

# Traveling Salesman

Given a set of cities  $(\{1, \dots, n\})$  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  $\pi$  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 96

*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. We take the graph  $G = (V, E)$  and set  $w_{ij} = 1$  if  $(i, j) \in E$  and  $w_{ij} = 2$  otherwise. The instance has polynomial size.

There exists a Hamiltonian Path in  $G$  if and only if there exists a TSP tour of cost  $n$ . Conversely, any tour has cost  $n$  if and only if it is a Hamiltonian cycle. A polynomial-time algorithm could use the cost of a TSP tour to decide if a Hamiltonian cycle exists. This is impossible.

# Traveling Salesman

## Theorem 96

*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 96

*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 96

*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 96

*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 96

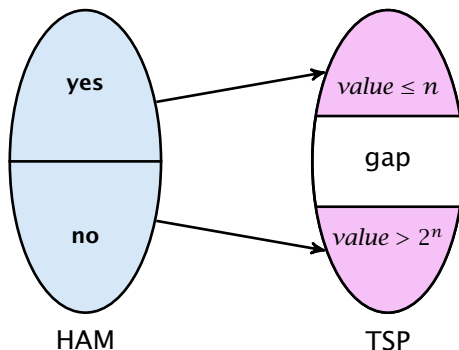
*There does not exist an  $O(2^n)$ -approximation algorithm for TSP.*

## 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  $P = NP$ .

## Gap Introducing Reduction



### Reduction from Hamiltonian cycle to TSP

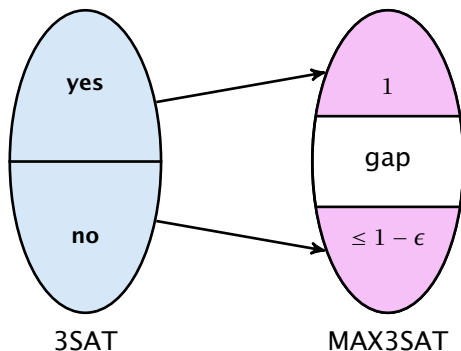
- ▶ instance that has Hamiltonian cycle is mapped to TSP instance with small cost
- ▶ otherwise it is mapped to instance with large cost
- ▶  $\Rightarrow$  there is no  $2^n/n$ -approximation for TSP



## PCP theorem: Approximation View

### Theorem 97 (PCP Theorem A)

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



# PCP theorem: Proof System View

## Definition 98 (NP)

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

### $[x \in L]$ **completeness**

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

### $[x \notin L]$ **soundness**

For any proof string  $y$ ,  $V(x, y) = \text{"reject"}$ .

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

# PCP theorem: Proof System View

## Definition 98 (NP)

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

### $[x \in L]$ **completeness**

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

### $[x \notin L]$ **soundness**

For any proof string  $y$ ,  $V(x, y) = \text{"reject"}$ .

Note that requiring  $|y| = \text{poly}(|x|)$  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  $\pi_{TSP}$  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  $y$ ,  $\pi_y$  is an oracle that upon given an index  $i$  returns the  $i$ -th character  $y_i$  of  $y$ .

# Probabilistic Checkable Proofs

## Definition 99 (PCP)

A language  $L \in \text{PCP}_{c(n),s(n)}(r(n), q(n))$  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 otherwise  $c(n) = 1$ .  
Probability of accepting a correct proof.

$s(n) < c(n)$  is called the **soundness**. If not specified otherwise  $s(n) = 1/2$ .  
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

- ▶  $P = PCP(0, 0)$

verifier without randomness and proof access is deterministic algorithm

- ▶  $PCP(\log n, 0) \subseteq P$

we can simulate a verifier with 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-instances)

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 long proofs by a deterministic algorithm in 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, exists 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

- ▶  $P = \text{PCP}(0, 0)$   
verifier without randomness and proof access is deterministic algorithm
- ▶  $\text{PCP}(\log n, 0) \subseteq P$   
we can simulate  $O(\log n)$  random bits in deterministic, polynomial time
- ▶  $\text{PCP}(0, \log n) \subseteq P$   
we can simulate short proofs in polynomial time
- ▶  $\text{PCP}(\text{poly}(n), 0) = \text{coRP} \stackrel{?!}{=} P$   
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

- ▶  $P = \text{PCP}(0, 0)$   
verifier without randomness and proof access is deterministic algorithm
- ▶  $\text{PCP}(\log n, 0) \subseteq P$   
we can simulate  $O(\log n)$  random bits in deterministic, polynomial time
- ▶  $\text{PCP}(0, \log n) \subseteq P$   
we can simulate short proofs in polynomial time
- ▶  $\text{PCP}(\text{poly}(n), 0) = \text{coRP} \stackrel{?!}{=} P$   
by definition, verifier randomized polynomial with one-sided error (positive probability of accepting NP-Instance)

