha,\beta,\gamma} \hat{f}_\alpha \hat{f}_\beta \hat{f}_\gamma \chi_\alpha(x)\chi_\alpha(y)\chi_\beta(x)\chi_\gamma(y)\Big]
\end{aligned}
$$

$$
\begin{aligned}
2\epsilon &\le E_{x,y}\Big[f(xy)f(x)f(y)\Big] \\
&= E_{x,y}\Big[\Big(\sum\nolimits_{\alpha}\hat{f}_\alpha\chi_\alpha(xy)\Big)\cdot\Big(\sum\nolimits_{\beta}\hat{f}_\beta\chi_\beta(x)\Big)\cdot\Big(\sum\nolimits_{\gamma}\hat{f}_\gamma\chi_\gamma(y)\Big)\Big] \\
&= E_{x,y}\Big[\sum\nolimits_{\alpha,\beta,\gamma}\hat{f}_\alpha\hat{f}_\beta\hat{f}_\gamma\chi_\alpha(x)\chi_\alpha(y)\chi_\beta(x)\chi_\gamma(y)\Big] \\
&= \sum\nolimits_{\alpha,\beta,\gamma}\hat{f}_\alpha\hat{f}_\beta\hat{f}_\gamma\cdot E_x\Big[\chi_\alpha(x)\chi_\beta(x)\Big]E_y\Big[\chi_\alpha(y)\chi_\gamma(y)\Big]
\end{aligned}
$$

$$
\begin{aligned}
2\epsilon &\le E_{x,y}\Big[f(xy)f(x)f(y)\Big] \\
&= E_{x,y}\Big[\Big(\sum\nolimits_{\alpha}\hat{f}_\alpha\chi_\alpha(xy)\Big)\cdot\Big(\sum\nolimits_{\beta}\hat{f}_\beta\chi_\beta(x)\Big)\cdot\Big(\sum\nolimits_{\gamma}\hat{f}_\gamma\chi_\gamma(y)\Big)\Big] \\
&= E_{x,y}\Big[\sum\nolimits_{\alpha,\beta,\gamma}\hat{f}_\alpha\hat{f}_\beta\hat{f}_\gamma\chi_\alpha(x)\chi_\alpha(y)\chi_\beta(x)\chi_\gamma(y)\Big] \\
&= \sum\nolimits_{\alpha,\beta,\gamma}\hat{f}_\alpha\hat{f}_\beta\hat{f}_\gamma\cdot E_x\Big[\chi_\alpha(x)\chi_\beta(x)\Big]E_y\Big[\chi_\alpha(y)\chi_\gamma(y)\Big] \\
&= \sum_\alpha \hat{f}_\alpha^3
\end{aligned}
$$

$$
\begin{aligned}
2\epsilon &\le E_{x,y}\Big[f(xy)f(x)f(y)\Big] \\
&= E_{x,y}\Big[\Big(\sum\nolimits_{\alpha}\hat{f}_{\alpha}\chi_{\alpha}(xy)\Big)\cdot\Big(\sum\nolimits_{\beta}\hat{f}_{\beta}\chi_{\beta}(x)\Big)\cdot\Big(\sum\nolimits_{\gamma}\hat{f}_{\gamma}\chi_{\gamma}(y)\Big)\Big] \\
&= E_{x,y}\Big[\sum\nolimits_{\alpha,\beta,\gamma}\hat{f}_{\alpha}\hat{f}_{\beta}\hat{f}_{\gamma}\chi_{\alpha}(x)\chi_{\alpha}(y)\chi_{\beta}(x)\chi_{\gamma}(y)\Big] \\
&= \sum\nolimits_{\alpha,\beta,\gamma}\hat{f}_{\alpha}\hat{f}_{\beta}\hat{f}_{\gamma}\cdot E_x\Big[\chi_{\alpha}(x)\chi_{\beta}(x)\Big]E_y\Big[\chi_{\alpha}(y)\chi_{\gamma}(y)\Big] \\
&= \sum_{\alpha}\hat{f}_{\alpha}^3 \\
&\le \max_{\alpha}\hat{f}_{\alpha}\cdot\sum_{\alpha}\hat{f}_{\alpha}^2 = \max_{\alpha}\hat{f}_{\alpha}
\end{aligned}
$$

Probabilistic proof for Graph NonIsomorphism

GNI is the language of pairs of non-isomorphic graphs

Probabilistic proof for Graph NonIsomorphism

GNI is the language of pairs of non-isomorphic graphs

Verifier gets input (G_0, G_1) (two graphs with n-nodes)

Probabilistic proof for Graph NonIsomorphism

GNI is the language of pairs of non-isomorphic graphs

Verifier gets input (G_0, G_1) (two graphs with n-nodes)

It expects a proof of the following form:

- For any labeled n-node graph H the H's bit $P[H]$ of the proof fulfills

$$
\begin{aligned}
G_0 \equiv H &\implies P[H] = 0 \\
G_1 \equiv H &\implies P[H] = 1 \\
G_0, G_1 \not\equiv H &\implies P[H] = \text{arbitrary}
\end{aligned}
$$

Probabilistic proof for Graph NonIsomorphism

Verifier:

- choose $b \in \{0, 1\}$ at random
- take graph G_b and apply a random permutation to obtain a labeled graph H
- check whether $P[H] = b$

Probabilistic proof for Graph NonIsomorphism

Verifier:

- choose $b \in \{0, 1\}$ at random
- take graph G_b and apply a random permutation to obtain a labeled graph H
- check whether $P[H] = b$

If $G_0 \not\equiv G_1$ then by using the obvious proof the verifier will always accept.

Probabilistic proof for Graph NonIsomorphism

Verifier:

- choose $b \in \{0, 1\}$ at random
- take graph G_b and apply a random permutation to obtain a labeled graph H
- check whether $P[H] = b$

If $G_0 \not\equiv G_1$ then by using the obvious proof the verifier will always accept.

If $G_0 \not\equiv G_1$ a proof only accepts with probability $1/2$.

- suppose $\pi(G_0) = G_1$
- if we accept for $b = 1$ and permutation π_{rand} we reject for permutation $b = 0$ and $\pi_{\text{rand}} \circ \pi$

How to show Harndess of Approximation?

Decision version of optimization problems:

Suppose we have some maximization problem.

How to show Harndess of Approximation?

Decision version of optimization problems:
Suppose we have some maximization problem.

The corresponding decision problem equips each instance with a parameter k and asks whether we can obtain a solution value of at least k. (where infeasible solutions are assumed to have value $-\infty$)

How to show Harndess of Approximation?

Decision version of optimization problems:
Suppose we have some maximization problem.

The corresponding decision problem equips each instance with a parameter k and asks whether we can obtain a solution value of at least k. (where infeasible solutions are assumed to have value $-\infty$)

(Analogous for minimization problems.)

How to show Harndess of Approximation?

Decision version of optimization problems:
Suppose we have some maximization problem.

The corresponding decision problem equips each instance with a parameter k and asks whether we can obtain a solution value of at least k. (where infeasible solutions are assumed to have value $-\infty$)

(Analogous for minimization problems.)

This is the standard way to show that some optimization problem is e.g. NP-hard.

How to show Harndess of Approximation?

