Jean Goubault-Larrecq

Randomized complexity classes

Today: approximation problems, PCP

Today

- Approximation problems
- The class **PCP**
- **MAX3SAT** is not ε-approximable iff **NP=PCP**
- The Arora-Safra theorem: **NP=PCP** (no proof…)

Approximation problems

Approximation

- Attempt to attack **NP**-complete problems, by relaxing requirements. E.g., **3SAT** is **NP**-complete. Instead, given $\varepsilon \in]0,1[$, let **MAX3SAT** be:
- **INPUT**: a finite set S of 3-clauses
  OUTPUT: an environment ϱ that satisfies $(1-\varepsilon)\mathrm{opt}(S)$
  where $\mathrm{opt}(S) \stackrel{\text{def}}{=} \max_{\varrho \text{ env.}} (\#\text{clauses of } S \text{ s.t. } \varrho \models S)$
- For which values of ε is that in **P**?

Maximization problems

- For each input x, [e.g., a set of 3-clauses]
  a finite set $F(x)$ of so-called
  feasible solutions [e.g., all assignments ϱ on the vars of S}]

Maximization problems

- For each input x, [e.g., a set of 3-clauses]
  a finite set $F(x)$ of so-called
  feasible solutions [e.g., all assignments ϱ on the vars of S}]
- For each $y \in F(x)$,
  a **value** $c(y)$ [e.g., #clauses satisfied by ϱ]

Maximization problems

- For each input x, [e.g., a set of 3-clauses]
  a finite set $F(x)$ of so-called
  feasible solutions [e.g., all assignments ϱ on the vars of S}]
- For each $y \in F(x)$,
  a **value** $c(y)$ [e.g., #clauses satisfied by ϱ]
- Goal: estimate $\mathrm{opt}(x) \stackrel{\mathrm{def}}{=} \max_{y \in F(x)} c(y)$

Maximization problems

- For each input x, [e.g., a set of 3-clauses]
  a finite set $F(x)$ of so-called
  feasible solutions [e.g., all assignments ϱ on the vars of S}]
- For each $y \in F(x)$,
  a **value** $c(y)$ [e.g., #clauses satisfied by ϱ]
- Goal: estimate $\mathrm{opt}(x) \stackrel{\text{def}}{=} \max_{y \in F(x)} c(y)$
- **ε-approximable** iff can find $y \in F(x)$ / $c(y) \geq (1-\varepsilon)\mathrm{opt}(x)$ in polynomial time

Maximization problems

- For each input x, [e.g., a set of 3-clauses]
  a finite set $F(x)$ of so-called
  feasible solutions [e.g., all assignments ϱ on the vars of S}]
- For each $y \in F(x)$,
  a **value** $c(y)$ [e.g., #clauses satisfied by ϱ]
- Goal: estimate $\mathrm{opt}(x) \stackrel{\mathrm{def}}{=} \max_{y \in F(x)} c(y)$
- **ε-approximable** iff can find $y \in F(x)$ / $c(y) \geq (1-\varepsilon)\mathrm{opt}(x)$ in polynomial time
- **Defn.** The **approximation threshold** $= \inf_{\varepsilon\text{-approximable}} \varepsilon$

Minimization problems

- For each input x,
  a finite set $F(x)$ of so-called
  feasible solutions
- For each $y \in F(x)$,
  a **cost** $c(y)$
- Goal: estimate $\mathrm{opt}(x) \stackrel{\mathrm{def}}{=} \min_{y \in F(x)} c(y)$
- **ε-approximable** iff can find $y \in F(x)$ / $c(y) \leq 1/(1-\varepsilon).\mathrm{opt}(x)$ in polynomial time
- **Defn.** The **approximation threshold** $= \inf_{\varepsilon\text{-approximable}} \varepsilon$

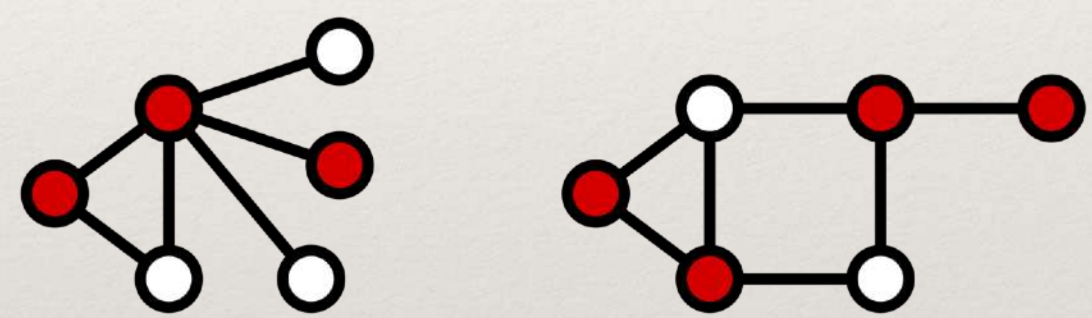
Optimization problems

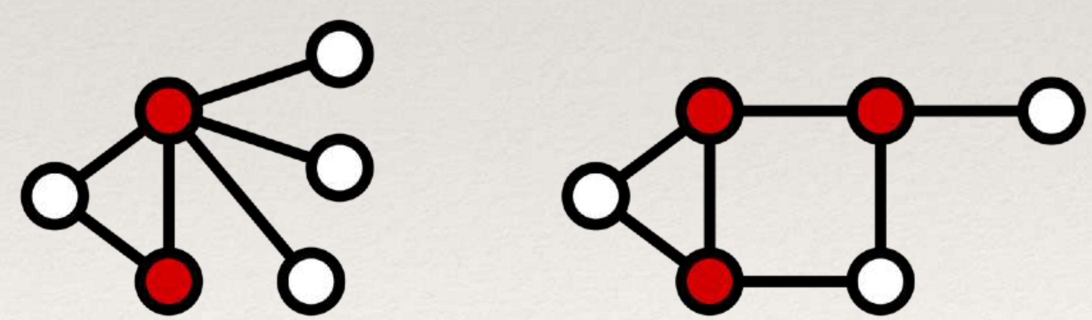
- Optimization = maximization or minimization
- **ε-approximable** iff can find $y \in F(x)$ /
  $$|c(y)-\mathrm{opt}(x)| / \max(c(y),\mathrm{opt}(x)) \leq \varepsilon$$
  in polynomial time (ugly formula, but generalizes the previous formulae)
- **Defn.** The **approximation threshold** = $\inf_{\varepsilon\text{-approximable}} \varepsilon$
- Let us see, through a few examples, that this can be pretty much any number in [0,1].

NODE COVER

NODE COVER

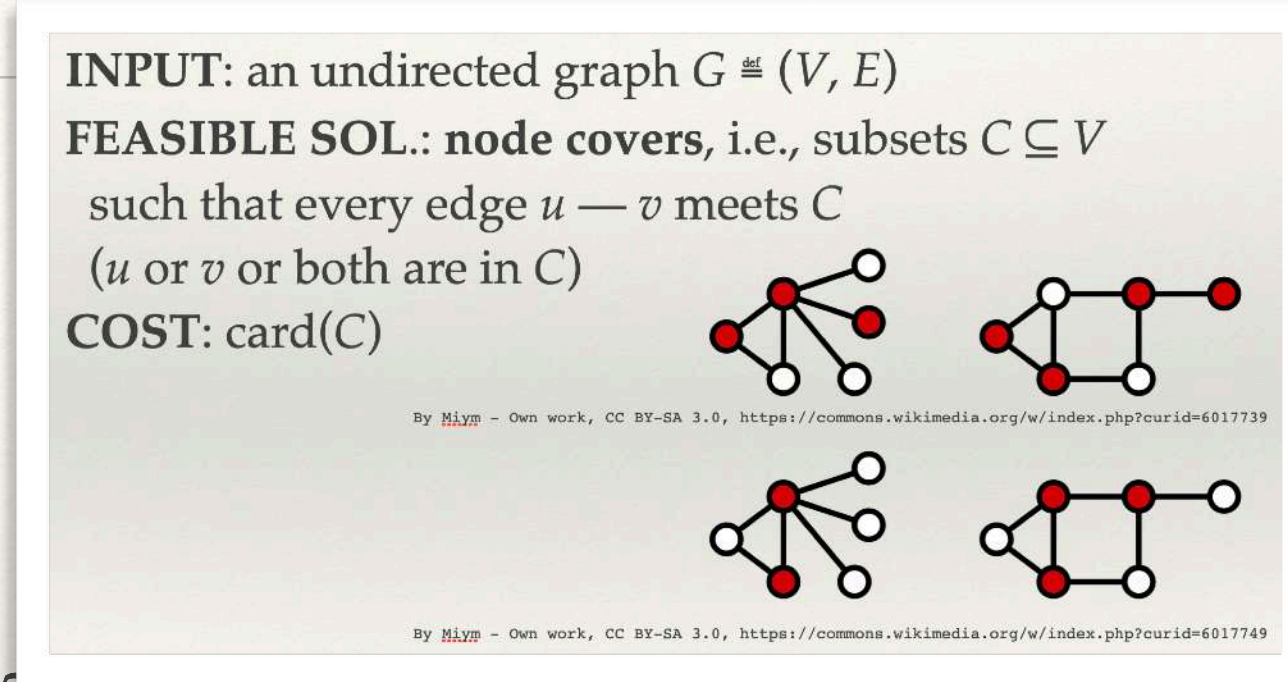
- **INPUT**: an undirected graph $G \stackrel{\text{def}}{=} (V, E)$
  FEASIBLE SOL.: **node covers**, i.e., subsets $C \subseteq V$
  such that every edge $u — v$ meets C
  (u or v or both are in C)
  COST: card(C)





NODE COVER

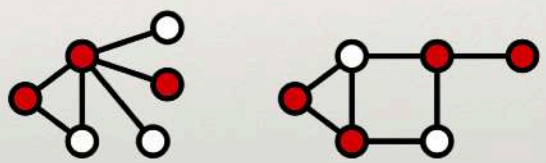
- The associated **decision problem**:
  INPUT: G, a budget k
  QUESTION: does G have a node cover C with $\mathrm{card}(C) \leq k$?
- is **NP**-complete.