Note that the first three statements also hold with equality

# Probabilistic Checkable Proofs

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

Note that the first three statements also hold with equality

# Probabilistic Checkable Proofs

- ▶  $P = \text{PCP}(0, 0)$   
verifier without randomness and proof access is deterministic algorithm
- ▶  $\text{PCP}(\log n, 0) \subseteq P$   
we can simulate  $O(\log n)$  random bits in deterministic, polynomial time
- ▶  $\text{PCP}(0, \log n) \subseteq P$   
we can simulate short proofs in polynomial time
- ▶  $\text{PCP}(\text{poly}(n), 0) = \text{coRP} \stackrel{?!}{=} P$   
by definition, verifier is randomized, polynomial time with one-sided error (positive probability of accepting false instance)

Note that the first three statements also hold with equality

# Probabilistic Checkable Proofs

- ▶  $P = \text{PCP}(0, 0)$   
verifier without randomness and proof access is deterministic algorithm
- ▶  $\text{PCP}(\log n, 0) \subseteq P$   
we can simulate  $O(\log n)$  random bits in deterministic, polynomial time
- ▶  $\text{PCP}(0, \log n) \subseteq P$   
we can simulate short proofs in polynomial time
- ▶  $\text{PCP}(\text{poly}(n), 0) = \text{coRP} \stackrel{?!}{=} P$   
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

- ▶  $P = PCP(0, 0)$   
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

- ▶  $P = PCP(0, 0)$   
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;  $\text{coRP}$  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

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



# Probabilistic Checkable Proofs

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

# Probabilistic Checkable Proofs

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

# Probabilistic Checkable Proofs

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

# PCP theorem: Proof System View

## Theorem 100 (PCP Theorem B)

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

# 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  $G_0 \not\cong G_1$  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  $P[H] = b$

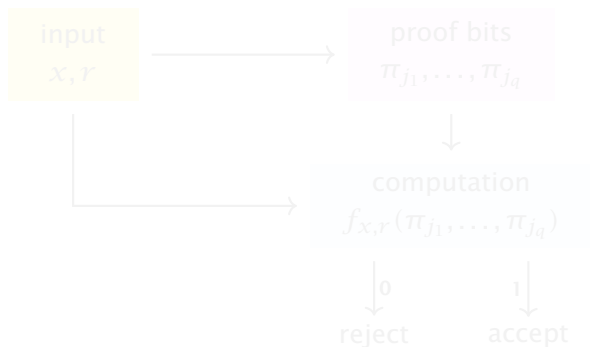
If  $G_0 \not\equiv G_1$  then by using the obvious proof the verifier will always accept.

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  $\pi_{\text{rand}}$  we reject for  $b = 0$  and permutation  $\pi_{\text{rand}} \circ \pi$

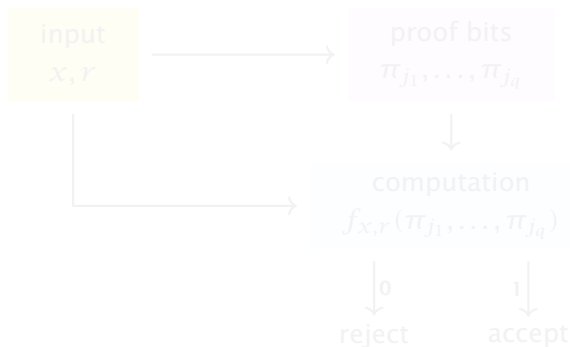
## Version B $\Rightarrow$ Version A

- ▶ 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  $r$ :



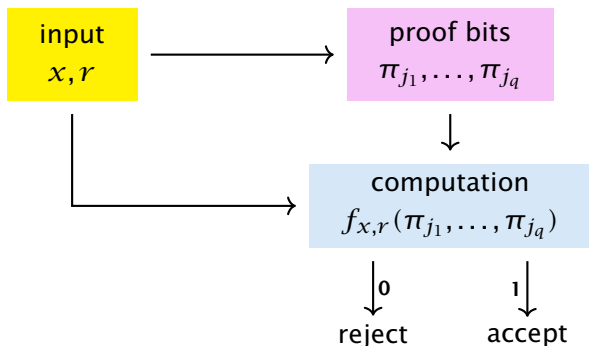
## Version B $\Rightarrow$ Version A

- ▶ 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 $\Rightarrow$ 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 $\Rightarrow$ 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 $\Rightarrow$ 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 $\Rightarrow$ 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 $\Rightarrow$ 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 $\Rightarrow$ 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 $\Rightarrow$ Version B

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

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

**Verifier:**

1. SAT is NP-complete; map instance  $\phi$  for  $L$  into SAT

instance  $\phi = \{C_1, \dots, C_m\}$  satisfiable iff

2.  $\exists \text{ assignment } \alpha$  to MAXSAT instance  $\phi - 1$

3. interpret  $\alpha$  as assignment to variables in  $\phi$

4. choose random clause  $C_i \in \phi$

5. query variable assignment  $\alpha$  for

6. output 1 if  $C_i$  is satisfied

## Version A $\Rightarrow$ Version B

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

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

Verifier:

