Traveling Salesman

Given a set of cities ($\{1,\ldots,n\}$) and a symmetric matrix $C = (c_{ij})$, $c_{ij} \geq 0$ that specifies for every pair $(i,j) \in [n] \times [n]$ the cost for travelling from city i to city j. Find a permutation π of the cities such that the round-trip cost

$$c_{\pi(1)\pi(n)} + \sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)}$$

is minimized.

Traveling Salesman

Theorem 3

There does not exist an $O(2^n)$-approximation algorithm for TSP.

Hamiltonian Cycle:
For a given undirected graph $G = (V, E)$ decide whether there exists a simple cycle that contains all nodes in G.

- Given an instance to HAMPATH we create an instance for TSP.
- If $(i,j) \notin E$ then set c_{ij} to $n2^n$ otw. set c_{ij} to 1. This instance has polynomial size.
- There exists a Hamiltonian Path iff there exists a tour with cost n. Otw. any tour has cost strictly larger than $n2^n$.
- An $O(2^n)$-approximation algorithm could decide btw. these cases. Hence, cannot exist unless $P = NP$.

Traveling Salesman

Theorem 3

There does not exist an $O(2^n)$-approximation algorithm for TSP.

Hamiltonian Cycle:
For a given undirected graph $G = (V, E)$ decide whether there exists a simple cycle that contains all nodes in G.

Traveling Salesman

Theorem 3

There does not exist an $O(2^n)$-approximation algorithm for TSP.

Hamiltonian Cycle:
For a given undirected graph $G = (V, E)$ decide whether there exists a simple cycle that contains all nodes in G.

- Given an instance to HAMPATH we create an instance for TSP.
- If $(i, j) \notin E$ then set c_{ij} to $n2^n$ otw. set c_{ij} to 1. This instance has polynomial size.
- There exists a Hamiltonian Path iff there exists a tour with cost n. Otw. any tour has cost strictly larger than $n2^n$.
- An $\mathcal{O}(2^n)$-approximation algorithm could decide btw. these cases. Hence, cannot exist unless $P = NP$.

Traveling Salesman

Theorem 3
There does not exist an $O(2^n)$-approximation algorithm for TSP.

Hamiltonian Cycle:
For a given undirected graph $G = (V, E)$ decide whether there exists a simple cycle that contains all nodes in G.

- Given an instance to HAMPATH we create an instance for TSP.
- If $(i, j) \notin E$ then set c_{ij} to $n2^n$ otw. set c_{ij} to 1. This instance has polynomial size.
- There exists a Hamiltonian Path iff there exists a tour with cost n. Otw. any tour has cost strictly larger than $n2^n$.
- An $\mathcal{O}(2^n)$-approximation algorithm could decide btw. these cases. Hence, cannot exist unless $P = NP$.

Traveling Salesman

Theorem 3

There does not exist an $O(2^n)$-approximation algorithm for TSP.

Hamiltonian Cycle:
For a given undirected graph $G = (V, E)$ decide whether there exists a simple cycle that contains all nodes in G.

- Given an instance to HAMPATH we create an instance for TSP.
- If $(i, j) \notin E$ then set c_{ij} to $n2^n$ otw. set c_{ij} to 1. This instance has polynomial size.
- There exists a Hamiltonian Path iff there exists a tour with cost n. Otw. any tour has cost strictly larger than $n2^n$.
- An $\mathcal{O}(2^n)$-approximation algorithm could decide btw. these cases. Hence, cannot exist unless $P = NP$.

Traveling Salesman

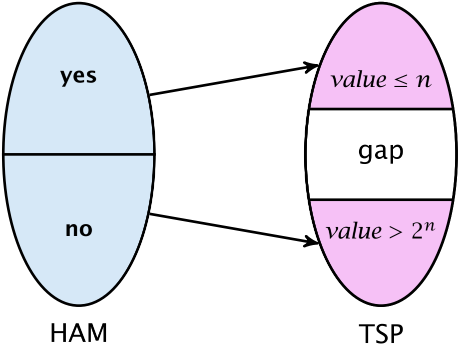
Theorem 3

There does not exist an $O(2^n)$-approximation algorithm for TSP.

Hamiltonian Cycle:
For a given undirected graph $G = (V, E)$ decide whether there exists a simple cycle that contains all nodes in G.

- Given an instance to HAMPATH we create an instance for TSP.
- If $(i, j) \notin E$ then set c_{ij} to $n2^n$ otw. set c_{ij} to 1. This instance has polynomial size.
- There exists a Hamiltonian Path iff there exists a tour with cost n. Otw. any tour has cost strictly larger than $n2^n$.
- An $\mathcal{O}(2^n)$-approximation algorithm could decide btw. these cases. Hence, cannot exist unless $P = NP$.