Gap version of optimization problems:

Suppose we have some maximization problem.

How to show Harndess of Approximation?

Gap version of optimization problems:

Suppose we have some maximization problem.

The corresponding (α, β)-gap problem asks the following:

How to show Harndess of Approximation?

Gap version of optimization problems:

Suppose we have some maximization problem.

The corresponding (α, β)-gap problem asks the following:

Suppose we are given an instance I and a promise that either $\text{opt}(I) \geq \beta$ or $\text{opt}(I) \leq \alpha$. Can we differentiate between these two cases?

How to show Harndess of Approximation?

Gap version of optimization problems:

Suppose we have some maximization problem.

The corresponding (α, β)-gap problem asks the following:

Suppose we are given an instance I and a promise that either $\text{opt}(I) \geq \beta$ or $\text{opt}(I) \leq \alpha$. Can we differentiate between these two cases?

An algorithm A has to output

- $A(I) = 1$ if $\text{opt}(I) \geq \beta$
- $A(I) = 0$ if $\text{opt}(I) \leq \alpha$
- $A(I) =$ arbitrary, otw

How to show Harndess of Approximation?

Gap version of optimization problems:
Suppose we have some maximization problem.

The corresponding (α, β)-gap problem asks the following:

Suppose we are given an instance I and a promise that either $\text{opt}(I) \geq \beta$ or $\text{opt}(I) \leq \alpha$. Can we differentiate between these two cases?

An algorithm A has to output

- $A(I) = 1$ if $\text{opt}(I) \geq \beta$
- $A(I) = 0$ if $\text{opt}(I) \leq \alpha$
- $A(I) =$ arbitrary, otw

Note that this is not a decision problem

An approximation algorithm with approximation guarantee $c \leq \beta/\alpha$ can solve an (α, β)-gap problem.

Constraint Satisfaction Problem

A qCSP ϕ consists of m n-ary Boolean functions $\phi_1, \dots, \phi_m$ (constraints), where each function only depends on q inputs. The goal is to maximize the number of satisifed constraints.

- $u \in \{0,1\}^n$ satsifies constraint ϕ_i if $\phi_i(u) = 1$
- $r(u) := \sum_i \phi_i(u)/m$ is fraction of satisfied constraints
- $\text{value}(\phi) = \max_u r(u)$
- ϕ is satisfiable if $\text{value}(\phi) = 1$.

3SAT is a constraint satsifaction problem with $q = 3$.

Constraint Satisfaction Problem

GAP version:

A ρGAPqCSP ϕ consists of m n-ary Boolean functions $\phi_1, \ldots, \phi_m$ (constraints), where each function only depends on q inputs. We know that either ϕ is satisfiable or $\mathrm{value}(\phi) < \rho$, and want to differentiate between these cases.

ρGAPqCSP is NP-hard if for any $L \in \mathrm{NP}$ there is a polytime computable function f mapping strings to instances of qCSP s.t.

- $x \in L \Longrightarrow \mathrm{value}(f(x)) = 1$
- $x \notin L \Longrightarrow \mathrm{value}(f(x)) < \rho$

Theorem 9

There exists constants q, ρ such that ρGAPqCSP is NP-hard.

We know that $\mathrm{NP} \subseteq \mathrm{PCP}(\log n, 1)$.

We reduce 3SAT to ρGAPqCSP.

3SAT has a PCP system in which the verifier makes a constant number of queries (q), and uses $c \log n$ random bits (for some c).

For input x and $r \in \{0,1\}^{c \log n}$ define

- $V_{x,r}$ as function that maps a proof π to the result (0/1) computed by the verifier when using proof π, instance x and random coins r.
- $V_{x,r}$ only depends on q bits of the proof

For any x the collection ϕ of the $V_{x,r}$'s over all r is polynomial size qCSP.

ϕ can be computed in polynomial time.

We know that $\mathrm{NP} \subseteq \mathrm{PCP}(\log n, 1)$.

We reduce 3SAT to ρGAPqCSP.

3SAT has a PCP system in which the verifier makes a constant number of queries (q), and uses $c \log n$ random bits (for some c).

For input x and $r \in \{0,1\}^{c \log n}$ define

- $V_{x,r}$ as function that maps a proof π to the result (0/1) computed by the verifier when using proof π, instance x and random coins r.
- $V_{x,r}$ only depends on q bits of the proof

For any x the collection ϕ of the $V_{x,r}$'s over all r is polynomial size qCSP.

ϕ can be computed in polynomial time.

We know that $\mathrm{NP} \subseteq \mathrm{PCP}(\log n, 1)$.

We reduce 3SAT to ρGAPqCSP.

3SAT has a PCP system in which the verifier makes a constant number of queries (q), and uses $c \log n$ random bits (for some c).

For input x and $r \in \{0,1\}^{c \log n}$ define

- $V_{x,r}$ as function that maps a proof π to the result (0/1) computed by the verifier when using proof π, instance x and random coins r.
- $V_{x,r}$ only depends on q bits of the proof

For any x the collection ϕ of the $V_{x,r}$'s over all r is polynomial size qCSP.

ϕ can be computed in polynomial time.

We know that $\mathrm{NP} \subseteq \mathrm{PCP}(\log n, 1)$.

We reduce 3SAT to ρGAPqCSP.

3SAT has a PCP system in which the verifier makes a constant number of queries (q), and uses $c \log n$ random bits (for some c).

For input x and $r \in \{0,1\}^{c \log n}$ define

- $V_{x,r}$ as function that maps a proof π to the result (0/1) computed by the verifier when using proof π, instance x and random coins r.
- $V_{x,r}$ only depends on q bits of the proof

For any x the collection ϕ of the $V_{x,r}$'s over all r is polynomial size qCSP.

ϕ can be computed in polynomial time.

We know that $\mathrm{NP} \subseteq \mathrm{PCP}(\log n, 1)$.

We reduce 3SAT to ρGAPqCSP.

3SAT has a PCP system in which the verifier makes a constant number of queries (q), and uses $c \log n$ random bits (for some c).

For input x and $r \in \{0,1\}^{c \log n}$ define

- $V_{x,r}$ as function that maps a proof π to the result (0/1) computed by the verifier when using proof π, instance x and random coins r.
- $V_{x,r}$ only depends on q bits of the proof

For any x the collection ϕ of the $V_{x,r}$'s over all r is polynomial size qCSP.

ϕ can be computed in polynomial time.

We know that $\mathrm{NP} \subseteq \mathrm{PCP}(\log n, 1)$.

We reduce 3SAT to ρGAPqCSP.

3SAT has a PCP system in which the verifier makes a constant number of queries (q), and uses $c \log n$ random bits (for some c).

For input x and $r \in \{0,1\}^{c \log n}$ define

- $V_{x,r}$ as function that maps a proof π to the result (0/1) computed by the verifier when using proof π, instance x and random coins r.
- $V_{x,r}$ only depends on q bits of the proof

For any x the collection ϕ of the $V_{x,r}$'s over all r is polynomial size qCSP.