1. Given an NP-completeness instance  $x$  for  $L$ , run  $V(x)$ .

2. If  $V(x) = \text{ACCEPT}$ , then  $x \in L$ .

3. If  $V(x) = \text{REJECT}$ , then  $x \notin L$ .

4. If  $V(x) = \text{REJECT}$ , then  $x \notin L$ .

5. If  $V(x) = \text{ACCEPT}$ , then  $x \in L$ .

6. If  $V(x) = \text{REJECT}$ , then  $x \notin L$ .

7. If  $V(x) = \text{ACCEPT}$ , then  $x \in L$ .

8. If  $V(x) = \text{REJECT}$ , then  $x \notin L$ .

9. If  $V(x) = \text{ACCEPT}$ , then  $x \in L$ .

10. If  $V(x) = \text{REJECT}$ , then  $x \notin L$ .

## Version A $\Rightarrow$ Version B

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

given  $L \in \text{NP}$  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  $\sigma$  for  $X$ ;
- ▶ accept if  $X(\sigma) = \text{true}$  otw. reject

## Version A $\Rightarrow$ Version B

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

given  $L \in \text{NP}$  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  $\sigma$  for  $X$ ;
- ▶ accept if  $X(\sigma) = \text{true}$  otw. reject

## Version A $\Rightarrow$ Version B

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

given  $L \in \text{NP}$  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  $\sigma$  for  $X$ ;
- ▶ accept if  $X(\sigma) = \text{true}$  otw. reject

## Version A $\Rightarrow$ Version B

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

given  $L \in \text{NP}$  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  $\sigma$  for  $X$ ;
  - ▶ accept if  $X(\sigma) = \text{true}$  otw. reject



## Version A $\Rightarrow$ Version B

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

given  $L \in \text{NP}$  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  $\sigma$  for  $X$ ;
- ▶ accept if  $X(\sigma) = \text{true}$  otw. reject

## Version A $\Rightarrow$ Version B

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

given  $L \in \text{NP}$  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  $\sigma$  for  $X$ ;
- ▶ accept if  $X(\sigma) = \text{true}$  otw. reject

## Version A $\Rightarrow$ 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  $\epsilon$ .

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

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

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

The idea is to encode an NP-witness (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.

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

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

The idea is to encode an NP-witness (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.

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

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

The idea is to encode an NP-witness (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:**

$\text{WH}_u : \{0, 1\}^n \rightarrow \{0, 1\}, x \mapsto x^T u$  (over  $\text{GF}(2)$ )

The code-word for  $u$  is  $\text{WH}_u$ . We identify this function by a bit-vector of length  $2^n$ .

# The Code

## Lemma 101

If  $u \neq u'$  then  $WH_u$  and  $WH_{u'}$  differ in at least  $2^{n-1}$  bits.

Proof:

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

$$WH_u(x) \neq WH_{u'}(x) \iff (u - u')^T x \neq 0$$

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



# The Code

## Lemma 101

If  $u \neq u'$  then  $WH_u$  and  $WH_{u'}$  differ in at least  $2^{n-1}$  bits.

### Proof:

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

$$WH_u(x) \neq WH_{u'}(x) \iff (u - u')^T x \neq 0$$

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

# The Code

Suppose we are given access to a function  $f : \{0, 1\}^n \rightarrow \{0, 1\}$  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.

# 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.**

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

Can we just check a constant number of positions?

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

## Definition 102

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

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

## Theorem 103 (proof deferred)

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

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

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

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

## Definition 102

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

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

## Theorem 103 (proof deferred)

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

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

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

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

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

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

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?



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

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?

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

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?

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

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 the above routine returns  $\tilde{f}(x)$ .

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

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

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 the above routine returns  $\tilde{f}(x)$ .

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

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

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 the above routine returns  $\tilde{f}(x)$ .

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

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

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

## QUADEQ

Given a system of quadratic equations over  $GF(2)$ . Is there a solution?

# QUADEQ is NP-complete

- ▶ given 3SAT instance  $C$  represent it as Boolean circuit  
e.g.  $C = (x_1 \vee x_2 \vee x_3) \wedge (x_3 \vee x_4 \vee \bar{x}_5) \wedge (x_6 \vee x_7 \vee x_8)$

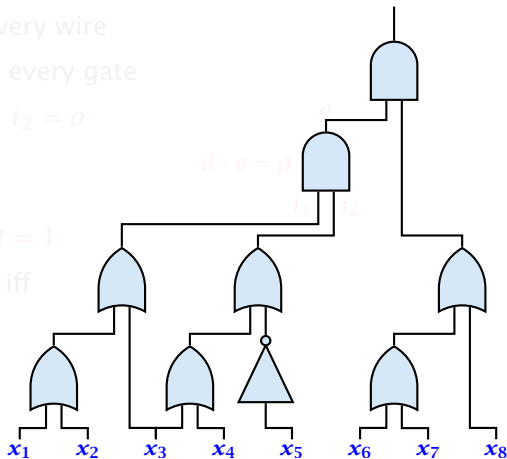
- ▶ add variable for every wire
- ▶ add constraint for every gate