Gap Introducing Reduction



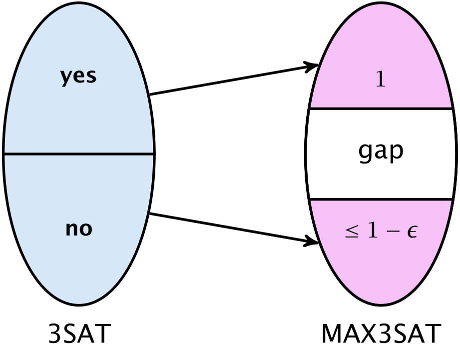
Reduction from Hamiltonian cycle to TSP

- instance that has Hamiltonian cycle is mapped to TSP instance with small cost
- otherwise it is mapped to instance with large cost
- $\Longrightarrow$ there is no $2^n/n$-approximation for TSP

PCP theorem: Approximation View

Theorem 4 (PCP Theorem A)

There exists $\epsilon > 0$ for which there is gap introducing reduction between 3SAT and MAX3SAT.



PCP theorem: Proof System View

Definition 5 (NP)

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

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

[$x \notin L$] **soundness**
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?**).

PCP theorem: Proof System View

Definition 5 (NP)

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

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

$[x \notin L]$ soundness
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 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.

Probabilistic Checkable Proofs

Definition 6 (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, s.t.

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

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

The verifier uses at most $\mathcal{O}(r(n))$ random bits and makes at most $\mathcal{O}(q(n))$ oracle queries.

Probabilistic Checkable Proofs

$c(n)$ 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.

Probabilistic Checkable Proofs

- $\mathrm{P} = \mathrm{PCP}(0, 0)$
  verifier without randomness and proof access is deterministic algorithm
- $\mathrm{PCP}(\log n, 0) \subseteq \mathrm{P}$
  we can simulate $O(\log n)$ random bits in deterministic polynomial time
- $\mathrm{PCP}(0, \log n) \subseteq \mathrm{P}$
  we can simulate short proofs in polynomial time
- $\mathrm{PCP}(\mathrm{poly}(n), 0) = \mathrm{coRP} \stackrel{?!}{=} \mathrm{P}$
  by definition; coRP is randomized polytime with one sided error (positive probability of accepting NO-instance)

Note that the first three statements also hold with equality

Probabilistic Checkable Proofs

- $\mathrm{P} = \mathrm{PCP}(0,0)$
  verifier without randomness and proof access is deterministic algorithm

Probabilistic Checkable Proofs

- $\mathrm{P} = \mathrm{PCP}(0,0)$
  verifier without randomness and proof access is deterministic algorithm
- $\mathrm{PCP}(\log n, 0) \subseteq \mathrm{P}$
  we can simulate $O(\log n)$ random bits in deterministic, polynomial time
- $\mathrm{PCP}(0, \log n) \subseteq \mathrm{P}$
  we can simulate short proofs in polynomial time
- $\mathrm{PCP}(\mathrm{poly}(n), 0) = \mathrm{coRP} \stackrel{?!}{=} \mathrm{P}$
  by definition; coRP is randomized polytime with one sided error (positive probability of accepting NO-instance)

Note that the first three statements also hold with equality

Probabilistic Checkable Proofs

- $P = PCP(0, 0)$
  verifier without randomness and proof access is deterministic algorithm
- $PCP(\log n, 0) \subseteq P$
  we can simulate $O(\log n)$ random bits in deterministic, polynomial time
- $PCP(0, \log n) \subseteq P$
  we can simulate short proofs in polynomial time
- $PCP(\text{poly}(n), 0) = coRP \stackrel{?!}{=} P$
  by definition; coRP is randomized polytime with one sided error (positive probability of accepting NO-instance)

Note that the first three statements also hold with equality

Probabilistic Checkable Proofs

- $P = PCP(0, 0)$
  verifier without randomness and proof access is deterministic algorithm
- $PCP(\log n, 0) \subseteq P$
  we can simulate $O(\log n)$ random bits in deterministic, polynomial time
- $PCP(0, \log n) \subseteq P$
  we can simulate short proofs in polynomial time
- $PCP(\text{poly}(n), 0) = coRP \stackrel{?!}{=} P$
  by definition; coRP is randomized polytime with one sided error (positive probability of accepting NO-instance)

Note that the first three statements also hold with equality

Probabilistic Checkable Proofs

- $\mathrm{P} = \mathrm{PCP}(0, 0)$
  verifier without randomness and proof access is deterministic algorithm
- $\mathrm{PCP}(\log n, 0) \subseteq \mathrm{P}$
  we can simulate $O(\log n)$ random bits in deterministic, polynomial time
- $\mathrm{PCP}(0, \log n) \subseteq \mathrm{P}$
  we can simulate short proofs in polynomial time
- $\mathrm{PCP}(\mathrm{poly}(n), 0) = \mathrm{coRP} \stackrel{?!}{=} \mathrm{P}$
  by definition; coRP is randomized polytime with one sided error (positive probability of accepting NO-instance)