ϕ can be computed in polynomial time.

$$x \in \text{3SAT} \implies \phi \text{ is satisfiable}$$

$$x \notin \text{3SAT} \implies \text{value}(\phi) \leq \frac{1}{2}$$

This means that ρGAPqCSP is NP-hard.

$$x \in \mathrm{3SAT} \Longrightarrow \phi \text{ is satisfiable}$$

$$x \notin \mathrm{3SAT} \Longrightarrow \mathrm{value}(\phi) \leq \frac{1}{2}$$

This means that ρGAPqCSP is NP-hard.

Suppose that ρGAPqCSP is NP-hard for some constants q, ρ ($\rho < 1$).

Suppose you get an input x, and have to decide whether $x \in L$.

We get a verifier as follows.

We use the reduction to map an input x into an instance ϕ of qCSP.

The proof is considered to be an assignment to the variables.

We can check a random constraint ϕ_i by making q queries. If $x \in L$ the verifier accepts with probability 1.

Otw. at most a ρ fraction of constraints are satisfied by the proof, and the verifier accepts with probability at most ρ.

Hence, $L \in \mathrm{PCP}_{1,\rho}(\log_2 m, q)$, where m is the number of constraints.

Suppose that ρGAPqCSP is NP-hard for some constants q, ρ ($\rho < 1$).

Suppose you get an input x, and have to decide whether $x \in L$.

We get a verifier as follows.

We use the reduction to map an input x into an instance ϕ of qCSP.

The proof is considered to be an assignment to the variables.

We can check a random constraint ϕ_i by making q queries. If $x \in L$ the verifier accepts with probability 1.

Otw. at most a ρ fraction of constraints are satisfied by the proof, and the verifier accepts with probability at most ρ.

Hence, $L \in \mathrm{PCP}_{1,\rho}(\log_2 m, q)$, where m is the number of constraints.

Suppose that ρGAPqCSP is NP-hard for some constants q, ρ ($\rho < 1$).

Suppose you get an input x, and have to decide whether $x \in L$.

We get a verifier as follows.

We use the reduction to map an input x into an instance ϕ of qCSP.

The proof is considered to be an assignment to the variables.

We can check a random constraint ϕ_i by making q queries. If $x \in L$ the verifier accepts with probability 1.

Otw. at most a ρ fraction of constraints are satisfied by the proof, and the verifier accepts with probability at most ρ.

Hence, $L \in \mathrm{PCP}_{1,\rho}(\log_2 m, q)$, where m is the number of constraints.

Suppose that ρGAPqCSP is NP-hard for some constants q, ρ ($\rho < 1$).

Suppose you get an input x, and have to decide whether $x \in L$.

We get a verifier as follows.

We use the reduction to map an input x into an instance ϕ of qCSP.

The proof is considered to be an assignment to the variables.

We can check a random constraint ϕ_i by making q queries. If $x \in L$ the verifier accepts with probability 1.

Otw. at most a ρ fraction of constraints are satisfied by the proof, and the verifier accepts with probability at most ρ.

Hence, $L \in \mathrm{PCP}_{1,\rho}(\log_2 m, q)$, where m is the number of constraints.

Suppose that ρGAPqCSP is NP-hard for some constants q, ρ ($\rho < 1$).

Suppose you get an input x, and have to decide whether $x \in L$.

We get a verifier as follows.

We use the reduction to map an input x into an instance ϕ of qCSP.

The proof is considered to be an assignment to the variables.

We can check a random constraint ϕ_i by making q queries. If $x \in L$ the verifier accepts with probability 1.

Otw. at most a ρ fraction of constraints are satisfied by the proof, and the verifier accepts with probability at most ρ.

Hence, $L \in \mathrm{PCP}_{1,\rho}(\log_2 m, q)$, where m is the number of constraints.

Suppose that ρGAPqCSP is NP-hard for some constants q, ρ ($\rho < 1$).

Suppose you get an input x, and have to decide whether $x \in L$.

We get a verifier as follows.

We use the reduction to map an input x into an instance ϕ of qCSP.

The proof is considered to be an assignment to the variables.

We can check a random constraint ϕ_i by making q queries. If $x \in L$ the verifier accepts with probability 1.

Otw. at most a ρ fraction of constraints are satisfied by the proof, and the verifier accepts with probability at most ρ.

Hence, $L \in \mathrm{PCP}_{1,\rho}(\log_2 m, q)$, where m is the number of constraints.

Suppose that ρGAPqCSP is NP-hard for some constants q, ρ ($\rho < 1$).

Suppose you get an input x, and have to decide whether $x \in L$.

We get a verifier as follows.

We use the reduction to map an input x into an instance ϕ of qCSP.

The proof is considered to be an assignment to the variables.

We can check a random constraint ϕ_i by making q queries. If $x \in L$ the verifier accepts with probability 1.

Otw. at most a ρ fraction of constraints are satisfied by the proof, and the verifier accepts with probability at most ρ.

Hence, $L \in \mathrm{PCP}_{1,\rho}(\log_2 m, q)$, where m is the number of constraints.

Theorem 10

For any positive constants $\epsilon, \delta > 0$*, it is the case that* $\mathrm{NP} \subseteq \mathrm{PCP}_{1-\epsilon,1/2+\delta}(\log n, 3)$*, and the verifier is restricted to use only the functions odd and even.*

It is NP-hard to approximate an ODD/EVEN constraint satisfaction problem by a factor better than $1/2 + \delta$, for any constant δ.

Theorem 11

For any positive constant $\delta > 0$, $\mathrm{NP} \subseteq \mathrm{PCP}_{1,7/8+\delta}(\mathcal{O}(\log n), 3)$ *and the verifier is restricted to use only functions that check the OR of three bits or their negations.*

It is NP-hard to approximate 3SAT better than $7/8 + \delta$.

The following GAP-problem is NP-hard for any $\epsilon > 0$.

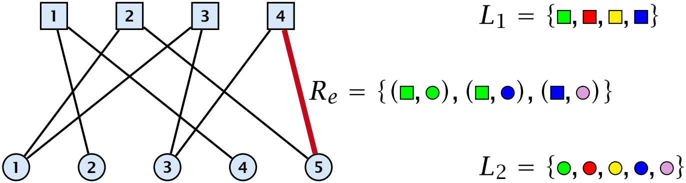
Given a graph $G = (V, E)$ composed of m independent sets of size 3 ($|V| = 3m$). Distinguish between

- the graph has a CLIQUE of size m
- the largest CLIQUE has size at most $(7/8 + \epsilon)m$

Label Cover

Input:

- bipartite graph $G = (V_1, V_2, E)$
- label sets L_1, L_2
- for every edge $(u, v) \in E$ a relation $R_{u,v} \subseteq L_1 \times L_2$ that describe assignments that make the edge *happy*.
- maximize number of happy edges



Label Cover

- an instance of label cover is (d_1, d_2)-regular if every vertex in L_1 has degree d_1 and every vertex in L_2 has degree d_2.
- if every vertex has the same degree d the instance is called d-regular

Minimization version:

- assign a set $L_x \subseteq L_1$ of labels to every node $x \in L_1$ and a set $L_y \subseteq L_2$ to every node $x \in L_2$
- make sure that for every edge (x, y) there is $\ell_x \in L_x$ and $\ell_y \in L_y$ s.t. $(\ell_x, \ell_y) \in R_{x,y}$
- minimize $\sum_{x \in L_1} |L_x| + \sum_{y \in L_2} |L_y|$ (total labels used)