OR:  $i_1 + i_2 + i_1 \cdot i_2 = 0$

AND:  $i_1 \cdot i_2 = 0$

NEG:  $i = 1 - o$

- ▶ add constraint  $out = 1$
- ▶ system is feasible iff  $C$  is satisfiable



# QUADEQ is NP-complete

- ▶ given 3SAT instance  $C$  represent it as Boolean circuit  
e.g.  $C = (x_1 \vee x_2 \vee x_3) \wedge (x_3 \vee x_4 \vee \bar{x}_5) \wedge (x_6 \vee x_7 \vee x_8)$
- ▶ add variable for every wire

- ▶ add constraint for every gate

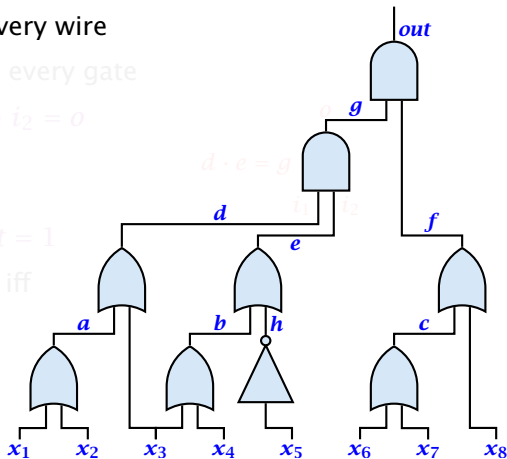
OR:  $i_1 + i_2 + i_1 \cdot i_2 = 0$

AND:  $i_1 \cdot i_2 = 0$

NEG:  $i = 1 - o$

- ▶ add constraint  $out = 1$

- ▶ system is feasible iff  
 $C$  is satisfiable





# QUADEQ is NP-complete

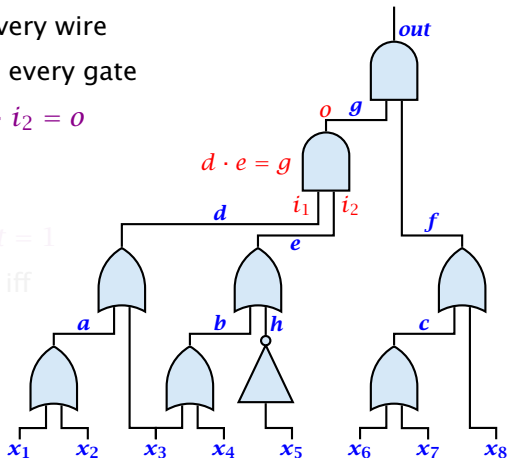
- ▶ given 3SAT instance  $C$  represent it as Boolean circuit  
e.g.  $C = (x_1 \vee x_2 \vee x_3) \wedge (x_3 \vee x_4 \vee \bar{x}_5) \wedge (x_6 \vee x_7 \vee x_8)$
- ▶ add variable for every wire
- ▶ add constraint for every gate

OR:  $i_1 + i_2 + i_1 \cdot i_2 = 0$

AND:  $i_1 \cdot i_2 = 0$

NEG:  $i = 1 - o$

- ▶ add constraint  $out = 1$
- ▶ system is feasible iff  $C$  is satisfiable



# QUADEQ is NP-complete

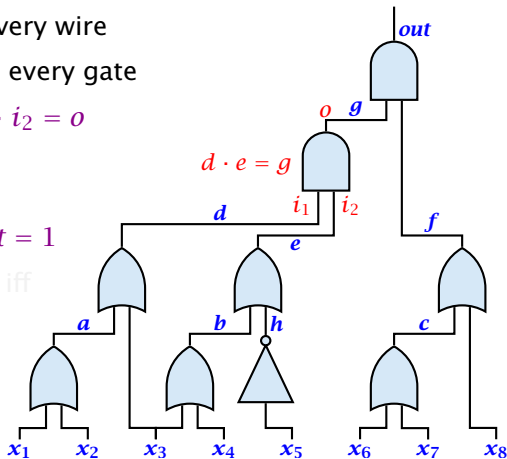
- ▶ given 3SAT instance  $C$  represent it as Boolean circuit  
e.g.  $C = (x_1 \vee x_2 \vee x_3) \wedge (x_3 \vee x_4 \vee \bar{x}_5) \wedge (x_6 \vee x_7 \vee x_8)$
- ▶ add variable for every wire
- ▶ add constraint for every gate

OR:  $i_1 + i_2 + i_1 \cdot i_2 = 0$

AND:  $i_1 \cdot i_2 = 0$

NEG:  $i = 1 - o$

- ▶ add constraint  $out = 1$
- ▶ system is feasible iff  $C$  is satisfiable



# QUADEQ is NP-complete

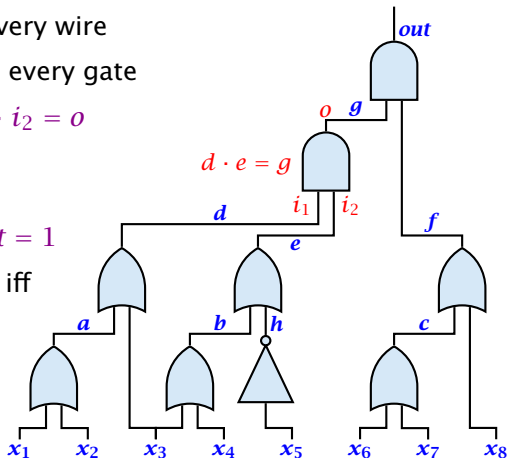
- ▶ given 3SAT instance  $C$  represent it as Boolean circuit  
e.g.  $C = (x_1 \vee x_2 \vee x_3) \wedge (x_3 \vee x_4 \vee \bar{x}_5) \wedge (x_6 \vee x_7 \vee x_8)$
- ▶ add variable for every wire
- ▶ add constraint for every gate