Note that the first three statements also hold with equality

Probabilistic Checkable Proofs

- $P = PCP(0, 0)$
  verifier without randomness and proof access is deterministic algorithm
- $PCP(\log n, 0) \subseteq P$
  we can simulate $O(\log n)$ random bits in deterministic, polynomial time
- $PCP(0, \log n) \subseteq P$
  we can simulate short proofs in polynomial time
- $PCP(\text{poly}(n), 0) = \text{coRP} \stackrel{?!}{=} P$
  by definition; coRP is randomized polytime with one sided error (positive probability of accepting NO-instance)

Note that the first three statements also hold with equality

Probabilistic Checkable Proofs

- $\mathrm{P} = \mathrm{PCP}(0,0)$
  verifier without randomness and proof access is deterministic algorithm
- $\mathrm{PCP}(\log n, 0) \subseteq \mathrm{P}$
  we can simulate $O(\log n)$ random bits in deterministic, polynomial time
- $\mathrm{PCP}(0, \log n) \subseteq \mathrm{P}$
  we can simulate short proofs in polynomial time
- $\mathrm{PCP}(\mathrm{poly}(n), 0) = \mathrm{coRP} \stackrel{?!}{=} \mathrm{P}$
  by definition; coRP is randomized polytime with one sided error (positive probability of accepting NO-instance)

Note that the first three statements also hold with equality

Probabilistic Checkable Proofs

- $\mathrm{P} = \mathrm{PCP}(0,0)$
  verifier without randomness and proof access is deterministic algorithm
- $\mathrm{PCP}(\log n, 0) \subseteq \mathrm{P}$
  we can simulate $O(\log n)$ random bits in deterministic, polynomial time
- $\mathrm{PCP}(0, \log n) \subseteq \mathrm{P}$
  we can simulate short proofs in polynomial time
- $\mathrm{PCP}(\mathrm{poly}(n), 0) = \mathrm{coRP} \stackrel{?!}{=} \mathrm{P}$
  by definition; coRP is randomized polytime with one sided error (positive probability of accepting NO-instance)

Note that the first three statements also hold with equality

Probabilistic Checkable Proofs

- $\mathrm{PCP}(0, \mathrm{poly}(n)) = \mathrm{NP}$
  by definition; NP-verifier does not use randomness and asks polynomially many queries
- $\mathrm{PCP}(\log n, \mathrm{poly}(n)) \subseteq \mathrm{NP}$
  NP-verifier can simulate $\mathcal{O}(\log n)$ random bits
- $\mathrm{PCP}(\mathrm{poly}(n), 0) = \mathrm{coRP} \overset{?!}{\subseteq} \mathrm{NP}$
- $\mathrm{NP} \subseteq \mathrm{PCP}(\log n, 1)$
  hard part of the PCP-theorem

Probabilistic Checkable Proofs

- $\mathrm{PCP}(0, \mathrm{poly}(n)) = \mathrm{NP}$
  by definition; NP-verifier does not use randomness and asks polynomially many queries
- $\mathrm{PCP}(\log n, \mathrm{poly}(n)) \subseteq \mathrm{NP}$
  NP-verifier can simulate $\mathcal{O}(\log n)$ random bits
- $\mathrm{PCP}(\mathrm{poly}(n), 0) = \mathrm{coRP} \overset{?!}{\subseteq} \mathrm{NP}$
- $\mathrm{NP} \subseteq \mathrm{PCP}(\log n, 1)$
  hard part of the PCP-theorem

Probabilistic Checkable Proofs

- $\mathrm{PCP}(0, \mathrm{poly}(n)) = \mathrm{NP}$
  by definition; NP-verifier does not use randomness and asks polynomially many queries
- $\mathrm{PCP}(\log n, \mathrm{poly}(n)) \subseteq \mathrm{NP}$
  NP-verifier can simulate $\mathcal{O}(\log n)$ random bits
- $\mathrm{PCP}(\mathrm{poly}(n), 0) = \mathrm{coRP} \overset{?!}{\subseteq} \mathrm{NP}$
- $\mathrm{NP} \subseteq \mathrm{PCP}(\log n, 1)$
  hard part of the PCP-theorem

Probabilistic Checkable Proofs

- $\mathrm{PCP}(0, \mathrm{poly}(n)) = \mathrm{NP}$
  by definition; NP-verifier does not use randomness and asks polynomially many queries
- $\mathrm{PCP}(\log n, \mathrm{poly}(n)) \subseteq \mathrm{NP}$
  NP-verifier can simulate $\mathcal{O}(\log n)$ random bits
- $\mathrm{PCP}(\mathrm{poly}(n), 0) = \mathrm{coRP} \overset{?!}{\subseteq} \mathrm{NP}$
- $\mathrm{NP} \subseteq \mathrm{PCP}(\log n, 1)$
  hard part of the PCP-theorem