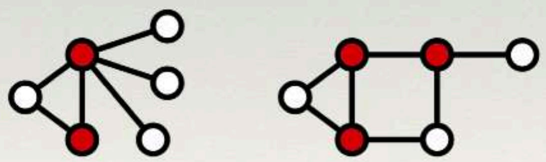
INPUT: an undirected graph $G \stackrel{\text{def}}{=} (V, E)$
FEASIBLE SOL.: **node covers**, i.e., subsets $C \subseteq V$ such that every edge $u - v$ meets C (u or v or both are in C)
COST: $\mathrm{card}(C)$

NODE COVER

INPUT: an undirected graph $G \stackrel{\text{def}}{=} (V, E)$
FEASIBLE SOL.: **node covers**, i.e., subsets $C \subseteq V$ such that every edge $u - v$ meets C (u or v or both are in C)
COST: card(C)



By Miym - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=6017739

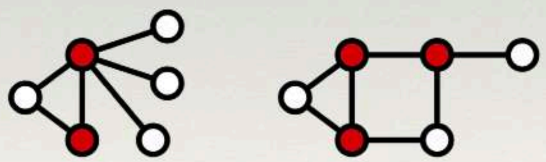


By Miym - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=6017749

- The associated **decision problem**:
  INPUT: G, a budget k
  QUESTION: does G have a node cover C with card(C) $\leq k$?
- is **NP**-complete.
- What is the approximation threshold of **NODE COVER**?

NODE COVER

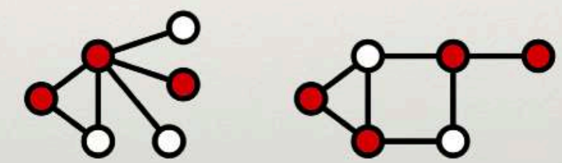
INPUT: an undirected graph $G \stackrel{\text{def}}{=} (V, E)$
FEASIBLE SOL.: **node covers**, i.e., subsets $C \subseteq V$
such that every edge u — v meets C
(u or v or both are in C)
COST: card(C)



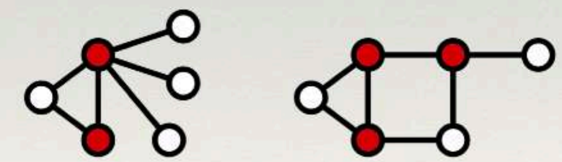
- The associated **decision problem**:
  INPUT: G, a budget k
  QUESTION: does G have a node cover C with card(C) $\leq k$?
- is **NP**-complete.
- What is the approximation threshold of **NODE COVER**?
- Hint: the best known approximation algorithm is also one of the dumbest… and no, picking a vertex to be put in the cover, removing all incident edges, and going on is not dumb enough

NODE COVER is ½-approximable

INPUT: an undirected graph $G \stackrel{\text{def}}{=} (V, E)$
FEASIBLE SOL.: **node covers**, i.e., subsets $C \subseteq V$ such that every edge u — v meets C (u or v or both are in C)
COST: card(C)



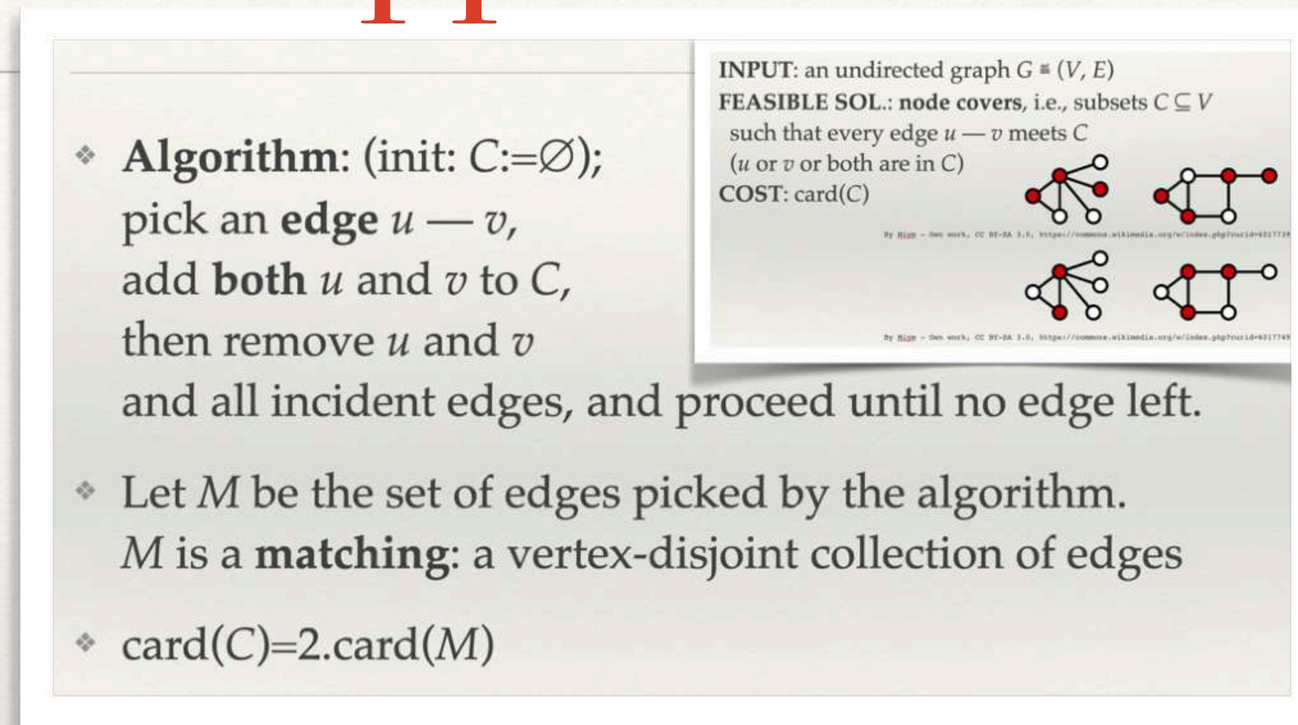
By Miym - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=6017739



By Miym - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=6017749

- **Algorithm**: (init: C:=∅);
  pick an **edge** u — v,
  add **both** u and v to C,
  then remove u and v
  and all incident edges, and proceed until no edge left.
- Let M be the set of edges picked by the algorithm.
  M is a **matching**: a vertex-disjoint collection of edges
- card(C)=2.card(M)

NODE COVER is ½-approximable



- Algorithm: (init: C:=∅); pick an **edge** u — v, add **both** u and v to C, then remove u and v and all incident edges, and proceed until no edge left.
- Let M be the set of edges picked by the algorithm. M is a **matching**: a vertex-disjoint collection of edges
- card(C)=2.card(M)

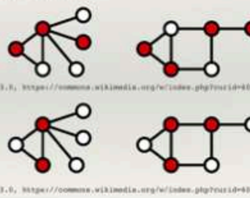
INPUT: an undirected graph $G = (V, E)$
FEASIBLE SOL.: **node covers**, i.e., subsets $C \subseteq V$ such that every edge u — v meets C (u or v or both are in C)
COST: card(C)

- Given a node cover C', every edge of M meets C' at a **distinct** vertex
- So card(M) ≤ card(C')
- Since card(C) = 2.card(M), card(C) ≤ 2. card(C')

NODE COVER is ½-approximable

- Given a node cover C', every edge of M meets C' at a **distinct** vertex
- So $\mathrm{card}(M) \leq \mathrm{card}(C')$
- Since $\mathrm{card}(C) = 2.\mathrm{card}(M)$, $\mathrm{card}(C) \leq 2.\ \mathrm{card}(C')$
- Hence **NODE COVER** is ½-approximable. (½ is in fact the best we can do, unless **P=NP**)

INPUT: an undirected graph $G = (V, E)$
FEASIBLE SOL.: **node covers**, i.e., subsets $C \subseteq V$ such that every edge $u — v$ meets C (u or v or both are in C)
COST: card(C)



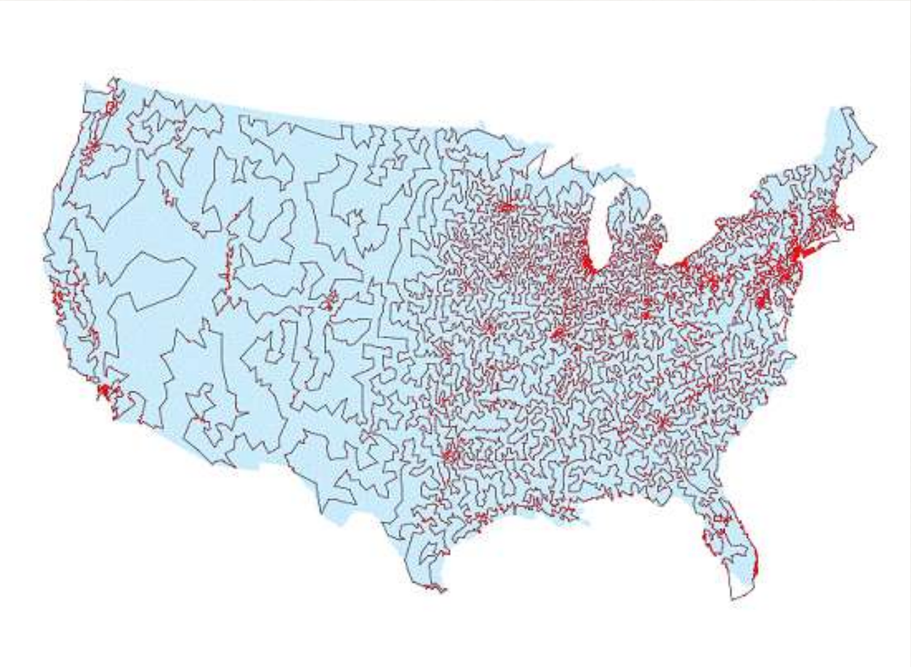
- **Algorithm:** (init: $C{:=}\varnothing$); pick an **edge** $u — v$, add **both** u and v to C, then remove u and v and all incident edges, and proceed until no edge left.
- Let M be the set of edges picked by the algorithm. M is a **matching**: a vertex-disjoint collection of edges
- card(C)=2.card(M)

- **ε-approximable** iff can find $y \in F(x)$ / $c(y) \geq (1-\varepsilon)\mathrm{opt}(x)$ in polynomial time
- **Defn.** The **approximation threshold** = $\inf_{\varepsilon\text{-approximable}}\ \varepsilon$