OR:  $i_1 + i_2 + i_1 \cdot i_2 = 0$

AND:  $i_1 \cdot i_2 = 0$

NEG:  $i = 1 - o$

- ▶ add constraint  $out = 1$
- ▶ system is feasible iff  $C$  is satisfiable



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

We encode an instance of **QUADEQ** by a matrix  $A$  that has  $n^2$  columns; one for every pair  $i, j$ ; and a right hand side vector  $b$ .

For an  $n$ -dimensional vector  $x$  we use  $x \otimes x$  to denote the  $n^2$ -dimensional vector whose  $i, j$ -th entry is  $x_i x_j$ .

Then we are asked whether

$$A(x \otimes x) = b$$

has a solution.

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

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.

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

## Step 1. Linearity Test.

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

We do a 0.999-linearity test for both functions (requires a constant number of queries).

We also assume that for the remaining constant number of accesses WH-decoding succeeds and we recover  $\tilde{f}(x)$ .

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

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

## Step 1. Linearity Test.

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

We do a 0.999-linearity test for both functions (requires a constant number of queries).

We also assume that for the remaining constant number of accesses WH-decoding succeeds and we recover  $\tilde{f}(x)$ .

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

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

## Step 1. Linearity Test.

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

We do a 0.999-linearity test for both functions (requires a constant number of queries).

We also assume that for the remaining constant number of accesses WH-decoding succeeds and we recover  $\tilde{f}(x)$ .

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



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

## Step 1. Linearity Test.

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

We do a 0.999-linearity test for both functions (requires a constant number of queries).

We also assume that for the remaining constant number of accesses WH-decoding succeeds and we recover  $\tilde{f}(x)$ .

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

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

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

**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

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

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

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

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

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

$$\begin{aligned} f(r) \cdot f(r') &= u^T r \cdot u^T r' \\ &= \left( \sum_i u_i r_i \right) \cdot \left( \sum_j u_j r'_j \right) \end{aligned}$$

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

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

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

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



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

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

where  $U$  is matrix with  $U_{ij} = u_i \cdot u_j$

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

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

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  $u \otimes u$ ).

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

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  $u \otimes u$ ).

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

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

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  $u \otimes u$ ).

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

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

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  $u \otimes u$ ).

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

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

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  $u \otimes u$ ).

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

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

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  $u \otimes u$ ).

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

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



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

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  $u \otimes u$ ).

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

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

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

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  $u \otimes u$ ).

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

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

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

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

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

**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  $r^T A$ , where  $r \in_{\mathcal{R}} \{0, 1\}^m$ . If  $u$  is not a satisfying assignment then with probability 1/2 the vector  $r$  will hit an odd number of violated constraints.

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

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

**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  $r^T A$ , 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 constraints.

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

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

**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  $r^T A$ , 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 constraints.

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

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

**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  $r^T A$ , 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 constraints.

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

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

We used the following theorem for the linearity test:

## Theorem 103

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

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

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



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

## Fourier Transform over 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  $f : \{-1, 1\}^n \rightarrow \mathbb{R}$  form a  $2^n$ -dimensional **Hilbert space**.

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

## Hilbert space

- ▶ addition  $(f + g)(x) = f(x) + g(x)$
- ▶ scalar multiplication  $(\alpha f)(x) = \alpha f(x)$
- ▶ inner product  $\langle f, g \rangle = E_{x \in \{-1, 1\}^n} [f(x)g(x)]$   
(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

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

for some vector  $L$ .

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

## standard basis

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

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

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

## fourier basis

For  $\alpha \subseteq [n]$  define

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

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

## fourier basis

For  $\alpha \subseteq [n]$  define

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

Note that

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

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

## fourier basis

For  $\alpha \subseteq [n]$  define

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

Note that

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

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

## fourier basis

For  $\alpha \subseteq [n]$  define

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

Note that

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

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

## fourier basis

For  $\alpha \subseteq [n]$  define

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

Note that

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



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

## fourier basis

For  $\alpha \subseteq [n]$  define

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

Note that

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

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

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

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

This means the function  $\chi_\alpha$  correspond to linear functions in the GF(2) world.

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

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

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

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

### Lemma 104

1.  $\langle f, g \rangle = \sum_{\alpha} \hat{f}_{\alpha} \hat{g}_{\alpha}$
2.  $\langle f, f \rangle = \sum_{\alpha} \hat{f}_{\alpha}^2$

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

# Linearity Test

**in 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

**in 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.

**in Hilbert space:** (we will prove)

Suppose  $f : \{\pm 1\}^n \rightarrow \{-1, 1\}$  fulfills

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

Then there is some  $\alpha \subseteq [n]$ , s.t.  $\hat{f}_\alpha \geq 2\epsilon$ .