MAX E3SAT via Label Cover

instance:

$\Phi(x) = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_4 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_4)$

corresponding graph:

label sets: $L_1 = \{T, F\}^3, L_2 = \{T, F\}$ (T=true, F=false)

relation: $R_{C,x_i} = \{((u_i, u_j, u_k), u_i)\}$, where the clause C is over variables x_i, x_j, x_k and assignment (u_i, u_j, u_k) satisfies C

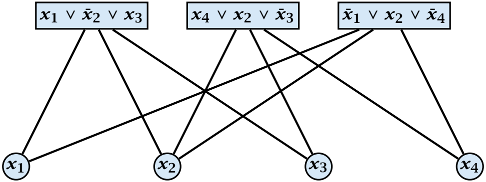
$$R = \{((F,F,F),F),((F,T,F),F),((F,F,T),T),((F,T,T),T),\\ ((T,T,T),T),((T,T,F),F),((T,F,F),F)\}$$

MAX E3SAT via Label Cover

instance:

$\Phi(x) = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_4 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_4)$

corresponding graph:



label sets: $L_1 = \{T, F\}^3, L_2 = \{T, F\}$ (T=true, F=false)

relation: $R_{C,x_i} = \{((u_i, u_j, u_k), u_i)\}$, where the clause C is over variables x_i, x_j, x_k and assignment (u_i, u_j, u_k) satisfies C

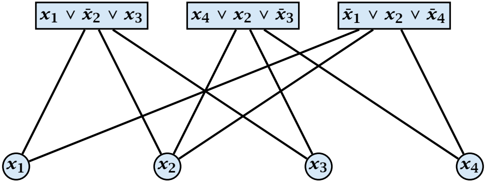
$$R = \{((F,F,F),F), ((F,T,F),F), ((F,F,T),T), ((F,T,T),T), ((T,T,T),T), ((T,T,F),F), ((T,F,F),F)\}$$

MAX E3SAT via Label Cover

instance:

$\Phi(x) = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_4 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_4)$

corresponding graph:



label sets: $L_1 = \{T, F\}^3, L_2 = \{T, F\}$ (T=true, F=false)

relation: $R_{C,x_i} = \{((u_i, u_j, u_k), u_i)\}$, where the clause C is over variables x_i, x_j, x_k and assignment (u_i, u_j, u_k) satisfies C

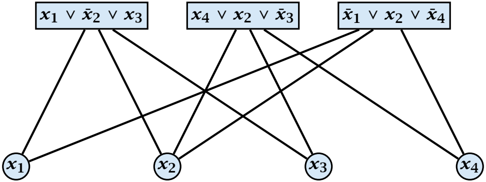
$$R = \{((F,F,F),F), ((F,T,F),F), ((F,F,T),T), ((F,T,T),T),$$
$$((T,T,T),T), ((T,T,F),F), ((T,F,F),F)\}$$

MAX E3SAT via Label Cover

instance:

$\Phi(x) = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_4 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_4)$

corresponding graph:



label sets: $L_1 = \{T, F\}^3, L_2 = \{T, F\}$ (T=true, F=false)

relation: $R_{C,x_i} = \{((u_i, u_j, u_k), u_i)\}$, where the clause C is over variables x_i, x_j, x_k and assignment (u_i, u_j, u_k) satisfies C

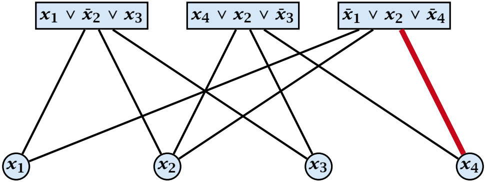
$$R = \{((F,F,F),F), ((F,T,F),F), ((F,F,T),T), ((F,T,T),T), ((T,T,T),T), ((T,T,F),F), ((T,F,F),F)\}$$

MAX E3SAT via Label Cover

instance:

$\Phi(x) = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_4 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_4)$

corresponding graph:



label sets: $L_1 = \{T, F\}^3, L_2 = \{T, F\}$ (T=true, F=false)

relation: $R_{C,x_i} = \{((u_i, u_j, u_k), u_i)\}$, where the clause C is over variables x_i, x_j, x_k and assignment (u_i, u_j, u_k) satisfies C

$$R = \{((F,F,F),F), ((F,T,F),F), ((F,F,T),T), ((F,T,T),T), ((T,T,T),T), ((T,T,F),F), ((T,F,F),F)\}$$

MAX E3SAT via Label Cover

Lemma 12

If we can satisfy k out of m clauses in ϕ we can make at least $3k + 2(m - k)$ edges happy.

Proof:

- for V_2 use the setting of the assignment that satisfies k clauses
- for satisfied clauses in V_1 use the corresponding assignment to the clause-variables (gives $3k$ happy edges)
- for unsatisfied clauses flip assignment of one of the variables; this makes one incident edge unhappy (gives $2(m - k)$ happy edges)

MAX E3SAT via Label Cover

Lemma 12

If we can satisfy k out of m clauses in ϕ we can make at least $3k + 2(m - k)$ edges happy.

Proof:

- for V_2 use the setting of the assignment that satisfies k clauses
- for satisfied clauses in V_1 use the corresponding assignment to the clause-variables (gives $3k$ happy edges)
- for unsatisfied clauses flip assignment of one of the variables; this makes one incident edge unhappy (gives $2(m - k)$ happy edges)

MAX E3SAT via Label Cover

Lemma 12

If we can satisfy k out of m clauses in ϕ we can make at least $3k + 2(m - k)$ edges happy.

Proof:

- for V_2 use the setting of the assignment that satisfies k clauses
- for satisfied clauses in V_1 use the corresponding assignment to the clause-variables (gives $3k$ happy edges)
- for unsatisfied clauses flip assignment of one of the variables; this makes one incident edge unhappy (gives $2(m - k)$ happy edges)

MAX E3SAT via Label Cover

Lemma 12

If we can satisfy k out of m clauses in ϕ we can make at least $3k + 2(m - k)$ edges happy.

Proof:

- for V_2 use the setting of the assignment that satisfies k clauses
- for satisfied clauses in V_1 use the corresponding assignment to the clause-variables (gives $3k$ happy edges)
- for unsatisfied clauses flip assignment of one of the variables; this makes one incident edge unhappy (gives $2(m - k)$ happy edges)

MAX E3SAT via Label Cover

Lemma 13

If we can satisfy at most k clauses in Φ we can make at most $3k + 2(m-k) = 2m + k$ edges happy.

Proof:

- the labeling of nodes in V_2 gives an assignment
- every unsatisfied clause in this assignment cannot be assigned a label that satisfies all 3 incident edges
- hence at most $3m - (m - k) = 2m + k$ edges are happy

MAX E3SAT via Label Cover

Lemma 13

If we can satisfy at most k clauses in Φ we can make at most $3k + 2(m - k) = 2m + k$ edges happy.

Proof:

- the labeling of nodes in V_2 gives an assignment
- every unsatisfied clause in this assignment cannot be assigned a label that satisfies all 3 incident edges
- hence at most $3m - (m - k) = 2m + k$ edges are happy

MAX E3SAT via Label Cover

Lemma 13

If we can satisfy at most k clauses in Φ we can make at most $3k + 2(m-k) = 2m + k$ edges happy.