PCP theorem: Proof System View

Theorem 7 (PCP Theorem B)

$\mathrm{NP} = \mathrm{PCP}(\log n, 1)$

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

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

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 $G_0 \equiv G_1$ 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 $b = 0$ and permutation $\pi_{\text{rand}} \circ \pi$

Version B $\Longrightarrow$ Version A

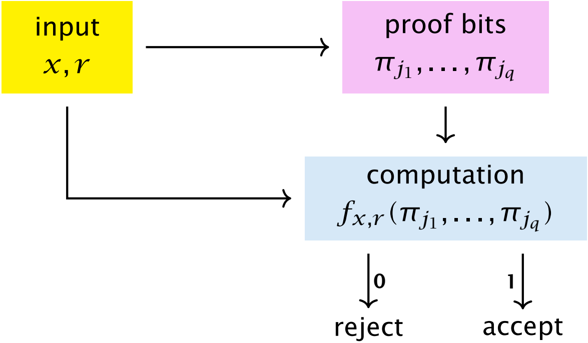
- For 3SAT there exists a verifier that uses $c \log n$ random bits, reads $q = \mathcal{O}(1)$ bits from the proof, has completeness 1 and soundness $1/2$.
- fix x and r:

Version B $\Longrightarrow$ Version A

- For 3SAT there exists a verifier that uses $c \log n$ random bits, reads $q = \mathcal{O}(1)$ bits from the proof, has completeness 1 and soundness $1/2$.
- fix x and r:

Version B $\Longrightarrow$ Version A

- For 3SAT there exists a verifier that uses $c \log n$ random bits, reads $q = \mathcal{O}(1)$ bits from the proof, has completeness 1 and soundness $1/2$.
- fix x and r:



Version B $\Longrightarrow$ Version A

- transform Boolean formula $f_{x,r}$ into 3SAT formula $C_{x,r}$ (constant size, variables are proof bits)
- consider 3SAT formula $C_x := \bigwedge_r C_{x,r}$

$[x \in L]$ There exists proof string y, s.t. all formulas $C_{x,r}$ evaluate to 1. Hence, all clauses in C_x satisfied.

$[x \notin L]$ For any proof string y, at most 50% of formulas $C_{x,r}$ evaluate to 1. Since each contains only a constant number of clauses, a constant fraction of clauses in C_x are not satisfied.

- this means we have gap introducing reduction

Version B $\implies$ Version A

- transform Boolean formula $f_{x,r}$ into 3SAT formula $C_{x,r}$ (constant size, variables are proof bits)
- consider 3SAT formula $C_x \coloneqq \bigwedge_r C_{x,r}$

$[x \in L]$ There exists proof string y, s.t. all formulas $C_{x,r}$ evaluate to 1. Hence, all clauses in C_x satisfied.

$[x \notin L]$ For any proof string y, at most 50% of formulas $C_{x,r}$ evaluate to 1. Since each contains only a constant number of clauses, a constant fraction of clauses in C_x are not satisfied.

- this means we have gap introducing reduction

Version B $\Longrightarrow$ Version A

- transform Boolean formula $f_{x,r}$ into 3SAT formula $C_{x,r}$ (constant size, variables are proof bits)
- consider 3SAT formula $C_x := \bigwedge_r C_{x,r}$

[$x \in L$] There exists proof string y, s.t. all formulas $C_{x,r}$ evaluate to 1. Hence, all clauses in C_x satisfied.

[$x \notin L$] For any proof string y, at most 50% of formulas $C_{x,r}$ evaluate to 1. Since each contains only a constant number of clauses, a constant fraction of clauses in C_x are not satisfied.

- this means we have gap introducing reduction

Version B $\Longrightarrow$ Version A

- transform Boolean formula $f_{x,r}$ into 3SAT formula $C_{x,r}$ (constant size, variables are proof bits)
- consider 3SAT formula $C_x \coloneqq \bigwedge_r C_{x,r}$

[$x \in L$] There exists proof string y, s.t. all formulas $C_{x,r}$ evaluate to 1. Hence, all clauses in C_x satisfied.

[$x \notin L$] For any proof string y, at most 50% of formulas $C_{x,r}$ evaluate to 1. Since each contains only a constant number of clauses, a constant fraction of clauses in C_x are not satisfied.

- this means we have gap introducing reduction

Version B $\Longrightarrow$ Version A

- transform Boolean formula $f_{x,r}$ into 3SAT formula $C_{x,r}$ (constant size, variables are proof bits)
- consider 3SAT formula $C_x := \bigwedge_r C_{x,r}$

[$x \in L$] There exists proof string y, s.t. all formulas $C_{x,r}$ evaluate to 1. Hence, all clauses in C_x satisfied.