# Linearity Test

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

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

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

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

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

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  $\chi_\alpha$  is at least  $\frac{1}{2} + \epsilon$ .

# Linearity Test

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

means that the fraction of inputs  $x, y$  on which  $f(x \circ y)$  and  $f(x)f(y)$  agree is at least  $1/2 + \epsilon$ .

This gives

$$\begin{aligned} E_{x,y}[f(x \circ y)f(x)f(y)] &= \text{agreement} - \text{disagreement} \\ &= 2\text{agreement} - 1 \\ &\geq 2\epsilon \end{aligned}$$

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

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

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

$$\begin{aligned}
2\epsilon &\leq E_{x,y} \left[ f(x \circ y) f(x) f(y) \right] \\
&= E_{x,y} \left[ \left( \sum_{\alpha} \hat{f}_{\alpha} \chi_{\alpha}(x \circ y) \right) \cdot \left( \sum_{\beta} \hat{f}_{\beta} \chi_{\beta}(x) \right) \cdot \left( \sum_{\gamma} \hat{f}_{\gamma} \chi_{\gamma}(y) \right) \right] \\
&= E_{x,y} \left[ \sum_{\alpha,\beta,\gamma} \hat{f}_{\alpha} \hat{f}_{\beta} \hat{f}_{\gamma} \chi_{\alpha}(x) \chi_{\alpha}(y) \chi_{\beta}(x) \chi_{\gamma}(y) \right] \\
&= \sum_{\alpha,\beta,\gamma} \hat{f}_{\alpha} \hat{f}_{\beta} \hat{f}_{\gamma} \cdot E_x \left[ \chi_{\alpha}(x) \chi_{\beta}(x) \right] E_y \left[ \chi_{\alpha}(y) \chi_{\gamma}(y) \right]
\end{aligned}$$



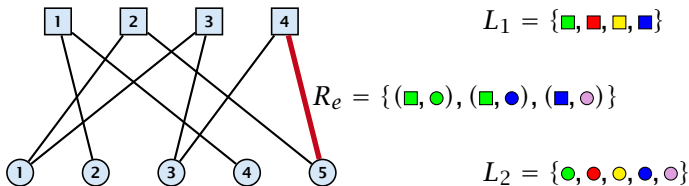
$$\begin{aligned}
2\epsilon &\leq E_{x,y} \left[ f(x \circ y) f(x) f(y) \right] \\
&= E_{x,y} \left[ \left( \sum_{\alpha} \hat{f}_{\alpha} \chi_{\alpha}(x \circ y) \right) \cdot \left( \sum_{\beta} \hat{f}_{\beta} \chi_{\beta}(x) \right) \cdot \left( \sum_{\gamma} \hat{f}_{\gamma} \chi_{\gamma}(y) \right) \right] \\
&= E_{x,y} \left[ \sum_{\alpha,\beta,\gamma} \hat{f}_{\alpha} \hat{f}_{\beta} \hat{f}_{\gamma} \chi_{\alpha}(x) \chi_{\alpha}(y) \chi_{\beta}(x) \chi_{\gamma}(y) \right] \\
&= \sum_{\alpha,\beta,\gamma} \hat{f}_{\alpha} \hat{f}_{\beta} \hat{f}_{\gamma} \cdot E_x \left[ \chi_{\alpha}(x) \chi_{\beta}(x) \right] E_y \left[ \chi_{\alpha}(y) \chi_{\gamma}(y) \right] \\
&= \sum_{\alpha} \hat{f}_{\alpha}^3
\end{aligned}$$

$$\begin{aligned}
2\epsilon &\leq E_{x,y} \left[ f(x \circ y) f(x) f(y) \right] \\
&= E_{x,y} \left[ \left( \sum_{\alpha} \hat{f}_{\alpha} \chi_{\alpha}(x \circ y) \right) \cdot \left( \sum_{\beta} \hat{f}_{\beta} \chi_{\beta}(x) \right) \cdot \left( \sum_{\gamma} \hat{f}_{\gamma} \chi_{\gamma}(y) \right) \right] \\
&= E_{x,y} \left[ \sum_{\alpha,\beta,\gamma} \hat{f}_{\alpha} \hat{f}_{\beta} \hat{f}_{\gamma} \chi_{\alpha}(x) \chi_{\alpha}(y) \chi_{\beta}(x) \chi_{\gamma}(y) \right] \\
&= \sum_{\alpha,\beta,\gamma} \hat{f}_{\alpha} \hat{f}_{\beta} \hat{f}_{\gamma} \cdot E_x \left[ \chi_{\alpha}(x) \chi_{\beta}(x) \right] E_y \left[ \chi_{\alpha}(y) \chi_{\gamma}(y) \right] \\
&= \sum_{\alpha} \hat{f}_{\alpha}^3 \\
&\leq \max_{\alpha} \hat{f}_{\alpha} \cdot \sum_{\alpha} \hat{f}_{\alpha}^2 = \max_{\alpha} \hat{f}_{\alpha}
\end{aligned}$$