The traveling salesman problem (TSP)

TSP

- **INPUT**: a matrix $D \stackrel{\text{def}}{=} (d_{ij})_{1 \le i,j \le n}$ of 'distances' between cities
  (only constraint: d_{ii}=0)
  FEASIBLE SOL.: **tours**, i.e., permutations π of $\{1,\ldots,n\}$
  COST: $d_{\pi(1)\pi(2)}+d_{\pi(2)\pi(3)}+\ldots+d_{\pi(n-1)\pi(n)}+d_{\pi(n)\pi(1)}$

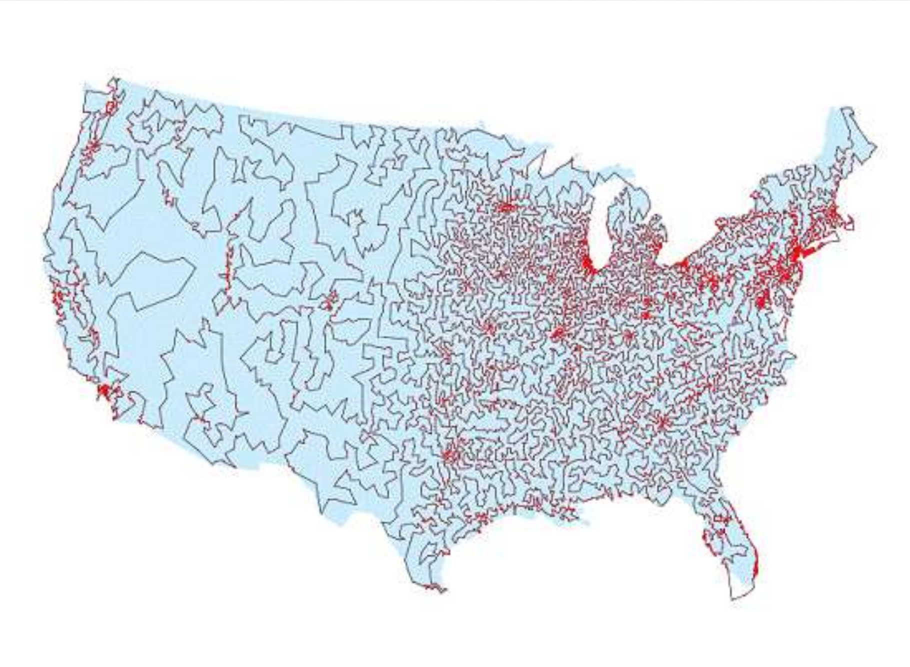


13,509 U.S. cities with populations of more than 500 people connected optimally

`http://www.crpc.rice.edu/CRPC/newsletters/sum98/news_tsp.html`

TSP

- **INPUT**: a matrix $D \stackrel{\text{def}}{=} (d_{ij})_{1 \le i,j \le n}$ of 'distances' between cities
  (only constraint: $d_{ii}=0$)
  FEASIBLE SOL.: **tours**, i.e., permutations π of $\{1,\ldots,n\}$
  COST: $d_{\pi(1)\pi(2)}+d_{\pi(2)\pi(3)}+\ldots+d_{\pi(n-1)\pi(n)}+d_{\pi(n)\pi(1)}$
- Decision problem (is cost ≤ some given budget?) is **NP**-complete

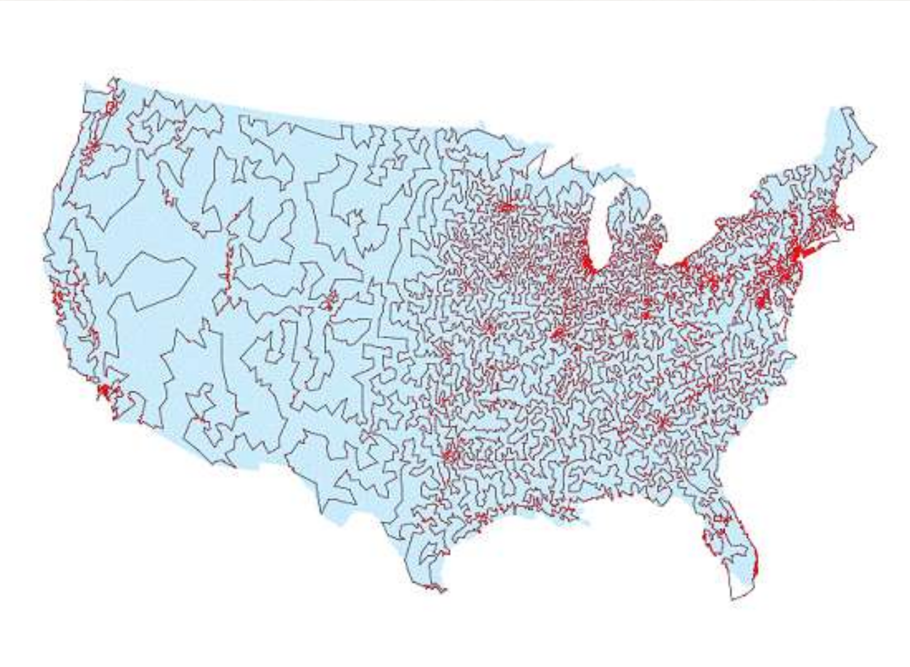


13,509 U.S. cities with populations of more than 500 people connected optimally

`http://www.crpc.rice.edu/CRPC/newsletters/sum98/news_tsp.html`

TSP

- **INPUT**: a matrix $D \stackrel{\text{def}}{=} (d_{ij})_{1 \leq i,j \leq n}$ of 'distances' between cities
  (only constraint: $d_{ii}=0$)
  FEASIBLE SOL.: **tours**, i.e., permutations π of $\{1,\ldots,n\}$
  COST: $d_{\pi(1)\pi(2)} + d_{\pi(2)\pi(3)} + \ldots + d_{\pi(n-1)\pi(n)} + d_{\pi(n)\pi(1)}$
- Decision problem (is cost $\leq$ some given budget?) is **NP**-complete
- ε-approximable for **no** $\varepsilon \in]0,1[$ unless **P**=**NP**. Hence approximation threshold is 1 (worst possible!)



13,509 U.S. cities with populations of more than 500 people connected optimally

http://www.crpc.rice.edu/CRPC/newsletters/sum98/news_tsp.html

TSP is not approximable

- We use the fact that **HAMILTONIAN CYCLE**:
  INPUT: an undirected graph $G \stackrel{\text{def}}{=} (V, E)$
  QUESTION: $\exists$cycle in G going once through each vertex?
  is **NP**-complete

TSP is not approximable

- We use the fact that **HAMILTONIAN CYCLE**:
  INPUT: an undirected graph $G \stackrel{\text{def}}{=} (V, E)$
  QUESTION: $\exists$cycle in G going once through each vertex?
  is **NP**-complete
- We build a poly time reduction from **HAMILTONIAN CYCLE** to (the decision form) of **TSP**,

TSP is not approximable

- We use the fact that **HAMILTONIAN CYCLE**:
  INPUT: an undirected graph $G \stackrel{\text{def}}{=} (V, E)$
  QUESTION: $\exists$cycle in G going once through each vertex?
  is **NP**-complete
- We build a poly time reduction from **HAMILTONIAN CYCLE** to (the decision form) of **TSP**,
- showing that if **TSP** is ε-approximable, then **HAMILTONIAN CYCLE** is in **P**, hence **P**=**NP**.

TSP is not approximable

- Given G [$N \stackrel{\text{def}}{=} \text{card}(V)$] and
  $M > 1/(1{-}\varepsilon).N$,
  let $d_{ij} \stackrel{\text{def}}{=} 1$ if edge i — j,
  M if no edge. Defines an instance D of **TSP**.

HAMILTONIAN CYCLE:
INPUT: an undirected graph $G \stackrel{\text{def}}{=} (V, E)$
QUESTION: ∃cycle in G going once through each vertex?
is **NP**-complete

TSP is not approximable

- Given G [$N \stackrel{\text{def}}{=} \text{card}(V)$] and
  $M > 1/(1-\varepsilon).N$,
  let $d_{ij} \stackrel{\text{def}}{=} 1$ if edge $i — j$,
  M if no edge. Defines an instance D of **TSP**.

HAMILTONIAN CYCLE:
INPUT: an undirected graph $G \stackrel{\text{def}}{=} (V, E)$
QUESTION: ∃cycle in G going once through each vertex?
is **NP**-complete

- Tour π: cost = N if Hamiltonian cycle, ≥ M otherwise

TSP is not approximable

- Given G [$N \stackrel{\text{def}}{=} \text{card}(V)$] and
  $M > 1/(1\text{–}\varepsilon).N$,
  let $d_{ij} \stackrel{\text{def}}{=} 1$ if edge i — j,
  M if no edge. Defines an instance D of **TSP**.

HAMILTONIAN CYCLE:
INPUT: an undirected graph $G \stackrel{\text{def}}{=} (V, E)$
QUESTION: ∃cycle in G going once through each vertex?
is **NP**-complete

- Tour π: cost = N if Hamiltonian cycle, $\geq M$ otherwise
- Assume an ε-approximation (poly time) algorithm A for TSP

TSP is not approximable

HAMILTONIAN CYCLE:
INPUT: an undirected graph $G \stackrel{\text{def}}{=} (V, E)$
QUESTION: $\exists$cycle in G going once through each vertex?
is **NP**-complete

- Given G [$N \stackrel{\text{def}}{=} \text{card}(V)$] and
  $M > 1/(1-\varepsilon).N$,
  let $d_{ij} \stackrel{\text{def}}{=} 1$ if edge $i — j$,
  M if no edge. Defines an instance D of **TSP**.
- Tour π: cost $= N$ if Hamiltonian cycle, $\geq M$ otherwise
- Assume an ε-approximation (poly time) algorithm A for TSP
- If G has a Hamiltonian cycle, opt(D) = N
  A(D) will find a tour of cost $\leq 1/(1-\varepsilon)$.opt(D) $< M$,
  hence a Hamiltonian cycle, in poly time

TSP is not approximable

> **HAMILTONIAN CYCLE**:
> **INPUT**: an undirected graph $G \stackrel{\text{def}}{=} (V, E)$
> **QUESTION**: $\exists$cycle in G going once through each vertex?
> is **NP**-complete

- Given G [$N \stackrel{\text{def}}{=} \text{card}(V)$] and
  $M > 1/(1-\varepsilon).N$,
  let $d_{ij} \stackrel{\text{def}}{=} 1$ if edge $i — j$,
  M if no edge. Defines an instance D of **TSP**.
- Tour π: cost $= N$ if Hamiltonian cycle, $\geq M$ otherwise
- Assume an ε-approximation (poly time) algorithm A for TSP
- If G has a Hamiltonian cycle, opt$(D) = N$
  A(D) will find a tour of cost $\leq 1/(1-\varepsilon)$.opt$(D) < M$,
  hence a Hamiltonian cycle, in poly time
- Hence **HAMILTONIAN CYCLE** is in **P**, so **P=NP**.