[$x \notin L$] For any proof string y, at most 50% of formulas $C_{x,r}$ evaluate to 1. Since each contains only a constant number of clauses, a constant fraction of clauses in C_x are not satisfied.

- this means we have gap introducing reduction

Version A $\implies$ Version B

We show: Version A $\implies \mathrm{NP} \subseteq \mathrm{PCP}_{1,1-\epsilon}(\log n, 1)$.

given $L \in \mathrm{NP}$ we build a PCP-verifier for L

Verifier:

Version A $\Longrightarrow$ Version B

We show: Version A $\Longrightarrow \mathrm{NP} \subseteq \mathrm{PCP}_{1,1-\epsilon}(\log n, 1)$.

given $L \in \mathrm{NP}$ we build a PCP-verifier for L

Verifier:

- 3SAT is NP-complete; map instance x for L into 3SAT instance I_x, s.t. I_x satisfiable iff $x \in L$
- map I_x to MAX3SAT instance C_x
- interpret proof as assignment to variables in C_x
- choose random clause X from C_x
- query variable assignment σ for X;
- accept if $X(\sigma) = \text{true}$ otw. reject

Version A $\implies$ Version B

We show: Version A $\implies \mathrm{NP} \subseteq \mathrm{PCP}_{1,1-\epsilon}(\log n, 1)$.

given $L \in \mathrm{NP}$ we build a PCP-verifier for L

Verifier:

- 3SAT is NP-complete; map instance x for L into 3SAT instance I_x, s.t. I_x satisfiable iff $x \in L$
- map I_x to MAX3SAT instance C_x (PCP Thm. Version A)
- interpret proof as assignment to variables in C_x
- choose random clause X from C_x
- query variable assignment σ for X;
- accept if $X(\sigma) = \text{true}$ otw. reject

Version A $\Longrightarrow$ Version B

We show: Version A $\Longrightarrow \mathrm{NP} \subseteq \mathrm{PCP}_{1,1-\epsilon}(\log n, 1)$.

given $L \in \mathrm{NP}$ we build a PCP-verifier for L

Verifier:

- 3SAT is NP-complete; map instance x for L into 3SAT instance I_x, s.t. I_x satisfiable iff $x \in L$
- map I_x to MAX3SAT instance C_x (PCP Thm. Version A)
- interpret proof as assignment to variables in C_x
- choose random clause X from C_x
- query variable assignment σ for X;
- accept if $X(\sigma) =$ true otw. reject

Version A $\Longrightarrow$ Version B

We show: Version A $\Longrightarrow \mathrm{NP} \subseteq \mathrm{PCP}_{1,1-\epsilon}(\log n, 1)$.

given $L \in \mathrm{NP}$ we build a PCP-verifier for L

Verifier:

- 3SAT is NP-complete; map instance x for L into 3SAT instance I_x, s.t. I_x satisfiable iff $x \in L$
- map I_x to MAX3SAT instance C_x (PCP Thm. Version A)
- interpret proof as assignment to variables in C_x
- choose random clause X from C_x
- query variable assignment σ for X;
- accept if $X(\sigma) = \text{true}$ otw. reject

Version A $\Longrightarrow$ Version B

We show: Version A $\Longrightarrow \mathrm{NP} \subseteq \mathrm{PCP}_{1,1-\epsilon}(\log n, 1)$.

given $L \in \mathrm{NP}$ we build a PCP-verifier for L

Verifier:

- 3SAT is NP-complete; map instance x for L into 3SAT instance I_x, s.t. I_x satisfiable iff $x \in L$
- map I_x to MAX3SAT instance C_x (PCP Thm. Version A)
- interpret proof as assignment to variables in C_x
- choose random clause X from C_x
- query variable assignment σ for X;
- accept if $X(\sigma) = \text{true}$ otw. reject

Version A $\Longrightarrow$ Version B

We show: Version A $\Longrightarrow \mathrm{NP} \subseteq \mathrm{PCP}_{1,1-\epsilon}(\log n, 1)$.

given $L \in \mathrm{NP}$ we build a PCP-verifier for L

Verifier:

- 3SAT is NP-complete; map instance x for L into 3SAT instance I_x, s.t. I_x satisfiable iff $x \in L$
- map I_x to MAX3SAT instance C_x (PCP Thm. Version A)
- interpret proof as assignment to variables in C_x
- choose random clause X from C_x
- query variable assignment σ for X;
- accept if $X(\sigma) =$ true otw. reject

Version A $\Longrightarrow$ Version B

We show: Version A $\Longrightarrow \mathrm{NP} \subseteq \mathrm{PCP}_{1,1-\epsilon}(\log n, 1)$.

given $L \in \mathrm{NP}$ we build a PCP-verifier for L

Verifier:

- 3SAT is NP-complete; map instance x for L into 3SAT instance I_x, s.t. I_x satisfiable iff $x \in L$
- map I_x to MAX3SAT instance C_x (PCP Thm. Version A)
- interpret proof as assignment to variables in C_x
- choose random clause X from C_x
- query variable assignment σ for X;
- accept if $X(\sigma) = \text{true}$ otw. reject

Version A $\Longrightarrow$ Version B

[$x \in L$] There exists proof string y, s.t. all clauses in C_x evaluate to 1. In this case the verifier returns 1.