# 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  $C$

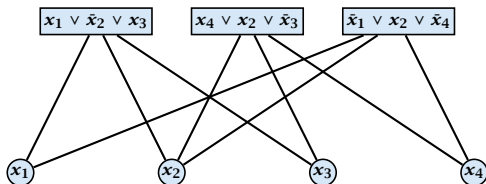
$$R = \{((F, F, F), F), ((F, T, F), F), ((F, F, T), T), ((F, T, T), T), \\ ((T, T, T), T), ((T, T, F), F), ((T, F, F), F)\}$$

# 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  $C$

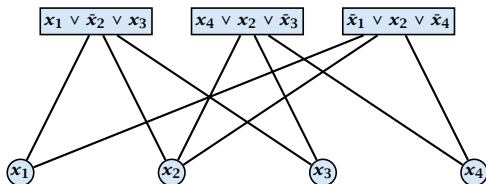
$$R = \{((F, F, F), F), ((F, T, F), F), ((F, F, T), T), ((F, T, T), T), ((T, T, T), T), ((T, T, F), F), ((T, F, F), F)\}$$

# 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  $C$

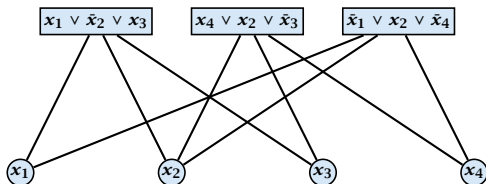
$$R = \{((F, F, F), F), ((F, T, F), F), ((F, F, T), T), ((F, T, T), T), ((T, T, T), T), ((T, T, F), F), ((T, F, F), F)\}$$

# 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  $C$

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

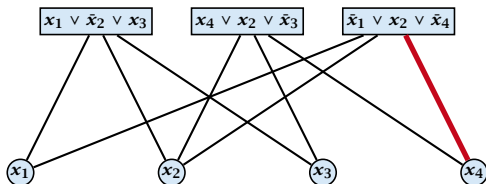


# MAX E3SAT via Label Cover

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 105

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

Proof:

# MAX E3SAT via Label Cover

## Lemma 105

If we can satisfy  $k$  out of  $m$  clauses in  $\phi$  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 105

If we can satisfy  $k$  out of  $m$  clauses in  $\phi$  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 105

If we can satisfy  $k$  out of  $m$  clauses in  $\phi$  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 106

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

Proof:

# MAX E3SAT via Label Cover

## Lemma 106

*If we can satisfy at most  $k$  clauses in  $\Phi$  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 106

*If we can satisfy at most  $k$  clauses in  $\Phi$  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



## Lemma 106

*If we can satisfy at most  $k$  clauses in  $\Phi$  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  $3k + 2(m - k) = 2m + k$  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 107

*There is a constant  $\rho$  s.t. MAXE3SAT is hard to approximate with a factor of  $\rho$  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  $\ell_1$  for  $x$  there is at most one label  $\ell_2$  for  $y$  that makes edge  $(x, y)$  happy (uniqueness property)

## (3, 5)-regular instances

### Theorem 107

*There is a constant  $\rho$  s.t. MAXE3SAT is hard to approximate with a factor of  $\rho$  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  $\ell_1$  for  $x$  there is at most one label  $\ell_2$  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**:

- ▶  $\text{MAX3SAT} \leq \text{MAX3SAT}(\leq 29)$
- ▶  $\text{MAX3SAT}(\leq 29) \leq \text{MAX3SAT}(\leq 5)$
- ▶  $\text{MAX3SAT}(\leq 5) \leq \text{MAX3SAT}(= 5)$
- ▶  $\text{MAX3SAT}(= 5) \leq \text{MAXE3SAT}(= 5)$

Here  $\text{MAX3SAT}(\leq 29)$  is the variant of  $\text{MAX3SAT}$  in which a variable appears in at most 29 clauses. Similar for the other problems.

## Theorem 108

*There is a constant  $\alpha < 1$  such if there is an  $\alpha$ -approximation algorithm for Label Cover on 15-regular instances than  $P=NP$ .*

Given a label  $\ell_1$  for  $x \in V_1$  there is at most one label  $\ell_2$  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  $I'$ :

- ▶  $V'_1 = V_1^k = V_1 \times \dots \times V_1$
- ▶  $V'_2 = V_2^k = V_2 \times \dots \times V_2$
- ▶  $L'_1 = L_1^k = L_1 \times \dots \times L_1$
- ▶  $L'_2 = L_2^k = L_2 \times \dots \times L_2$
- ▶  $E' = E^k = E \times \dots \times E$

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  $\sigma$  that satisfies just an  $\epsilon$ -fraction of edges in  $I$ .

We transfer this labelling to instance  $I'$ .

Each edge in  $I'$  gets label  $\sigma$ .

Each edge in  $I'$  gets label  $\sigma$ .

Each edge in  $I'$  gets label  $\sigma$ .

# 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  $\sigma$  that satisfies just an  $\epsilon$ -fraction of edges in  $I$ .

We transfer this labelling to instance  $I'$ .

What fraction of edges is satisfied?