KNAPSACK

KNAPSACK

- **INPUT**: prices v_i and weights w_i, $1 \leq i \leq n$, a max weight W
  (all are natural numbers)
  FEASIBLE SOL.: a subset $S \subseteq \{1,\ldots,n\}$ / $\Sigma_{i \in S}\, w_i \leq W$
  COST: $\Sigma_{i \in S}\, v_i$

KNAPSACK

- **INPUT**: prices v_i and weights w_i, $1 \le i \le n$, a max weight W
  (all are natural numbers)
  FEASIBLE SOL.: a subset $S \subseteq \{1,\ldots,n\}$ / $\Sigma_{i \in S}\, w_i \le W$
  COST: $\Sigma_{i \in S}\, v_i$
- Decision problem (is cost ≤ some given budget?) is **NP**-complete

KNAPSACK

- **INPUT**: prices v_i and weights w_i, $1 \leq i \leq n$, a max weight W
  (all are natural numbers)
  FEASIBLE SOL.: a subset $S \subseteq \{1,\ldots,n\}$ / $\Sigma_{i \in S}\, w_i \leq W$
  COST: $\Sigma_{i \in S}\, v_i$
- Decision problem (is cost ≤ some given budget?) is **NP**-complete
- ε-approximable for **every** $\varepsilon \in \,]0,1[$.
  Approximation threshold is 0 (best possible!)

KNAPSACK

INPUT: prices v_i and weights w_i, $1 \leq i \leq n$, a max weight W
FEASIBLE SOL.: a subset $S \subseteq \{1,\ldots,n\}$ / $\Sigma_{i \in S}\, w_i \leq W$
COST: $\Sigma_{i \in S}\, v_i$

- A well-known **dynamic programming** algorithm for KNAPSACK:
- Let $V \stackrel{\text{def}}{=} \Sigma_{i=1}{}^{n}\, v_i$, and, for all $1 \leq j \leq n$ and $0 \leq v \leq V$:

$$W(j,v) = \min \{\Sigma_{i \in S}\, w_i \mid S \subseteq \{1,\ldots,j\},\ \Sigma_{i \in S}\, v_i = v\}$$

KNAPSACK

INPUT: prices v_i and weights w_i, $1 \le i \le n$, a max weight W
FEASIBLE SOL.: a subset $S \subseteq \{1,\ldots,n\}$ / $\Sigma_{i \in S}\, w_i \le W$
COST: $\Sigma_{i \in S}\, v_i$

- A well-known **dynamic programming** algorithm for KNAPSACK:
- Let $V \stackrel{\text{def}}{=} \Sigma_{i=1}{}^{n}\, v_i$, and, for all $1 \le j \le n$ and $0 \le v \le V$:

$$W(j,v) = \min \{\Sigma_{i \in S}\, w_i \mid S \subseteq \{1,\ldots,j\},\ \Sigma_{i \in S}\, v_i = v\}$$

- Then $W(j,v) = \min(W(j-1,v), W(j-1,v-v_j)+w_i)$ if $v \ge v_j$
  $W(j-1,v)$ otherwise

  can be computed in time $O(nV)$

KNAPSACK

INPUT: prices v_i and weights w_i, $1 \leq i \leq n$, a max weight W
FEASIBLE SOL.: a subset $S \subseteq \{1,\ldots,n\}$ / $\Sigma_{i \in S} w_i \leq W$
COST: $\Sigma_{i \in S} v_i$

- A well-known **dynamic programming** algorithm for KNAPSACK:
- Let $V \stackrel{\text{def}}{=} \Sigma_{i=1}^{n} v_i$, and, for all $1 \leq j \leq n$ and $0 \leq v \leq V$:
$$W(j,v) = \min \{\Sigma_{i \in S} w_i \mid S \subseteq \{1,\ldots,j\}, \Sigma_{i \in S} v_i = v\}$$
- Then $W(j,v) = \min(W(j-1,v), W(j-1,v-v_j)+w_i)$ if $v \geq v_j$
$W(j-1,v)$ otherwise
can be computed in time $O(nV)$
- Finally, find largest v such that $W(n,v) \leq W$.

KNAPSACK

INPUT: prices v_i and weights w_i, $1 \leq i \leq n$, a max weight W
FEASIBLE SOL.: a subset $S \subseteq \{1,\ldots,n\}$ / $\Sigma_{i\in S}\, w_i \leq W$
COST: $\Sigma_{i\in S}\, v_i$

- A well-known **dynamic programming** algorithm for KNAPSACK:
- Let $V \stackrel{\text{def}}{=} \Sigma_{i=1}{}^{n}\, v_i$, and, for all $1 \leq j \leq n$ and $0 \leq v \leq V$:

$$W(j,v) = \min\,\{\Sigma_{i\in S}\, w_i \mid S \subseteq \{1,\ldots,j\},\ \Sigma_{i\in S}\, v_i = v\}$$

- Then $W(j,v) = \min(W(j-1,v),\, W(j-1,v-v_j)+w_i)$ if $v \geq v_j$
  $W(j-1,v)$ otherwise
  can be computed in time $O(nV)$

exponential in size(V)=O(log V)
if numbers in binary

- Finally, find largest v such that $W(n,v) \leq W$.

KNAPSACK

INPUT: prices v_i and weights w_i, $1 \le i \le n$, a max weight W
FEASIBLE SOL.: a subset $S \subseteq \{1,\ldots,n\}$ / $\Sigma_{i \in S}\, w_i \le W$
COST: $\Sigma_{i \in S}\, v_i$

- A well-known **dynamic programming** algorithm for KNAPSACK:
- Let $V \stackrel{\text{def}}{=} \Sigma_{i=1}^{n} v_i$, and, for all $1 \le j \le n$ and $0 \le v \le V$:
  $W(j,v) = \min \{\Sigma_{i \in S}\, w_i \mid S \subseteq \{1,\ldots,j\},\ \Sigma_{i \in S}\, v_i = v\}$
- Then $W(j,v) = \min(W(j-1,v),\ W(j-1,v-v_j)+w_i)$ if $v \ge v_j$
  $W(j-1,v)$ otherwise
  can be computed in time $O(nV)$
- Finally, find largest v such that $W(n,v) \le W$.

KNAPSACK

- Do all computations on values (i.e., v, v_i) by only keeping the **k most significant bits** of each number and **rounding down**

INPUT: prices v_i and weights w_i, $1 \le i \le n$, a max weight W
FEASIBLE SOL.: a subset $S \subseteq \{1,\dots,n\}$ / $\Sigma_{i \in S} w_i \le W$
COST: $\Sigma_{i \in S} v_i$

- A well-known **dynamic programming** algorithm for KNAPSACK:
- Let $V \stackrel{\text{def}}{=} \Sigma_{i=1}^{n} v_i$, and, for all $1 \le j \le n$ and $0 \le v \le V$:
  $$W(j,v) = \min \{\Sigma_{i \in S} w_i \mid S \subseteq \{1,\dots,j\}, \Sigma_{i \in S} v_i = v\}$$
- Then $W(j,v) = \min(W(j-1,v), W(j-1,v-v_j)+w_i)$ if $v \ge v_j$
  $W(j-1,v)$ otherwise
  can be computed in time $O(nV)$
- Finally, find largest v such that $W(n,v) \le W$.

KNAPSACK

INPUT: prices v_i and weights w_i, $1 \leq i \leq n$, a max weight W
FEASIBLE SOL.: a subset $S \subseteq \{1,\ldots,n\}$ / $\Sigma_{i \in S}\, w_i \leq W$
COST: $\Sigma_{i \in S}\, v_i$

- A well-known **dynamic programming** algorithm for KNAPSACK:
- Let $V \stackrel{\text{def}}{=} \Sigma_{i=1}^{n} v_i$, and, for all $1 \leq j \leq n$ and $0 \leq v \leq V$:
  $$W(j,v) = \min \{\Sigma_{i \in S}\, w_i \mid S \subseteq \{1,\ldots,j\},\ \Sigma_{i \in S}\, v_i = v\}$$
- Then $W(j,v) = \min(W(j-1,v), W(j-1,v-v_j)+w_i)$ if $v \geq v_j$
  $W(j-1,v)$ otherwise
  can be computed in time $O(nV)$
- Finally, find largest v such that $W(n,v) \leq W$.

- Do all computations on values (i.e., v, v_i) by only keeping the
  k most significant bits
  of each number and
  rounding down
- I.e., represent v_i by the k-bit number $\lfloor v_i / 2^{k0-k} \rfloor$ [$k0 \stackrel{\text{def}}{=}$ #bits in nV]

KNAPSACK

- Do all computations on values (i.e., v, v_i) by only keeping the
  k most significant bits
  of each number and
  rounding down

INPUT: prices v_i and weights w_i, $1 \le i \le n$, a max weight W
FEASIBLE SOL.: a subset $S \subseteq \{1,\ldots,n\}$ / $\Sigma_{i \in S}\, w_i \le W$
COST: $\Sigma_{i \in S}\, v_i$

- A well-known **dynamic programming** algorithm for KNAPSACK:
- Let $V \stackrel{\text{def}}{=} \Sigma_{i=1}^{n}\, v_i$, and, for all $1 \le j \le n$ and $0 \le v \le V$:
  $$W(j,v) = \min \{\Sigma_{i \in S}\, w_i \mid S \subseteq \{1,\ldots,j\},\ \Sigma_{i \in S}\, v_i = v\}$$
- Then $W(j,v) = \min(W(j-1,v),\ W(j-1,v-v_j)+w_i)$ if $v \ge v_j$
  $W(j-1,v)$ otherwise
  can be computed in time $O(nV)$
- Finally, find largest v such that $W(n,v) \le W$.