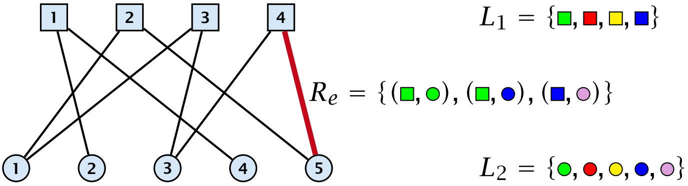
[$x \notin L$] For any proof string y, at most a $(1-\epsilon)$-fraction of clauses in C_x evaluate to 1. The verifier will reject with probability at least ϵ.

To show Theorem B we only need to run this verifier a constant number of times to push rejection probability above $1/2$.

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

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

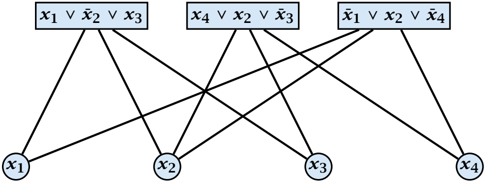
$$R = \{((F,F,F),F),((F,T,F),F),((F,F,T),T),((F,T,T),T),$$
$$((T,T,T),T),((T,T,F),F),((T,F,F),F)\}$$

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

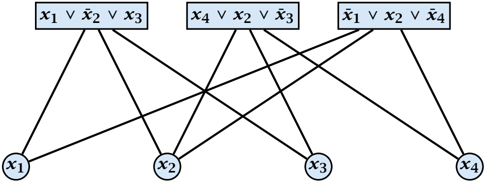
$$R = \{((F,F,F),F),((F,T,F),F),((F,F,T),T),((F,T,T),T),$$
$$((T,T,T),T),((T,T,F),F),((T,F,F),F)\}$$

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

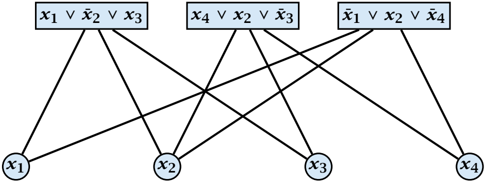
$$R = \{((F,F,F),F), ((F,T,F),F), ((F,F,T),T), ((F,T,T),T), ((T,T,T),T), ((T,T,F),F), ((T,F,F),F)\}$$

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

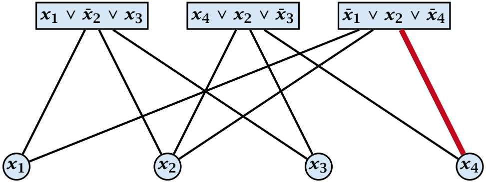
$$R = \{((F,F,F),F), ((F,T,F),F), ((F,F,T),T), ((F,T,T),T), ((T,T,T),T), ((T,T,F),F), ((T,F,F),F)\}$$

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 8

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 8
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 8
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 8

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 9

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 9

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 9

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 9

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 $2m + (1-\epsilon)m = (3-\epsilon)m$ out of the $3m$ edges can be made happy

Hence, we cannot obtain an approximation constant $\alpha > \frac{3-\epsilon}{3}$.

Hardness for Label Cover

We cannot distinguish between the following two cases

- all $3m$ edges can be made happy
- at most $2m + (1-\epsilon)m = (3-\epsilon)m$ out of the $3m$ edges can be made happy

Hence, we cannot obtain an approximation constant $\alpha > \frac{3-\epsilon}{3}$.

$(3,5)$-regular instances

Theorem 10
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 10
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

The previous theorem can be obtained with a series of gap-preserving reductions:

- MAX3SAT $\leq$ MAX3SAT(≤ 29)
- MAX3SAT(≤ 29) $\leq$ MAX3SAT(≤ 5)
- MAX3SAT(≤ 5) $\leq$ MAX3SAT$(= 5)$
- MAX3SAT$(= 5)$ $\leq$ MAXE3SAT$(= 5)$

Here MAX3SAT(≤ 29) is the variant of MAX3SAT in which a variable appears in at most 29 clauses. Similar for the other problems.

Regular instances

Theorem 11
There is a constant $\alpha < 1$ such if 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)

Parallel Repetition

We would like to increase the inapproximability for Label Cover.