What is the gap of  $I'$ ?

# 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  $\ell_1, \ell_2$  that satisfies just an  $\alpha$ -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  $\alpha$  fraction of edges are happy (just  $\alpha$  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  $\ell_1, \ell_2$  that satisfies just an  $\alpha$ -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?

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  $\ell_1, \ell_2$  that satisfies just an  $\alpha$ -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  $\alpha^k$  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  $\ell_1, \ell_2$  that satisfies just an  $\alpha$ -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  $\alpha^k$  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  $\ell_1, \ell_2$  that satisfies just an  $\alpha$ -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  $\alpha^k$  fraction)

**Does this always work?**



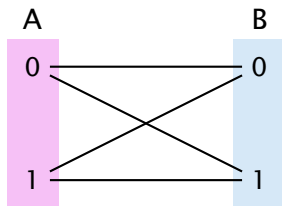
# Counter Example

## 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

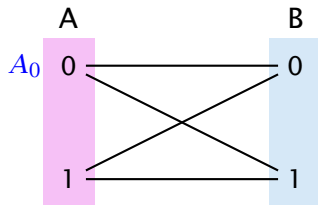
The provers can win with probability at most  $1/2$ .



Regardless what we do 50% of edges are unhappy!

# Counter Example

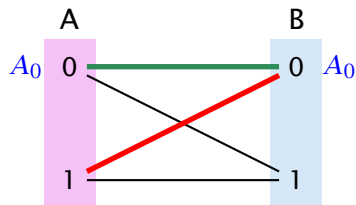
The provers can win with probability at most  $1/2$ .



Regardless what we do 50% of edges are unhappy!

# Counter Example

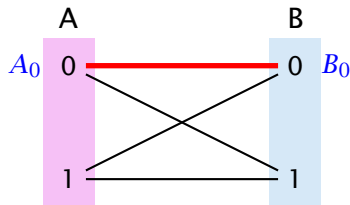
The provers can win with probability at most  $1/2$ .



Regardless what we do 50% of edges are unhappy!

# Counter Example

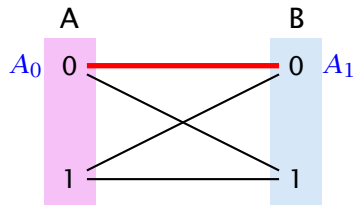
The provers can win with probability at most  $1/2$ .



Regardless what we do 50% of edges are unhappy!

# Counter Example

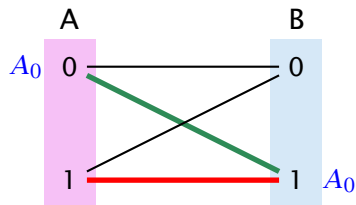
The provers can win with probability at most  $1/2$ .



Regardless what we do 50% of edges are unhappy!

# Counter Example

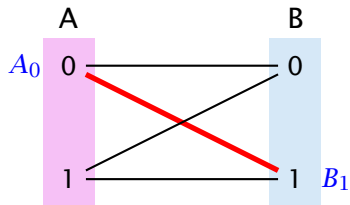
The provers can win with probability at most  $1/2$ .



Regardless what we do 50% of edges are unhappy!

# Counter Example

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

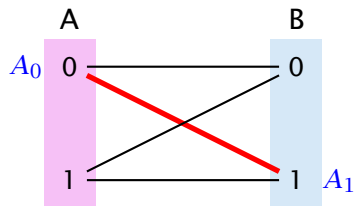


Regardless what we do 50% of edges are unhappy!



# Counter Example

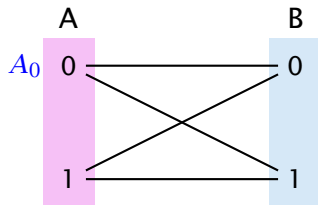
The provers can win with probability at most  $1/2$ .



Regardless what we do 50% of edges are unhappy!

# Counter Example

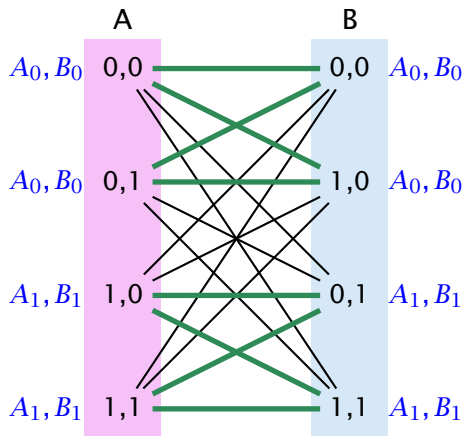
The provers can win with probability at most  $1/2$ .



Regardless what we do 50% of edges are unhappy!

## Counter Example

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



## Theorem 109

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

proof is highly non-trivial

## Theorem 109

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

proof is highly non-trivial

# Hardness of Label Cover

## Theorem 110

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

- ▶  $\text{OPT}(I) = |E|$ , or
- ▶  $\text{OPT}(I) = |E|(1 - \delta)^{ck}$

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

## Corollary 111

There is no  $\alpha$ -approximation for Label Cover for *any* constant  $\alpha$ .