- I.e., represent v_i by the k-bit number $\lfloor v_i / 2^{k0-k} \rfloor$ [$k0 \stackrel{\text{def}}{=}$ #bits in nV]
- replace all values v by k-bit approximations v' ($v \approx 2^{k0-k}\, v'$)

KNAPSACK

- A well-known **dynamic programming** algorithm for KNAPSACK:

INPUT: prices v_i and weights w_i, $1 \leq i \leq n$, a max weight W
FEASIBLE SOL.: a subset $S \subseteq \{1,\ldots,n\}$ / $\Sigma_{i \in S}\, w_i \leq W$
COST: $\Sigma_{i \in S}\, v_i$

- Let $V \stackrel{\text{def}}{=} \Sigma_{i=1}^{n}\, v_i$, and, for all $1 \leq j \leq n$ and $0 \leq v \leq V$:
  $W(j,v) = \min\, \{\Sigma_{i \in S}\, w_i \mid S \subseteq \{1,\ldots,j\},\, \Sigma_{i \in S}\, v_i = v\}$
- Then $W(j,v) = \min(W(j-1,v), W(j-1,v-v_j)+w_i)$ if $v \geq v_j$
  $W(j-1,v)$ otherwise
  can be computed in time $O(nV)$
- Finally, find largest v such that $W(n,v) \leq W$.

- Do all computations on values (i.e., v, v_i) by only keeping the
  ***k* most significant bits**
  of each number and
  rounding down
- I.e., represent v_i by the k-bit number $\lfloor v_i / 2^{k0-k} \rfloor$ [$k0 \stackrel{\text{def}}{=}$ #bits in nV]
- replace all values v by k-bit approximations v' ($v \approx 2^{k0-k}\, v'$)
- replace computation of $v-v_i$ by $v' - \lfloor v_i / 2^{k0-k} \rfloor$

KNAPSACK

INPUT: prices v_i and weights w_i, $1 \le i \le n$, a max weight W
FEASIBLE SOL.: a subset $S \subseteq \{1,\dots,n\}$ / $\Sigma_{i \in S}\, w_i \le W$
COST: $\Sigma_{i \in S}\, v_i$

- A well-known **dynamic programming** algorithm for KNAPSACK:
- Let $V \stackrel{\text{def}}{=} \Sigma_{i=1}{}^{n}\, v_i$, and, for all $1 \le j \le n$ and $0 \le v \le V$:
  $$W(j,v) = \min \{\Sigma_{i \in S}\, w_i \mid S \subseteq \{1,\dots,j\},\ \Sigma_{i \in S}\, v_i = v\}$$
- Then $W(j,v) = \min(W(j-1,v),\ W(j-1,v-v_j)+w_i)$ if $v \ge v_j$
  $W(j-1,v)$ otherwise
  can be computed in time $O(nV)$
- Finally, find largest v such that $W(n,v) \le W$.

KNAPSACK

- We choose
  $k \stackrel{\text{def}}{=} \lceil \text{size}(nV) - \log_2(\varepsilon V / n) \rceil$
  $= \log_2 (n^2 / \varepsilon) + O(1)$

INPUT: prices v_i and weights w_i, $1 \leq i \leq n$, a max weight W
FEASIBLE SOL.: a subset $S \subseteq \{1,\ldots,n\}$ / $\Sigma_{i \in S}\, w_i \leq W$
COST: $\Sigma_{i \in S}\, v_i$

- A well-known **dynamic programming** algorithm for KNAPSACK:
- Let $V \stackrel{\text{def}}{=} \Sigma_{i=1}^{n}\, v_i$, and, for all $1 \leq j \leq n$ and $0 \leq v \leq V$:
  $W(j,v) = \min \{\Sigma_{i \in S}\, w_i \mid S \subseteq \{1,\ldots,j\}, \Sigma_{i \in S}\, v_i = v\}$
- Then $W(j,v) = \min(W(j-1,v), W(j-1,v-v_j)+w_i)$ if $v \geq v_j$
  $W(j-1,v)$ otherwise
  can be computed in time $O(nV)$
- Finally, find largest v such that $W(n,v) \leq W$.

KNAPSACK

- We choose
  $k \stackrel{\text{def}}{=} \lceil \text{size}(nV) - \log_2(\varepsilon V / n) \rceil$
  $= \log_2 (n^2 / \varepsilon) + O(1)$

- There are now at most
  $2^k = O(n^2 / \varepsilon)$ different values
  instead of $O(V)$,
  hence times goes down to $O(n\, 2^k) = O(n^3 / \varepsilon)$

INPUT: prices v_i and weights w_i, $1 \leq i \leq n$, a max weight W
FEASIBLE SOL.: a subset $S \subseteq \{1,\ldots,n\}$ / $\Sigma_{i \in S}\, w_i \leq W$
COST: $\Sigma_{i \in S}\, v_i$

- A well-known **dynamic programming** algorithm for KNAPSACK:
- Let $V \stackrel{\text{def}}{=} \Sigma_{i=1}^{n}\, v_i$, and, for all $1 \leq j \leq n$ and $0 \leq v \leq V$:
  $W(j,v) = \min \{\Sigma_{i \in S}\, w_i \mid S \subseteq \{1,\ldots,j\}, \Sigma_{i \in S}\, v_i = v\}$
- Then $W(j,v) = \min(W(j-1,v), W(j-1,v-v_j)+w_i)$ if $v \geq v_j$
  $W(j-1,v)$ otherwise
  can be computed in time $O(nV)$
- Finally, find largest v such that $W(n,v) \leq W$.

KNAPSACK

- We choose
  $k \stackrel{\text{def}}{=} \lceil \mathrm{size}(nV) - \log_2(\varepsilon V / n) \rceil$
  $= \log_2 (n^2 / \varepsilon) + O(1)$
- There are now at most
  $2^k = O(n^2 / \varepsilon)$ different values
  instead of $O(V)$,
  hence times goes down to $O(n\, 2^k) = O(n^3 / \varepsilon)$
- And final value is between $(1-\varepsilon)$opt and opt
  (see lecture notes for details, Prop. 2.7).

INPUT: prices v_i and weights w_i, $1 \leq i \leq n$, a max weight W
FEASIBLE SOL.: a subset $S \subseteq \{1,\dots,n\}$ / $\Sigma_{i \in S}\, w_i \leq W$
COST: $\Sigma_{i \in S}\, v_i$

- A well-known **dynamic programming** algorithm for KNAPSACK:
- Let $V \stackrel{\text{def}}{=} \Sigma_{i=1}^{n} v_i$, and, for all $1 \leq j \leq n$ and $0 \leq v \leq V$:
  $W(j,v) = \min \{\Sigma_{i \in S}\, w_i \mid S \subseteq \{1,\dots,j\},\ \Sigma_{i \in S}\, v_i = v\}$
- Then $W(j,v) = \min(W(j-1,v),\ W(j-1,v-v_j)+w_i)$ if $v \geq v_j$
  $W(j-1,v)$ otherwise
  can be computed in time $O(nV)$
- Finally, find largest v such that $W(n,v) \leq W$.

MAX3SAT

MAXSAT

- **INPUT**: a finite list *S* of clauses
  FEASIBLE SOL.: an environment ϱ
  VALUE: #clauses satisfied by ϱ
- Decision problem (is value ≥ some given goal?)
  is **NP**-complete
- ε-approximable for **which** ε ∈]0,1[?
  Let me give you the best known (and silliest) algorithm…

Johnson's algorithm

- Rough idea: while there is a variable A left, decide to set A to 1 (true) or 0 (false) depending on which of

  E(#clauses of $S[A{:=}1]$ satisfied by ϱ) and

  E(#clauses of $S[A{:=}0]$ satisfied by ϱ) is larger,

  where ϱ is drawn at random.

Johnson's algorithm

- Rough idea: while there is a variable A left, decide to set A to 1 (true) or 0 (false) depending on which of

  E(#clauses of $S[A{:=}1]$ satisfied by ϱ) and

  E(#clauses of $S[A{:=}0]$ satisfied by ϱ) is larger,

  where ϱ is drawn at random.

- $S[A{:=}1]$: remove clauses where A occurs positively,

  remove $\neg A$ from remaining clauses

Johnson's algorithm

- Rough idea: while there is a variable A left, decide to set A to 1 (true) or 0 (false) depending on which of
  E(#clauses of $S[A:=1]$ satisfied by ϱ) and
  E(#clauses of $S[A:=0]$ satisfied by ϱ) is larger,
  where ϱ is drawn at random.
- $S[A:=1]$: remove clauses where A occurs positively,
  remove $\neg A$ from remaining clauses
- $S[A:=0]$: remove clauses where $\neg A$ occurs
  (i.e., A occurs negatively),
  remove A from remaining clauses

Johnson's algorithm

- In reality: we compare

  E(#clauses of $S[A{:=}1]$ **not** satisfied by ϱ) and

  E(#clauses of $S[A{:=}0]$ **not** satisfied by ϱ)

- If the first one is smaller, set A to 1, $S := S[A{:=}1]$
- Otherwise, set A to 0, $S := S[A{:=}0]$

Computing E(#clauses of S not satisfied by ϱ)

- Let $S \stackrel{\text{def}}{=} [C_1, \ldots, C_m]$, each C_j being a clause or $\top$
  $E(S) \stackrel{\text{def}}{=} E(\#j \;/\, C_j$ not satisfied by ϱ, ϱ uniformly random)

Computing E(#clauses of S not satisfied by ϱ)

- Let $S \stackrel{\text{def}}{=} [C_1, \ldots, C_m]$, each C_j being a clause or $\top$
  $E(S) \stackrel{\text{def}}{=} E(\# j \,/\, C_j$ not satisfied by ϱ, ϱ uniformly random$)$
- $E(S) = \Sigma_{j=1}{}^m \, E([C_j]) = \Sigma_{j=1}{}^m \, \Pr_\varrho(\text{not } \varrho \vDash C_j)$
  [linearity of expectation]

Computing E(#clauses of S not satisfied by ϱ)