In the verifier view, in order to decrease the acceptance probability of a wrong proof (or as here: a pair of wrong proofs) one could repeat the verification several times.

Unfortunately, we have a 2P1R-system, i.e., we are stuck with a single round and cannot simply repeat.

The idea is to use parallel repetition, i.e., we simply play several rounds in parallel and hope that the acceptance probability of wrong proofs goes down.

Parallel Repetition

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

Parallel Repetition

If I is regular than also I'.

If I has the uniqueness property than also I'.

Did the gap increase?

- Suppose we have labelling ℓ_L, ℓ_R that satisfies just an α-fraction of edges in I.
- We transfer this labelling to instance I': vertex $(v_1, \dots, v_k)$ gets label $(\ell_L(v_1), \dots, \ell_L(v_k))$, vertex $(u_1, \dots, u_k)$ gets label $(\ell_R(u_1), \dots, \ell_R(u_k))$.
- How many edges are happy?

Does this always work?

Parallel Repetition

If I is regular than also I'.

If I has the uniqueness property than also I'.

Did the gap increase?

Parallel Repetition

If I is regular than also I'.

If I has the uniqueness property than also I'.

Did the gap increase?

- Suppose we have labelling ℓ_1, ℓ_2 that satisfies just an α-fraction of edges in I.
- We transfer this labelling to instance I':
  vertex $(x_1, \dots, x_k)$ gets label $(\ell_1(x_1), \dots, \ell_1(x_k))$,
  vertex $(y_1, \dots, y_k)$ gets label $(\ell_2(y_1), \dots, \ell_2(y_k))$.
- How many edges are happy?
  only $(\alpha|E|)^k$ out of $|E|^k$!!! (just an α^k fraction)

Does this always work?

Parallel Repetition

If I is regular than also I'.

If I has the uniqueness property than also I'.

Did the gap increase?

- Suppose we have labelling ℓ_1, ℓ_2 that satisfies just an α-fraction of edges in I.
- We transfer this labelling to instance I':
  vertex $(x_1, \dots, x_k)$ gets label $(\ell_1(x_1), \dots, \ell_1(x_k))$,
  vertex $(y_1, \dots, y_k)$ gets label $(\ell_2(y_1), \dots, \ell_2(y_k))$.
- How many edges are happy?
  only $(\alpha|E|)^k$ out of $|E|^k$!!! (just an α^k fraction)

Does this always work?

Parallel Repetition

If I is regular than also I'.

If I has the uniqueness property than also I'.

Did the gap increase?

- Suppose we have labelling ℓ_1, ℓ_2 that satisfies just an α-fraction of edges in I.
- We transfer this labelling to instance I':
  vertex $(x_1, \dots, x_k)$ gets label $(\ell_1(x_1), \dots, \ell_1(x_k))$,
  vertex $(y_1, \dots, y_k)$ gets label $(\ell_2(y_1), \dots, \ell_2(y_k))$.
- How many edges are happy?
  only $(\alpha|E|)^k$ out of $|E|^k$!!! (just an α^k fraction)

Does this always work?

Parallel Repetition

If I is regular than also I'.

If I has the uniqueness property than also I'.

Did the gap increase?

- Suppose we have labelling ℓ_1, ℓ_2 that satisfies just an α-fraction of edges in I.
- We transfer this labelling to instance I':
  vertex $(x_1, \ldots, x_k)$ gets label $(\ell_1(x_1), \ldots, \ell_1(x_k))$,
  vertex $(y_1, \ldots, y_k)$ gets label $(\ell_2(y_1), \ldots, \ell_2(y_k))$.
- How many edges are happy?
  only $(\alpha|E|)^k$ out of $|E|^k$!!! (just an α^k fraction)

Does this always work?

Parallel Repetition

If I is regular than also I'.

If I has the uniqueness property than also I'.

Did the gap increase?

- Suppose we have labelling ℓ_1, ℓ_2 that satisfies just an α-fraction of edges in I.
- We transfer this labelling to instance I':
  vertex $(x_1, \dots, x_k)$ gets label $(\ell_1(x_1), \dots, \ell_1(x_k))$,
  vertex $(y_1, \dots, y_k)$ gets label $(\ell_2(y_1), \dots, \ell_2(y_k))$.
- How many edges are happy?
  only $(\alpha|E|)^k$ out of $|E|^k$!!! (just an α^k fraction)

Does this always work?

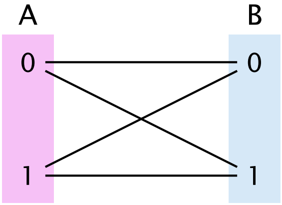
Counter Example

Non interactive agreement:

- Two provers A and B
- The verifier generates two random bits b_A, and b_B, and sends one to A and one to B.
- Each prover has to answer one of A_0, A_1, B_0, B_1 with the meaning $A_0 :=$ prover A has been given a bit with value 0.
- The provers win if they give the same answer and if the answer is correct.