Proof:

- the labeling of nodes in V_2 gives an assignment
- every unsatisfied clause in this assignment cannot be assigned a label that satisfies all 3 incident edges
- hence at most $3m - (m-k) = 2m + k$ edges are happy

MAX E3SAT via Label Cover

Lemma 13

If we can satisfy at most k clauses in Φ we can make at most $3k + 2(m - k) = 2m + k$ edges happy.

Proof:

- the labeling of nodes in V_2 gives an assignment
- every unsatisfied clause in this assignment cannot be assigned a label that satisfies all 3 incident edges
- hence at most $3m - (m - k) = 2m + k$ edges are happy

Hardness for Label Cover

We cannot distinguish between the following two cases

- all $3m$ edges can be made happy
- at most $2m + (7/8 + \epsilon)m \approx (\frac{23}{8} + \epsilon)m$ out of the $3m$ edges can be made happy

Hence, we cannot obtain an approximation constant $\alpha > \frac{23}{24}$.

Here α is a constant!!! Maybe a guarantee of the form $\frac{23}{8} + \frac{1}{m}$ is possible.

Hardness for Label Cover

We cannot distinguish between the following two cases

- all $3m$ edges can be made happy
- at most $2m + (7/8 + \epsilon)m \approx (\frac{23}{8} + \epsilon)m$ out of the $3m$ edges can be made happy

Hence, we cannot obtain an approximation constant $\alpha > \frac{23}{24}$.

Here α is a constant!!! Maybe a guarantee of the form $\frac{23}{8} + \frac{1}{m}$ is possible.

Hardness for Label Cover

We cannot distinguish between the following two cases

- all $3m$ edges can be made happy
- at most $2m + (7/8 + \epsilon)m \approx (\frac{23}{8} + \epsilon)m$ out of the $3m$ edges can be made happy

Hence, we cannot obtain an approximation constant $\alpha > \frac{23}{24}$.

Here α is a constant!!! Maybe a guarantee of the form $\frac{23}{8} + \frac{1}{m}$ is possible.

$(3,5)$-regular instances

Theorem 14
There is a constant ρ s.t. MAXE3SAT is hard to approximate with a factor of ρ even if restricted to instances where a variable appears in exactly 5 clauses.

Then our reduction has the following properties:

- the resulting Label Cover instance is $(3,5)$-regular
- it is hard to approximate for a constant $\alpha < 1$
- given a label ℓ_1 for x there is at most one label ℓ_2 for y that makes edge (x,y) happy (uniqueness property)

$(3, 5)$-regular instances

Theorem 14
There is a constant ρ s.t. MAXE3SAT is hard to approximate with a factor of ρ even if restricted to instances where a variable appears in exactly 5 clauses.

Then our reduction has the following properties:

- the resulting Label Cover instance is $(3, 5)$-regular
- it is hard to approximate for a constant $\alpha < 1$
- given a label ℓ_1 for x there is at most one label ℓ_2 for y that makes edge (x, y) happy (uniqueness property)

Regular instances

Theorem 15
If for a particular constant $\alpha < 1$ there is an α-approximation algorithm for Label Cover on 15-regular instances than P=NP.

Given a label ℓ_1 for $x \in V_1$ there is at most one label ℓ_2 for y that makes (x, y) happy. (uniqueness property)

Regular instances

proof...

Boosting

Given Label Cover instance I with $G = (V_1, V_2, E)$, label sets L_1 and L_2 we construct a new instance I':

- $V_1' = V_1^k = V_1 \times \cdots \times V_1$
- $V_2' = V_2^k = V_2 \times \cdots \times V_2$
- $L_1' = L_1^k = L_1 \times \cdots \times L_1$
- $L_2' = L_2^k = L_2 \times \cdots \times L_2$
- $E' = E^k = E \times \cdots \times E$

An edge $((x_1, \ldots, x_k), (y_1, \ldots, y_k))$ whose end-points are labelled by $(\ell_1^x, \ldots, \ell_k^x)$ and $(\ell_1^y, \ldots, \ell_k^y)$ is happy if $(\ell_i^x, \ell_i^y) \in R_{x_i, y_i}$ for all i.

Boosting

If I is regular than also I'.

If I has the uniqueness property than also I'.

Theorem 16
There is a constant $c > 0$ such if $\mathrm{OPT}(I) = |E|(1-\delta)$ then $\mathrm{OPT}(I') \le |E'|(1-\delta)^{\frac{ck}{\log L}}$, where $L = |L_1| + |L_2|$ denotes total number of labels in I.

proof is highly non-trivial

Boosting

If I is regular than also I'.

If I has the uniqueness property than also I'.

Theorem 16
There is a constant $c > 0$ such if $\mathrm{OPT}(I) = |E|(1-\delta)$ then $\mathrm{OPT}(I') \leq |E'|(1-\delta)^{\frac{ck}{\log L}}$, where $L = |L_1| + |L_2|$ denotes total number of labels in I.

proof is highly non-trivial

Boosting

If I is regular than also I'.

If I has the uniqueness property than also I'.

Theorem 16
There is a constant $c > 0$ such if $\mathrm{OPT}(I) = |E|(1-\delta)$ then $\mathrm{OPT}(I') \le |E'|(1-\delta)^{\frac{ck}{\log L}}$, where $L = |L_1| + |L_2|$ denotes total number of labels in I.

proof is highly non-trivial

Boosting

If I is regular than also I'.

If I has the uniqueness property than also I'.

Theorem 16
There is a constant $c > 0$ such if $\mathrm{OPT}(I) = |E|(1-\delta)$ then $\mathrm{OPT}(I') \le |E'|(1-\delta)^{\frac{ck}{\log L}}$, where $L = |L_1| + |L_2|$ denotes total number of labels in I.

proof is highly non-trivial

Boosting

If I is regular than also I'.

If I has the uniqueness property than also I'.

Theorem 16
There is a constant $c > 0$ *such if* $\mathrm{OPT}(I) = |E|(1-\delta)$ *then* $\mathrm{OPT}(I') \leq |E'|(1-\delta)^{\frac{ck}{\log L}}$, *where* $L = |L_1| + |L_2|$ *denotes total number of labels in* I.

proof is highly non-trivial

Theorem 17

There are constants $c > 0$, $\delta < 1$ *s.t. for any* k *we cannot distinguish regular instances for Label Cover in which either*

- $\mathrm{OPT}(I) = |E|$, *or*
- $\mathrm{OPT}(I) = |E|(1-\delta)^{\frac{ck}{\log 10}}$

unless each problem in NP has an algorithm running in time $\mathcal{O}(n^{\mathcal{O}(k)})$.

Corollary 18

There is no α*-approximation for Label Cover for any constant* α.

Set Cover

Theorem 19
There exist regular Label Cover instances s.t. we cannot distinguish whether

- *all edges are satisfiable, or*
- *at most a $1/\log^2(|L_2||E|)$-fraction is satisfiable*

unless NP-problems have algorithms with running time $\mathcal{O}(n^{\mathcal{O}(\log\log n)})$.

choose $k = \frac{2\log 10}{c}\log_{1/(1-\delta)}(\log(|L_2||E|)) = \mathcal{O}(\log\log n)$.