- Let $S \stackrel{\text{def}}{=} [C_1, \ldots, C_m]$, each C_j being a clause or $\top$
  $E(S) \stackrel{\text{def}}{=} E(\# j \,/\, C_j$ not satisfied by ϱ, ϱ uniformly random)
- $E(S) = \Sigma_{j=1}{}^m\, E([C_j]) = \Sigma_{j=1}{}^m\, \Pr_\varrho(\text{not } \varrho \models C_j)$
  [linearity of expectation]
- If C_j is a tautology $A \vee \neg A \vee \ldots$ (or $\top$), $\Pr_\varrho(\text{not } \varrho \models C) = 0$
  else $\Pr_\varrho(\text{not } \varrho \models C) = 1/2^{|C|}$,
  e.g., $\Pr_\varrho(\text{not } \varrho \models A \vee \neg B \vee \neg C) = 1/8$

The key observation

- **Claim.** $\mathrm{E}(S) = \tfrac{1}{2}(\mathrm{E}(S[A{:=}1]) + \mathrm{E}(S[A{:=}0]))$

The key observation

- **Claim.** $E(S) = ½(E(S[A{:=}1]) + E(S[A{:=}0]))$
- *Proof.* By linearity of expectation, enough to check it for a single clause C_j

The key observation

- **Claim.** E(S) = ½(E($S[A{:=}1]$)+E($S[A{:=}0]$))
- *Proof.* By linearity of expectation, enough to check it for a single clause C_j
- If C_j tautology, 0 = ½(0+0), otherwise…

The key observation

- **Claim.** $\mathrm{E}(S) = ½(\mathrm{E}(S[A{:=}1]) + \mathrm{E}(S[A{:=}0]))$
- *Proof.* By linearity of expectation, enough to check it for a single clause C_j
- If C_j tautology, $0 = ½(0+0)$, otherwise…
- If $C_j = A \vee \mathit{rest}$, $\mathrm{E}(C_j) = 1/2^{|C_j|} = ½\, 1/2^{|\mathit{rest}|}$,

$$\mathrm{E}(C_j[A{:=}1]) = \mathrm{E}(\top) = 0$$
$$\mathrm{E}(C_j[A{:=}0]) = \mathrm{E}(\mathit{rest}) = 1/2^{|\mathit{rest}|}$$

The key observation

- **Claim.** $\mathrm{E}(S) = ½(\mathrm{E}(S[A{:=}1]) + \mathrm{E}(S[A{:=}0]))$
- *Proof.* By linearity of expectation, enough to check it for a single clause C_j
- If C_j tautology, $0 = ½(0+0)$, otherwise…
- If $C_j = A \vee rest$, $\mathrm{E}(C_j) = 1/2^{|C_j|} = ½\, 1/2^{|rest|}$,

  $$\mathrm{E}(C_j[A{:=}1]) = \mathrm{E}(\top) = 0$$

  $$\mathrm{E}(C_j[A{:=}0]) = \mathrm{E}(rest) = 1/2^{|rest|}$$
- Similarly if $C_j = \neg A \vee rest$

The key observation

- **Claim.** $\mathrm{E}(S) = ½(\mathrm{E}(S[A{:=}1])+\mathrm{E}(S[A{:=}0]))$
- *Proof.* By linearity of expectation, enough to check it for a single clause C_j
- If C_j tautology, $0 = ½(0+0)$, otherwise…
- If $C_j = A \vee \mathit{rest}$, $\mathrm{E}(C_j) = 1/2^{|C_j|} = ½\, 1/2^{|\mathit{rest}|}$,

  $\mathrm{E}(C_j[A{:=}1]) = \mathrm{E}(\top) = 0$

  $\mathrm{E}(C_j[A{:=}0]) = \mathrm{E}(\mathit{rest}) = 1/2^{|\mathit{rest}|}$
- Similarly if $C_j = \neg A \vee \mathit{rest}$
- If neither A nor $\neg A$ occurs in C_j, $C_j[A{:=}1] = C_j[A{:=}0] = C_j$. □

Decreasing expectations

- **Claim.** $\mathrm{E}(S) = \frac{1}{2}(\mathrm{E}(S[A{:=}1]) + \mathrm{E}(S[A{:=}0]))$

Decreasing expectations

- **Claim.** $\mathrm{E}(S) = ½(\mathrm{E}(S[A{:=}1])+\mathrm{E}(S[A{:=}0]))$
- List all the variables as $A_0, \ldots, A_n$. Set $S_0 \stackrel{\text{def}}{=} S$ $(= [C_1, \ldots, C_m])$

Decreasing expectations

- **Claim.** $\mathrm{E}(S) = ½(\mathrm{E}(S[A{:=}1]) + \mathrm{E}(S[A{:=}0]))$
- List all the variables as $A_0, \ldots, A_n$. Set $S_0 \stackrel{\text{def}}{=} S$ $(= [C_1, \ldots, C_m])$

$\mathrm{E}(S_0[A_1{:=}1]) \leq \mathrm{E}(S_0[A_1{:=}0])$?

yes

no

set $A_1{:=}1$

$S_1{:=}S[A_1{:=}1]$

set $A_1{:=}0$

$S_1{:=}S[A_1{:=}0]$

Decreasing expectations

- **Claim.** $\mathrm{E}(S) = ½(\mathrm{E}(S[A{:=}1])+\mathrm{E}(S[A{:=}0]))$
- List all the variables as $A_0, \ldots, A_n$. Set $S_0 \stackrel{\text{def}}{=} S$ $(= [C_1, \ldots, C_m])$

$\mathrm{E}(S_0[A_1{:=}1]) \leq \mathrm{E}(S_0[A_1{:=}0])$?

yes: set $A_1{:=}1$, $S_1{:=}S[A_1{:=}1]$

no: set $A_1{:=}0$, $S_1{:=}S[A_1{:=}0]$

$\mathrm{E}(S_1[A_2{:=}1]) \leq \mathrm{E}(S_1[A_2{:=}0])$?

yes: set $A_2{:=}1$, $S_2{:=}S_1[A_2{:=}1]$

no: set $A_2{:=}0$, $S_2{:=}S_1[A_2{:=}0]$

Decreasing expectations

- **Claim.** $\mathrm{E}(S) = ½(\mathrm{E}(S[A{:=}1])+\mathrm{E}(S[A{:=}0]))$
- List all the variables as $A_0, \ldots, A_n$. Set $S_0 \stackrel{\text{def}}{=} S$ $(= [C_1, \ldots, C_m])$

$\mathrm{E}(S_0[A_1{:=}1]) \le \mathrm{E}(S_0[A_1{:=}0])$?

yes: set $A_1{:=}1$, $S_1{:=}S[A_1{:=}1]$

no: set $A_1{:=}0$, $S_1{:=}S[A_1{:=}0]$

$\mathrm{E}(S_1[A_2{:=}1]) \le \mathrm{E}(S_1[A_2{:=}0])$?

yes: set $A_2{:=}1$, $S_2{:=}S_1[A_2{:=}1]$

no: set $A_2{:=}0$, $S_2{:=}S_1[A_2{:=}0]$

$\mathrm{E}(S_2[A_3{:=}1]) \le \mathrm{E}(S_2[A_3{:=}0])$?

yes: set $A_3{:=}1$, $S_3{:=}S_2[A_3{:=}1]$

no: set $A_3{:=}0$, $S_3{:=}S_2[A_3{:=}0]$

etc.

Decreasing expectations

- **Claim.** $E(S) = ½(E(S[A{:=}1]) + E(S[A{:=}0]))$
- List all the variables as $A_0, \ldots, A_n$. Set $S_0 \stackrel{\text{def}}{=} S$ $(= [C_1, \ldots, C_m])$

$E(S_0[A_1{:=}1]) \leq E(S_0[A_1{:=}0])$?

yes: set $A_1{:=}1$, $S_1{:=}S[A_1{:=}1]$

no: set $A_1{:=}0$, $S_1{:=}S[A_1{:=}0]$

$E(S_1[A_2{:=}1]) \leq E(S_1[A_2{:=}0])$?

yes: set $A_2{:=}1$, $S_2{:=}S_1[A_2{:=}1]$

no: set $A_2{:=}0$, $S_2{:=}S_1[A_2{:=}0]$

$E(S_2[A_3{:=}1]) \leq E(S_2[A_3{:=}0])$?

yes: set $A_3{:=}1$, $S_3{:=}S_2[A_3{:=}1]$

no: set $A_3{:=}0$, $S_3{:=}S_2[A_3{:=}0]$

etc.

- By the claim, $E(S_{i+1}) \leq E(S_i)$. So $E(S_n) \leq E(S)$.

Decreasing expectations

- **Claim.** $E(S) = ½(E(S[A{:=}1])+E(S[A{:=}0]))$
- List all the variables as $A_0, \ldots, A_n$. Set $S_0 \stackrel{\text{def}}{=} S$ $(= [C_1, \ldots, C_m])$

$E(S_0[A_1{:=}1]) \leq E(S_0[A_1{:=}0])$?

yes: set $A_1{:=}1$, $S_1{:=}S[A_1{:=}1]$

no: set $A_1{:=}0$, $S_1{:=}S[A_1{:=}0]$

$E(S_1[A_2{:=}1]) \leq E(S_1[A_2{:=}0])$?

yes: set $A_2{:=}1$, $S_2{:=}S_1[A_2{:=}1]$

no: set $A_2{:=}0$, $S_2{:=}S_1[A_2{:=}0]$

$E(S_2[A_3{:=}1]) \leq E(S_2[A_3{:=}0])$?

yes: set $A_3{:=}1$, $S_3{:=}S_2[A_3{:=}1]$

no: set $A_3{:=}0$, $S_3{:=}S_2[A_3{:=}0]$

etc.