Counter Example

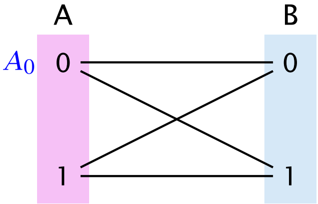
The provers can win with probability at most $1/2$.



Regardless what we do 50% of edges are unhappy!

Counter Example

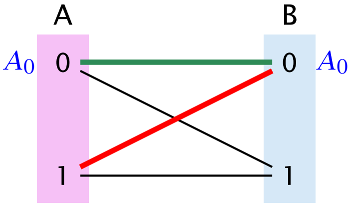
The provers can win with probability at most $1/2$.



Regardless what we do 50% of edges are unhappy!

Counter Example

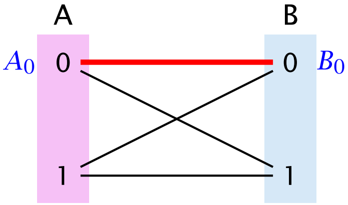
The provers can win with probability at most $1/2$.



Regardless what we do 50% of edges are unhappy!

Counter Example

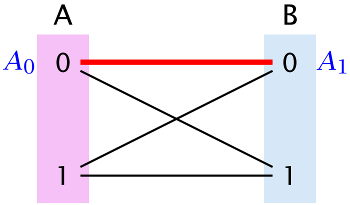
The provers can win with probability at most $1/2$.



Regardless what we do 50% of edges are unhappy!

Counter Example

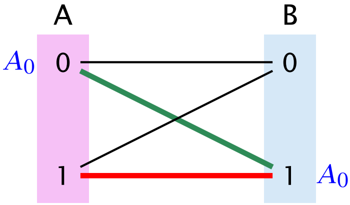
The provers can win with probability at most $1/2$.



Regardless what we do 50% of edges are unhappy!

Counter Example

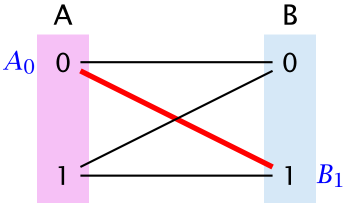
The provers can win with probability at most $1/2$.



Regardless what we do 50% of edges are unhappy!

Counter Example

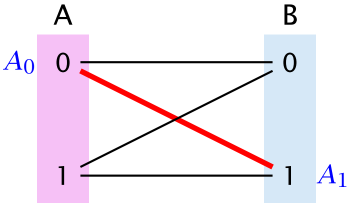
The provers can win with probability at most $1/2$.



Regardless what we do 50% of edges are unhappy!

Counter Example

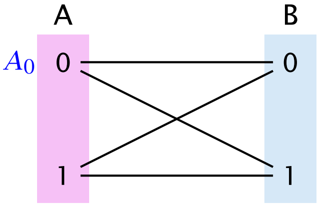
The provers can win with probability at most $1/2$.



Regardless what we do 50% of edges are unhappy!

Counter Example

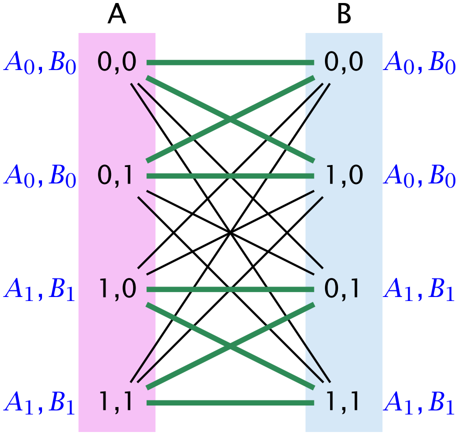
The provers can win with probability at most $1/2$.



Regardless what we do 50% of edges are unhappy!

Counter Example

In the repeated game the provers can also win with probability $1/2$:



Boosting

Theorem 12
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

Theorem 12
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

Hardness of Label Cover

Theorem 13

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*
- $\mathrm{OPT}(I) = |E|(1-\delta)^{ck}$

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

Corollary 14

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

Advanced PCP Theorem

Theorem 15

For any positive constant $\epsilon > 0$, it is the case that $\mathrm{NP} \subseteq \mathrm{PCP}_{1-\epsilon,1/2+\epsilon}(\log n, 3)$. Moreover, the verifier just reads three bits from the proof, and bases its decision only on the parity of these bits.

It is NP-hard to approximate a MAXE3LIN problem by a factor better than $1/2 + \delta$, for any constant δ.

It is NP-hard to approximate MAX3SAT better than $7/8 + \delta$, for any constant δ.