Set Cover

Partition System (s, t, h)

- universe U of size s
- t pairs of sets $(A_1, \bar{A}_1), \dots, (A_t, \bar{A}_t)$;
  $A_i \subseteq U, \bar{A}_i = U \setminus A_i$
- choosing from any h pairs only one of A_i, $\bar{A}_i$ we do not cover the whole set U

For any h, t with $h \le t$ there exist systems with $s = |U| \le 2^{2h+2}t^2$.

Set Cover

Given a Label Cover instance we construct a Set Cover instance;

The universe is $E \times U$, where U is the universe of some partition system; $(t = |L_2|, h = (\log |E||L_2|))$

for all $v \in V_2, j \in L_2$

$$S_{v,j} = \{((u,v),a) \mid (u,v) \in E, a \in A_j\}$$

for all $u \in V_1, i \in L_1$

$$S_{u,i} = \{((u,v),a) \mid (u,v) \in E, a \in \bar{A}_j, \text{ where } (i,j) \in R_{(u,v)}\}$$

note that $S_{u,i}$ is well-defined because of the uniqueness property

Set Cover

Given a Label Cover instance we construct a Set Cover instance;

The universe is $E \times U$, where U is the universe of some partition system; ($t = |L_2|$, $h = (\log |E||L_2|)$)

for all $v \in V_2, j \in L_2$

$$S_{v,j} = \{((u,v),a) \mid (u,v) \in E, a \in A_j\}$$

for all $u \in V_1, i \in L_1$

$$S_{u,i} = \{((u,v),a) \mid (u,v) \in E, a \in \bar{A}_j, \text{ where } (i,j) \in R_{(u,v)}\}$$

note that $S_{u,i}$ is well-defined because of the uniqueness property

Set Cover

Given a Label Cover instance we construct a Set Cover instance;

The universe is $E \times U$, where U is the universe of some partition system; ($t = |L_2|$, $h = (\log |E||L_2|)$)

for all $v \in V_2, j \in L_2$

$$S_{v,j} = \{((u,v),a) \mid (u,v) \in E, a \in A_j\}$$

for all $u \in V_1, i \in L_1$

$$S_{u,i} = \{((u,v),a) \mid (u,v) \in E, a \in \bar{A}_j, \text{ where } (i,j) \in R_{(u,v)}\}$$

note that $S_{u,i}$ is well-defined because of the uniqueness property

Set Cover

Given a Label Cover instance we construct a Set Cover instance;

The universe is $E \times U$, where U is the universe of some partition system; ($t = |L_2|$, $h = (\log |E||L_2|)$)

for all $v \in V_2, j \in L_2$

$$S_{v,j} = \{((u,v),a) \mid (u,v) \in E, a \in A_j\}$$

for all $u \in V_1, i \in L_1$

$$S_{u,i} = \{((u,v),a) \mid (u,v) \in E, a \in \bar{A}_j, \text{ where } (i,j) \in R_{(u,v)}\}$$

note that $S_{u,i}$ is well-defined because of the uniqueness property

Set Cover

Given a Label Cover instance we construct a Set Cover instance;

The universe is $E \times U$, where U is the universe of some partition system; ($t = |L_2|$, $h = (\log |E||L_2|)$)

for all $v \in V_2, j \in L_2$

$$S_{v,j} = \{((u,v),a) \mid (u,v) \in E, a \in A_j\}$$

for all $u \in V_1, i \in L_1$

$$S_{u,i} = \{((u,v),a) \mid (u,v) \in E, a \in \bar{A}_j, \text{ where } (i,j) \in R_{(u,v)}\}$$

note that $S_{u,i}$ is well-defined because of the uniqueness property

Set Cover

Given a Label Cover instance we construct a Set Cover instance;

The universe is $E \times U$, where U is the universe of some partition system; ($t = |L_2|$, $h = (\log |E||L_2|)$)

for all $v \in V_2, j \in L_2$

$$S_{v,j} = \{((u,v),a) \mid (u,v) \in E, a \in A_j\}$$

for all $u \in V_1, i \in L_1$

$$S_{u,i} = \{((u,v),a) \mid (u,v) \in E, a \in \bar{A}_j, \text{where } (i,j) \in R_{(u,v)}\}$$

note that $S_{u,i}$ is well-defined because of the uniqueness property

Suppose that we can make all edges happy.

Choose sets $S_{u,i}$'s and $S_{v,j}$'s, where i is the label we assigned to u, and j the label for v. ($|V_1|+|V_2|$ sets)

For an edge (u,v), $S_{v,j}$ contains $\{(u,v)\} \times A_j$. For a happy edge $S_{u,i}$ contains $\{(u,v)\} \times \bar{A}_j$.

Since all edges are happy we have covered the whole universe.

Suppose that we can make all edges happy.

Choose sets $S_{u,i}$'s and $S_{v,j}$'s, where i is the label we assigned to u, and j the label for v. ($|V_1|+|V_2|$ sets)

For an edge (u,v), $S_{v,j}$ contains $\{(u,v)\} \times A_j$. For a happy edge $S_{u,i}$ contains $\{(u,v)\} \times \bar{A}_j$.

Since all edges are happy we have covered the whole universe.

Suppose that we can make all edges happy.

Choose sets $S_{u,i}$'s and $S_{v,j}$'s, where i is the label we assigned to u, and j the label for v. ($|V_1|+|V_2|$ sets)

For an edge (u,v), $S_{v,j}$ contains $\{(u,v)\} \times A_j$. For a happy edge $S_{u,i}$ contains $\{(u,v)\} \times \bar{A}_j$.

Since all edges are happy we have covered the whole universe.

Suppose that we can make all edges happy.

Choose sets $S_{u,i}$'s and $S_{v,j}$'s, where i is the label we assigned to u, and j the label for v. ($|V_1|+|V_2|$ sets)

For an edge (u,v), $S_{v,j}$ contains $\{(u,v)\} \times A_j$. For a happy edge $S_{u,i}$ contains $\{(u,v)\} \times \bar{A}_j$.

Since all edges are happy we have covered the whole universe.

Suppose that we can make all edges happy.

Choose sets $S_{u,i}$'s and $S_{v,j}$'s, where i is the label we assigned to u, and j the label for v. ($|V_1|$+$|V_2|$ sets)

For an edge (u,v), $S_{v,j}$ contains $\{(u,v)\} \times A_j$. For a happy edge $S_{u,i}$ contains $\{(u,v)\} \times \bar{A}_j$.

Since all edges are happy we have covered the whole universe.

Lemma 20

Given a solution to the set cover instance using at most $\frac{h}{8}(|V_1| + |V_2|)$ sets we can find a solution to the Label Cover instance satisfying at least $\frac{2}{h^2}|E|$ edges.