- By the claim, $E(S_{i+1}) \leq E(S_i)$. So $E(S_n) \leq E(S)$.
- Let ϱ be the final environment
  The only clauses in S_n are $\top$ (if $\varrho \vDash C_j$), or the empty clause $\bot$

Decreasing expectations

- **Claim.** $E(S) = ½(E(S[A{:=}1])+E(S[A{:=}0]))$
- List all the variables as $A_0, \ldots, A_n$. Set $S_0 \stackrel{\text{def}}{=} S$ $(= [C_1, \ldots, C_m])$

$E(S_0[A_1{:=}1]) \leq E(S_0[A_1{:=}0])$?
yes: set $A_1{:=}1$, $S_1{:=}S[A_1{:=}1]$
no: set $A_1{:=}0$, $S_1{:=}S[A_1{:=}0]$

$E(S_1[A_2{:=}1]) \leq E(S_1[A_2{:=}0])$?
yes: set $A_2{:=}1$, $S_2{:=}S_1[A_2{:=}1]$
no: set $A_2{:=}0$, $S_2{:=}S_1[A_2{:=}0]$

$E(S_2[A_3{:=}1]) \leq E(S_2[A_3{:=}0])$?
yes: set $A_3{:=}1$, $S_3{:=}S_2[A_3{:=}1]$
no: set $A_3{:=}0$, $S_3{:=}S_2[A_3{:=}0]$

etc.

- By the claim, $E(S_{i+1}) \leq E(S_i)$. So $E(S_n) \leq E(S)$.
- Let ϱ be the final environment
  The only clauses in S_n are $\top$ (if $\varrho \vDash C_j$), or the empty clause $\bot$
- Note: $E(S_n) = \#$empty clauses in S_n.
  So ϱ satisfies $m - E(S_n) \geq m - E(S)$ clauses in S.

MAXSAT is approximable

- ϱ satisfies m–E(S_n) ≥ m–E(S) clauses in $S = [C_1, \ldots, C_m]$

MAXSAT is approximable

- ϱ satisfies m–E(S_n) ≥ m–E(S) clauses in $S = [C_1, \ldots, C_m]$
- If each non-tautological C_j has $\geq k$ literals,
  $\Pr_\varrho(\text{not } \varrho \vDash C_j) \leq 1/2^k$ (=0 if tautological), so
  $E(S) = \Sigma_{j=1}^{m} \Pr_\varrho(\text{not } \varrho \vDash C_j) \leq m/2^k$

MAXSAT is approximable

- ϱ satisfies $m - \mathrm{E}(S_n) \geq m - \mathrm{E}(S)$ clauses in $S = [C_1, \ldots, C_m]$
- If each non-tautological C_j has $\geq k$ literals,
  $\Pr_\varrho(\text{not } \varrho \vDash C_j) \leq 1/2^k$ (=0 if tautological), so
  $\mathrm{E}(S) = \Sigma_{j=1}^{m} \Pr_\varrho(\text{not } \varrho \vDash C_j) \leq m/2^k$
- Therefore ϱ satisfies $\geq m\,(1 - 1/2^k) \geq \mathrm{opt}(S)\,(1 - 1/2^k)$ clauses in S:

MAXSAT is approximable

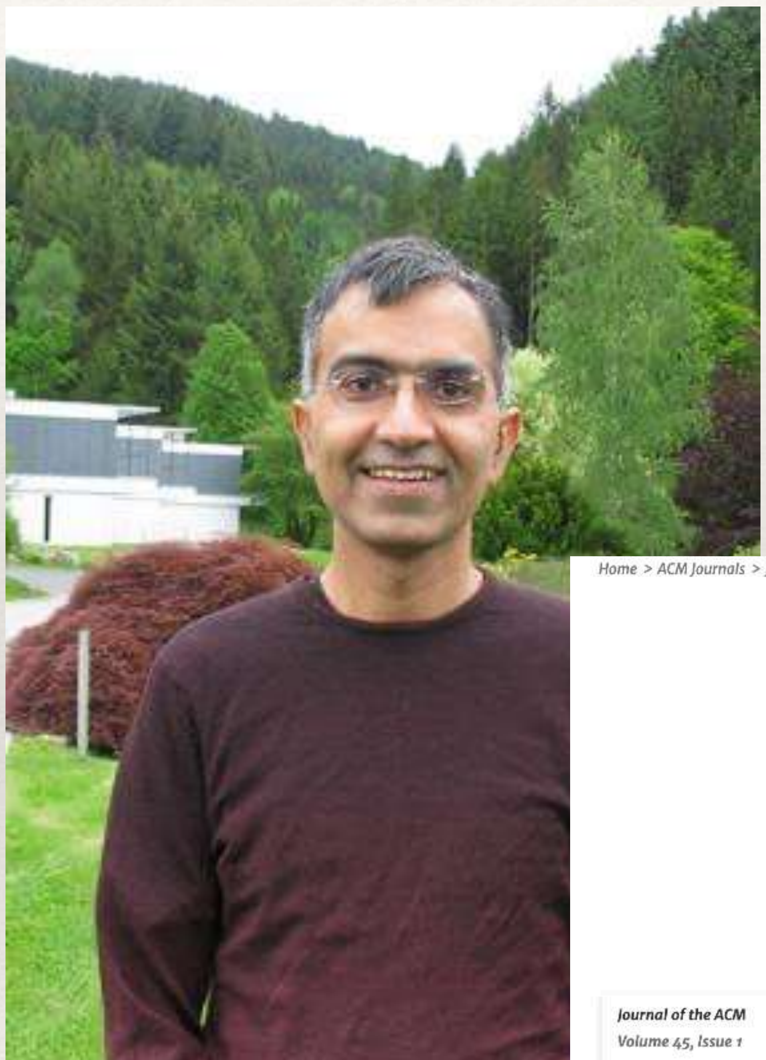
- ϱ satisfies m–E(S_n) ≥ m–E(S) clauses in $S = [C_1, \ldots, C_m]$
- If each non-tautological C_j has $\geq k$ literals, $\Pr_\varrho(\text{not } \varrho \vDash C_j) \leq 1/2^k$ (=0 if tautological), so $E(S) = \Sigma_{j=1}{}^m \Pr_\varrho(\text{not } \varrho \vDash C_j) \leq m/2^k$
- Therefore ϱ satisfies $\geq m\,(1- 1/2^k) \geq \text{opt}(S)\,(1- 1/2^k)$ clauses in S:
- **Thm. MAXSAT** restricted to S / each non-tautological C_j has $\geq k$ literals, is $1/2^k$-approximable.

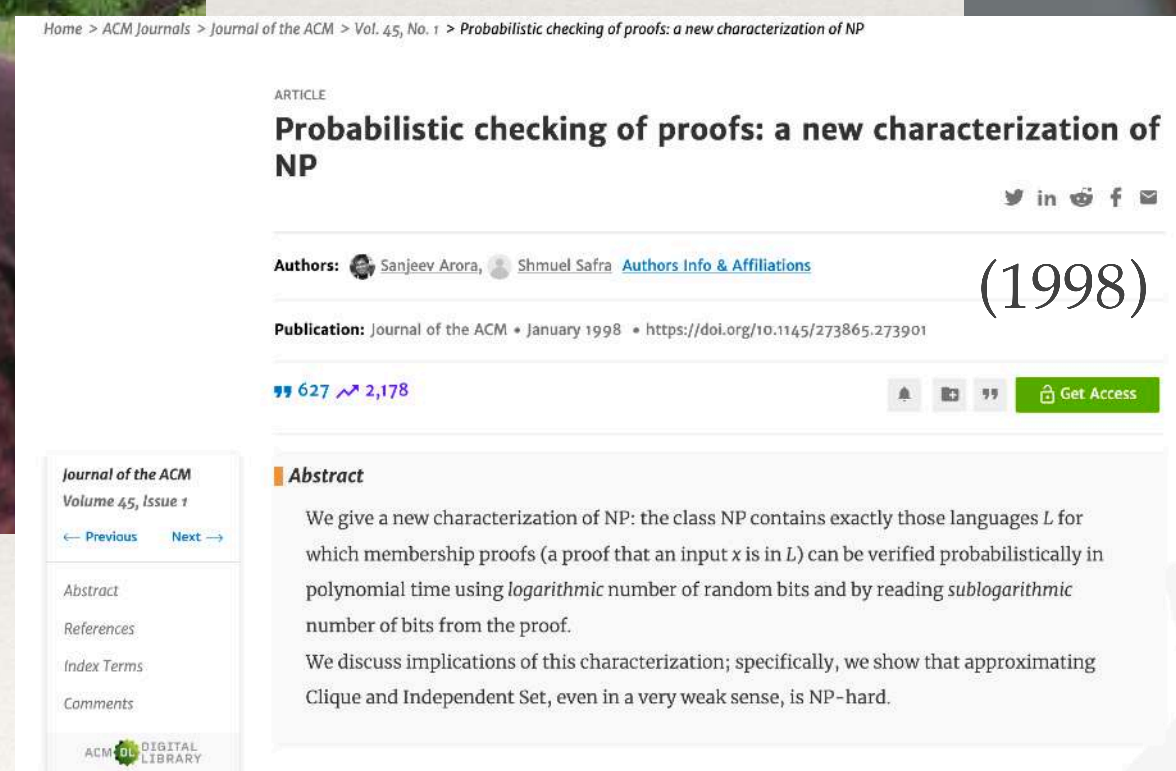
MAXSAT is approximable

- **Thm. MAXSAT** restricted to S / each non-tautological C_j has $\geq k$ literals, is $1/2^k$-approximable.
- One can always prepare S by eliminating unit clauses, so $k \geq 2$: **MAXSAT** is 1/4-approximable.
- If every clause in S has **at least** 3 literals, then 1/8-approximable.
- Hence **MAX=3SAT** (all clauses have exactly 3 literals) is **1/8-approximable**. It turns out that this is optimal.

PCP

Sanjeev Arora, Shmuel Safra





Home > ACM Journals > Journal of the ACM > Vol. 45, No. 1 > Probabilistic checking of proofs: a new characterization of NP

ARTICLE

Probabilistic checking of proofs: a new characterization of NP

Authors: Sanjeev Arora, Shmuel Safra Authors Info & Affiliations

Publication: Journal of the ACM • January 1998 • https://doi.org/10.1145/273865.273901

627 2,178

Get Access

Journal of the ACM
Volume 45, Issue 1

← Previous Next →

Abstract
References
Index Terms
Comments

ACM DL DIGITAL LIBRARY

Abstract

We give a new characterization of NP: the class NP contains exactly those languages L for which membership proofs (a proof that an input x is in L) can be verified probabilistically in polynomial time using *logarithmic* number of random bits and by reading *sublogarithmic* number of bits from the proof.

We discuss implications of this characterization; specifically, we show that approximating Clique and Independent Set, even in a very weak sense, is NP-hard.