- n_u: number of $S_{u,i}$'s in cover
- n_v: number of $S_{v,j}$'s in cover
- at most 1/4 of the vertices can have $n_u, n_v \geq h/2$; mark these vertices
- at least half of the edges have both end-points unmarked, as the graph is regular
- for such an edge (u, v) we must have chosen $S_{u,i}$ and a corresponding $S_{v,j}$, s.t. $(i, j) \in R_{u,v}$ (making (u, v) happy)
- we choose a random label for u from the (at most $h/2$) chosen $S_{u,i}$-sets and a random label for v from the (at most $h/2$) $S_{v,j}$-sets
- (u, v) gets happy with probability at least $4/h^2$
- hence we make an $2/h^2$-fraction of edges happy

- n_u: number of $S_{u,i}$'s in cover
- n_v: number of $S_{v,j}$'s in cover
- at most 1/4 of the vertices can have $n_u, n_v \geq h/2$; mark these vertices
- at least half of the edges have both end-points unmarked, as the graph is regular
- for such an edge (u,v) we must have chosen $S_{u,i}$ and a corresponding $S_{v,j}$, s.t. $(i,j) \in R_{u,v}$ (making (u,v) happy)
- we choose a random label for u from the (at most $h/2$) chosen $S_{u,i}$-sets and a random label for v from the (at most $h/2$) $S_{v,j}$-sets
- (u,v) gets happy with probability at least $4/h^2$
- hence we make an $2/h^2$-fraction of edges happy

- n_u: number of $S_{u,i}$'s in cover
- n_v: number of $S_{v,j}$'s in cover
- at most 1/4 of the vertices can have $n_u, n_v \geq h/2$; mark these vertices
- at least half of the edges have both end-points unmarked, as the graph is regular
- for such an edge (u, v) we must have chosen $S_{u,i}$ and a corresponding $S_{v,j}$, s.t. $(i, j) \in R_{u,v}$ (making (u, v) happy)
- we choose a random label for u from the (at most $h/2$) chosen $S_{u,i}$-sets and a random label for v from the (at most $h/2$) $S_{v,j}$-sets
- (u, v) gets happy with probability at least $4/h^2$
- hence we make an $2/h^2$-fraction of edges happy

- n_u: number of $S_{u,i}$'s in cover
- n_v: number of $S_{v,j}$'s in cover
- at most 1/4 of the vertices can have $n_u, n_v \geq h/2$; mark these vertices
- at least half of the edges have both end-points unmarked, as the graph is regular
- for such an edge (u, v) we must have chosen $S_{u,i}$ and a corresponding $S_{v,j}$, s.t. $(i, j) \in R_{u,v}$ (making (u, v) happy)
- we choose a random label for u from the (at most $h/2$) chosen $S_{u,i}$-sets and a random label for v from the (at most $h/2$) $S_{v,j}$-sets
- (u, v) gets happy with probability at least $4/h^2$
- hence we make an $2/h^2$-fraction of edges happy

- n_u: number of $S_{u,i}$'s in cover
- n_v: number of $S_{v,j}$'s in cover
- at most 1/4 of the vertices can have $n_u, n_v \geq h/2$; mark these vertices
- at least half of the edges have both end-points unmarked, as the graph is regular
- for such an edge (u,v) we must have chosen $S_{u,i}$ and a corresponding $S_{v,j}$, s.t. $(i,j) \in R_{u,v}$ (making (u,v) happy)
- we choose a random label for u from the (at most $h/2$) chosen $S_{u,i}$-sets and a random label for v from the (at most $h/2$) $S_{v,j}$-sets
- (u,v) gets happy with probability at least $4/h^2$
- hence we make an $2/h^2$-fraction of edges happy

- n_u: number of $S_{u,i}$'s in cover
- n_v: number of $S_{v,j}$'s in cover
- at most 1/4 of the vertices can have $n_u, n_v \geq h/2$; mark these vertices
- at least half of the edges have both end-points unmarked, as the graph is regular
- for such an edge (u,v) we must have chosen $S_{u,i}$ and a corresponding $S_{v,j}$, s.t. $(i,j) \in R_{u,v}$ (making (u,v) happy)
- we choose a random label for u from the (at most $h/2$) chosen $S_{u,i}$-sets and a random label for v from the (at most $h/2$) $S_{v,j}$-sets
- (u,v) gets happy with probability at least $4/h^2$
- hence we make an $2/h^2$-fraction of edges happy

- n_u: number of $S_{u,i}$'s in cover
- n_v: number of $S_{v,j}$'s in cover
- at most 1/4 of the vertices can have $n_u, n_v \geq h/2$; mark these vertices
- at least half of the edges have both end-points unmarked, as the graph is regular
- for such an edge (u, v) we must have chosen $S_{u,i}$ and a corresponding $S_{v,j}$, s.t. $(i, j) \in R_{u,v}$ (making (u, v) happy)
- we choose a random label for u from the (at most $h/2$) chosen $S_{u,i}$-sets and a random label for v from the (at most $h/2$) $S_{v,j}$-sets
- (u, v) gets happy with probability at least $4/h^2$
- hence we make an $2/h^2$-fraction of edges happy

- n_u: number of $S_{u,i}$'s in cover
- n_v: number of $S_{v,j}$'s in cover
- at most 1/4 of the vertices can have $n_u, n_v \geq h/2$; mark these vertices
- at least half of the edges have both end-points unmarked, as the graph is regular
- for such an edge (u, v) we must have chosen $S_{u,i}$ and a corresponding $S_{v,j}$, s.t. $(i, j) \in R_{u,v}$ (making (u, v) happy)
- we choose a random label for u from the (at most $h/2$) chosen $S_{u,i}$-sets and a random label for v from the (at most $h/2$) $S_{v,j}$-sets
- (u, v) gets happy with probability at least $4/h^2$
- hence we make an $2/h^2$-fraction of edges happy

Set Cover

Theorem 21

There is no $\frac{1}{32}\log N$-approximation for the unweighted Set Cover problem unless problems in NP can be solved in time $\mathcal{O}(n^{\mathcal{O}(\log\log n)})$.

Given label cover instance (V_1, V_2, E), label sets L_1 and L_2;

Set $h = \log(|E||L_2|)$ and $t = |L_2|$; Size of partition system is

$$s = |U| = 2^{2h+2}t^2 = 4(|E||L_2|)^2|L_2|^2 = 4|E|^2|L_2|^4$$

The size of the ground set is then

$$N = |E||U| = 4|E|^3|L_2|^4 \le (|E||L_2|)^4$$

for sufficiently large $|E|$. Then $h \ge \frac{1}{4}\log N$.

If we get an instance where all edges are satisfiable there exists a cover of size only $|V_1| + |V_2|$.

If we find a cover of size at most $\frac{h}{8}(|V_1| + |V_2|)$ we can use this to satisfy at least a fraction of $2/h^2 \ge 1/\log^2(|E||L_2|)$ of the edges. this is not possible...

Given label cover instance (V_1, V_2, E), label sets L_1 and L_2;

Set $h = \log(|E||L_2|)$ and $t = |L_2|$; Size of partition system is

$$s = |U| = 2^{2h+2}t^2 = 4(|E||L_2|)^2|L_2|^2 = 4|E|^2|L_2|^4$$

The size of the ground set is then

$$N = |E||U| = 4|E|^3|L_2|^4 \le (|E||L_2|)^4$$

for sufficiently large $|E|$. Then $h \ge \frac{1}{4}\log N$.

If we get an instance where all edges are satisfiable there exists a cover of size only $|V_1| + |V_2|$.

If we find a cover of size at most $\frac{h}{8}(|V_1| + |V_2|)$ we can use this to satisfy at least a fraction of $2/h^2 \ge 1/\log^2(|E||L_2|)$ of the edges. this is not possible...

Given label cover instance (V_1, V_2, E), label sets L_1 and L_2;

Set $h = \log(|E||L_2|)$ and $t = |L_2|$; Size of partition system is

$$s = |U| = 2^{2h+2}t^2 = 4(|E||L_2|)^2|L_2|^2 = 4|E|^2|L_2|^4$$

The size of the ground set is then

$$N = |E||U| = 4|E|^3|L_2|^4 \le (|E||L_2|)^4$$

for sufficiently large $|E|$. Then $h \ge \frac{1}{4}\log N$.

If we get an instance where all edges are satisfiable there exists a cover of size only $|V_1| + |V_2|$.

If we find a cover of size at most $\frac{h}{8}(|V_1| + |V_2|)$ we can use this to satisfy at least a fraction of $2/h^2 \ge 1/\log^2(|E||L_2|)$ of the edges. this is not possible...

Given label cover instance (V_1, V_2, E), label sets L_1 and L_2;

Set $h = \log(|E||L_2|)$ and $t = |L_2|$; Size of partition system is

$$s = |U| = 2^{2h+2}t^2 = 4(|E||L_2|)^2|L_2|^2 = 4|E|^2|L_2|^4$$

The size of the ground set is then

$$N = |E||U| = 4|E|^3|L_2|^4 \le (|E||L_2|)^4$$

for sufficiently large $|E|$. Then $h \ge \frac{1}{4}\log N$.

If we get an instance where all edges are satisfiable there exists a cover of size only $|V_1| + |V_2|$.

If we find a cover of size at most $\frac{h}{8}(|V_1| + |V_2|)$ we can use this to satisfy at least a fraction of $2/h^2 \ge 1/\log^2(|E||L_2|)$ of the edges. this is not possible...

Given label cover instance (V_1, V_2, E), label sets L_1 and L_2;

Set $h = \log(|E||L_2|)$ and $t = |L_2|$; Size of partition system is

$$s = |U| = 2^{2h+2}t^2 = 4(|E||L_2|)^2|L_2|^2 = 4|E|^2|L_2|^4$$

The size of the ground set is then

$$N = |E||U| = 4|E|^3|L_2|^4 \le (|E||L_2|)^4$$

for sufficiently large $|E|$. Then $h \ge \frac{1}{4}\log N$.

If we get an instance where all edges are satisfiable there exists a cover of size only $|V_1| + |V_2|$.

If we find a cover of size at most $\frac{h}{8}(|V_1| + |V_2|)$ we can use this to satisfy at least a fraction of $2/h^2 \ge 1/\log^2(|E||L_2|)$ of the edges. this is not possible...

Partition Systems

Lemma 22
Given h and t there is a partition system of size $s = 2^h h \ln(4t) \leq 2^{2h+2}t^2$.

We pick t sets at random from the possible $2^{|U|}$ subsets of U.

Fix a choice of h of these sets, and a choice of h bits (whether we choose A_i or $\bar{A}_i$). There are $2^h \cdot \binom{t}{h}$ such choices.

Partition Systems

Lemma 22
Given h and t there is a partition system of size $s = 2^h h \ln(4t) \leq 2^{2h+2}t^2$.

We pick t sets at random from the possible $2^{|U|}$ subsets of U.

Fix a choice of h of these sets, and a choice of h bits (whether we choose A_i or $\bar{A}_i$). There are $2^h \cdot \binom{t}{h}$ such choices.

Partition Systems

Lemma 22
Given h and t there is a partition system of size $s = 2^h h \ln(4t) \leq 2^{2h+2} t^2$.

We pick t sets at random from the possible $2^{|U|}$ subsets of U.

Fix a choice of h of these sets, and a choice of h bits (whether we choose A_i or $\bar{A}_i$). There are $2^h \cdot \binom{t}{h}$ such choices.

What is the probability that a given choice covers U?

The probability that an element $u \in A_i$ is $1/2$ (same for $\bar{A}_i$).

The probability that u is covered is $1 - \frac{1}{2^h}$.

The probability that all u are covered is $(1 - \frac{1}{2^h})^s$

The probability that there exists a choice such that all u are covered is at most

$$\binom{t}{h} 2^h \left(1 - \frac{1}{2^h}\right)^s \leq (2t)^h e^{-s/2^h} = (2t)^h \cdot e^{-h \ln(4t)} \leq \frac{1}{2^h}$$

What is the probability that a given choice covers U?

The probability that an element $u \in A_i$ is $1/2$ (same for $\bar{A}_i$).

The probability that u is covered is $1 - \frac{1}{2^h}$.

The probability that all u are covered is $(1 - \frac{1}{2^h})^s$

The probability that there exists a choice such that all u are covered is at most

$$\binom{t}{h} 2^h \left(1 - \frac{1}{2^h}\right)^s \leq (2t)^h e^{-s/2^h} = (2t)^h \cdot e^{-h \ln(4t)} \leq \frac{1}{2^h}$$

What is the probability that a given choice covers U?

The probability that an element $u \in A_i$ is $1/2$ (same for $\bar{A}_i$).

The probability that u is covered is $1 - \frac{1}{2^h}$.

The probability that all u are covered is $(1 - \frac{1}{2^h})^s$

The probability that there exists a choice such that all u are covered is at most

$$\binom{t}{h} 2^h \left(1 - \frac{1}{2^h}\right)^s \leq (2t)^h e^{-s/2^h} = (2t)^h \cdot e^{-h \ln(4t)} \leq \frac{1}{2^h}$$

What is the probability that a given choice covers U?

The probability that an element $u \in A_i$ is $1/2$ (same for $\bar{A}_i$).

The probability that u is covered is $1 - \frac{1}{2^h}$.

The probability that all u are covered is $(1 - \frac{1}{2^h})^s$

The probability that there exists a choice such that all u are covered is at most

$$\binom{t}{h} 2^h \left(1 - \frac{1}{2^h}\right)^s \le (2t)^h e^{-s/2^h} = (2t)^h \cdot e^{-h\ln(4t)} \le \frac{1}{2^h}$$

What is the probability that a given choice covers U?

The probability that an element $u \in A_i$ is $1/2$ (same for $\bar{A}_i$).

The probability that u is covered is $1 - \frac{1}{2^h}$.

The probability that all u are covered is $(1 - \frac{1}{2^h})^s$

The probability that there exists a choice such that all u are covered is at most

$$\binom{t}{h} 2^h \left(1 - \frac{1}{2^h}\right)^s \leq (2t)^h e^{-s/2^h} = (2t)^h \cdot e^{-h \ln(4t)} \leq \frac{1}{2^h}$$

What is the probability that a given choice covers U?

The probability that an element $u \in A_i$ is $1/2$ (same for $\bar{A}_i$).

The probability that u is covered is $1 - \frac{1}{2^h}$.

The probability that all u are covered is $(1 - \frac{1}{2^h})^s$

The probability that there exists a choice such that all u are covered is at most

$$\binom{t}{h} 2^h \left(1 - \frac{1}{2^h}\right)^s \leq (2t)^h e^{-s/2^h} = (2t)^h \cdot e^{-h \ln(4t)} \leq \frac{1}{2^h}$$