(1998)

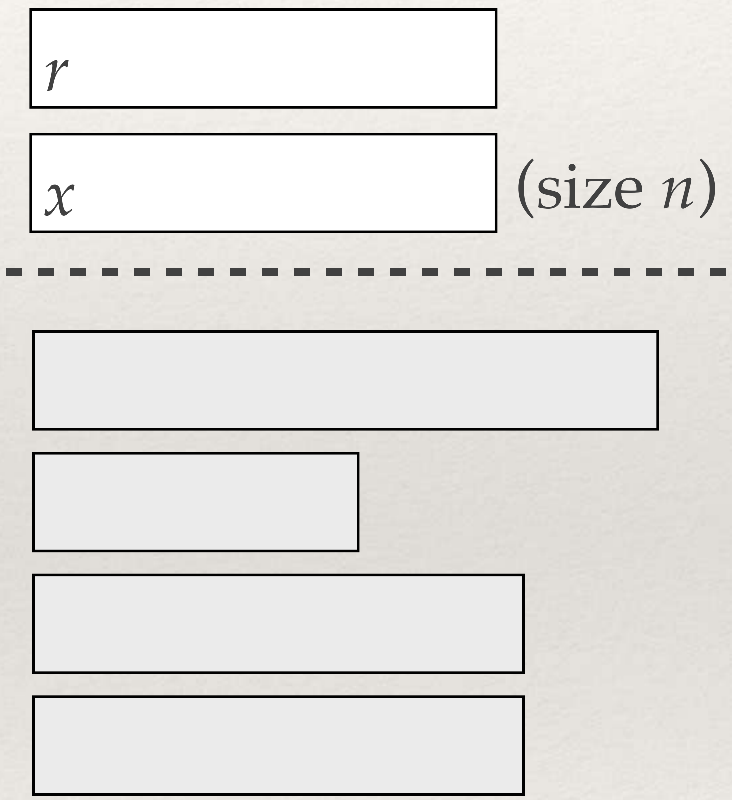
https://commons.wikimedia.org/wiki/File:Sanjeev_Arora.jpg#/media/Fichier:Sanjeev_Arora.jpg

https://simons.berkeley.edu/sites/default/files/styles/profile_main/public/dscn0007.jpg?itok=irtfi766

Reminder: randomized TMs

- Two **read-only** tapes

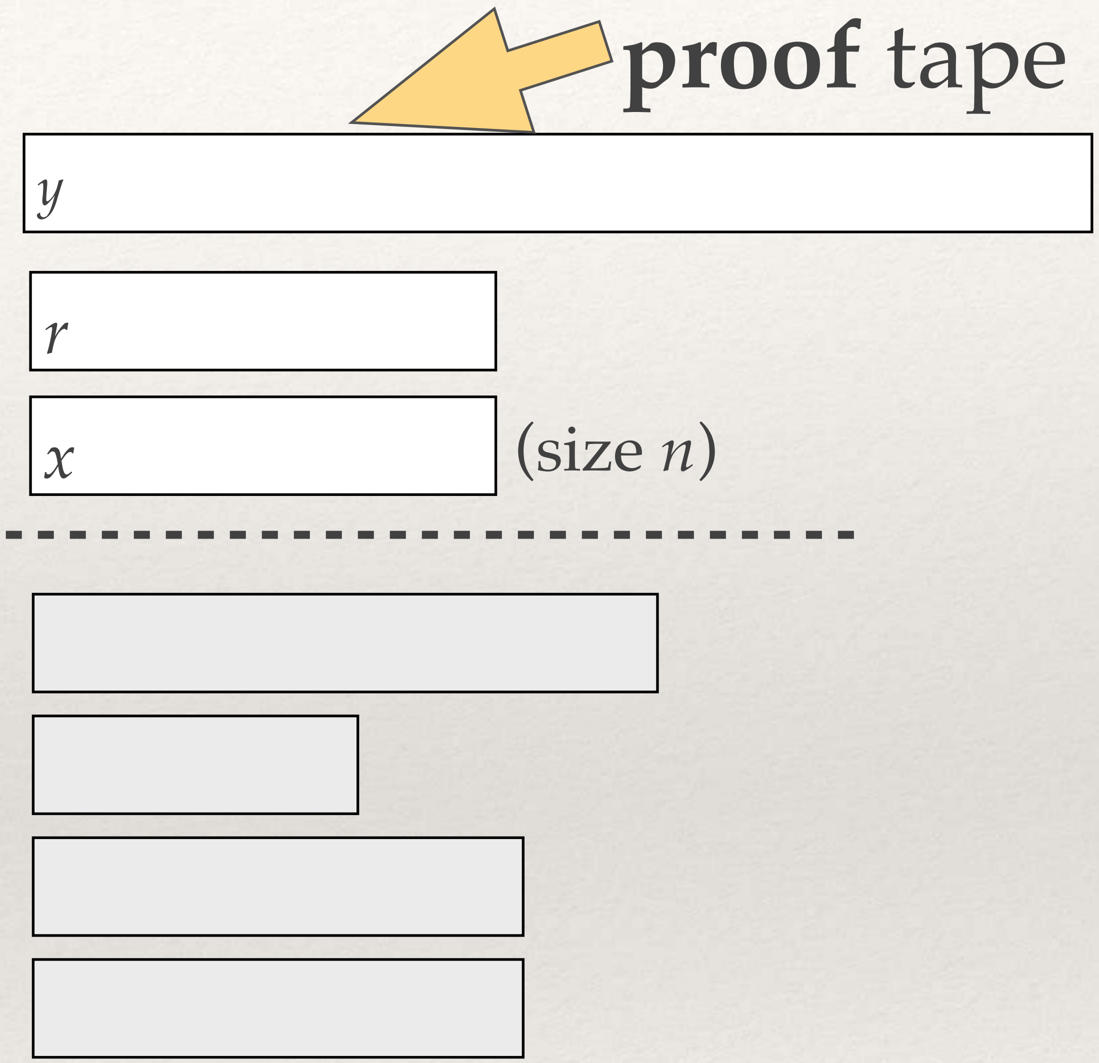
- As many **work tapes** as you need (but only a constant number!)



With the usual proviso:
head can only move right on random tape r

PCP machines

- ~~Two~~ Three **read-only** tapes
- As many **work tapes** as you need (but only a constant number!)

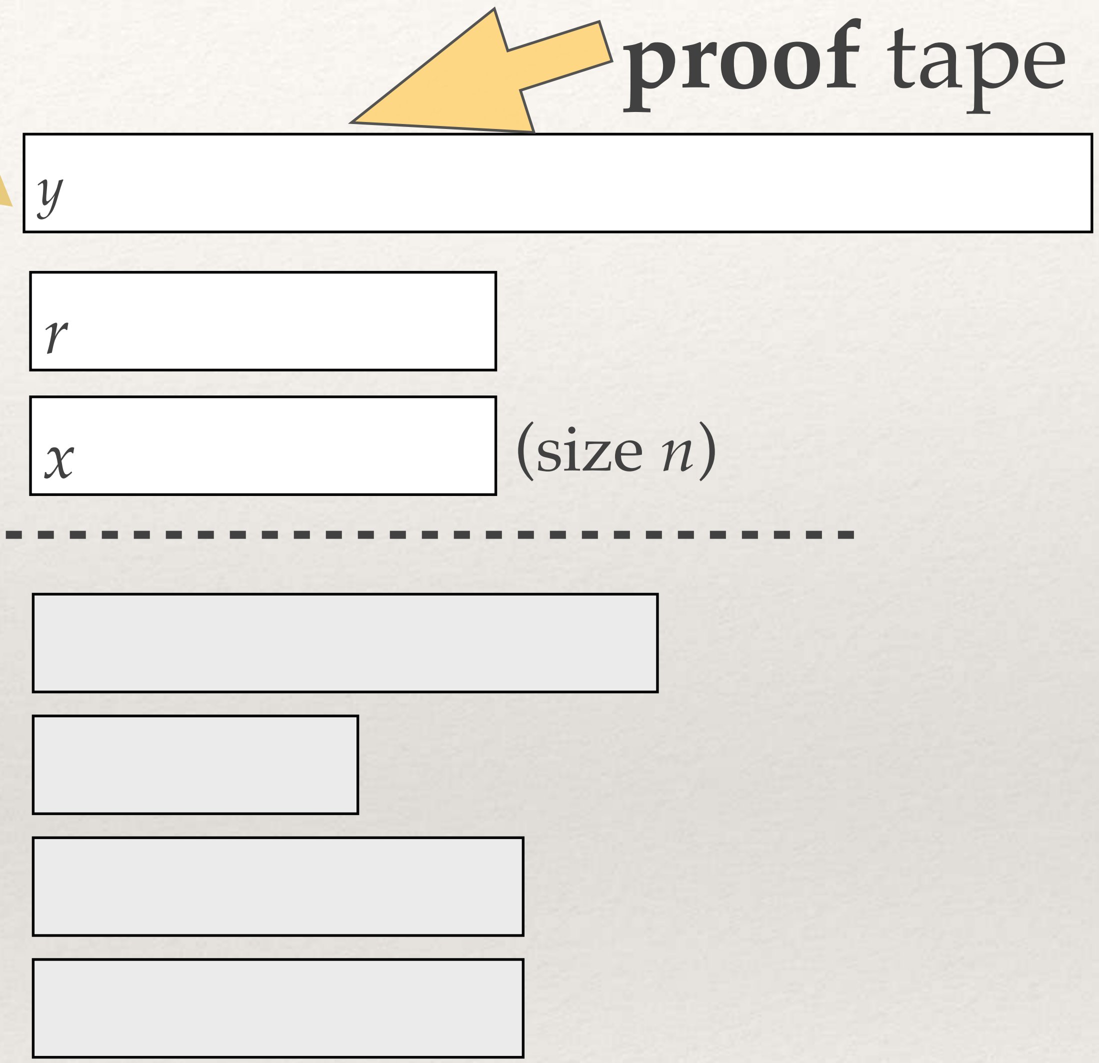


With the usual proviso:
head can only move right on random tape r

PCP machines

Proof tape is accessed in **random access** mode

- ~~Two~~ Three **read-only** tapes



- As many **work tapes** as you need (but only a constant number!)

With the usual proviso:
head can only move right on random tape r

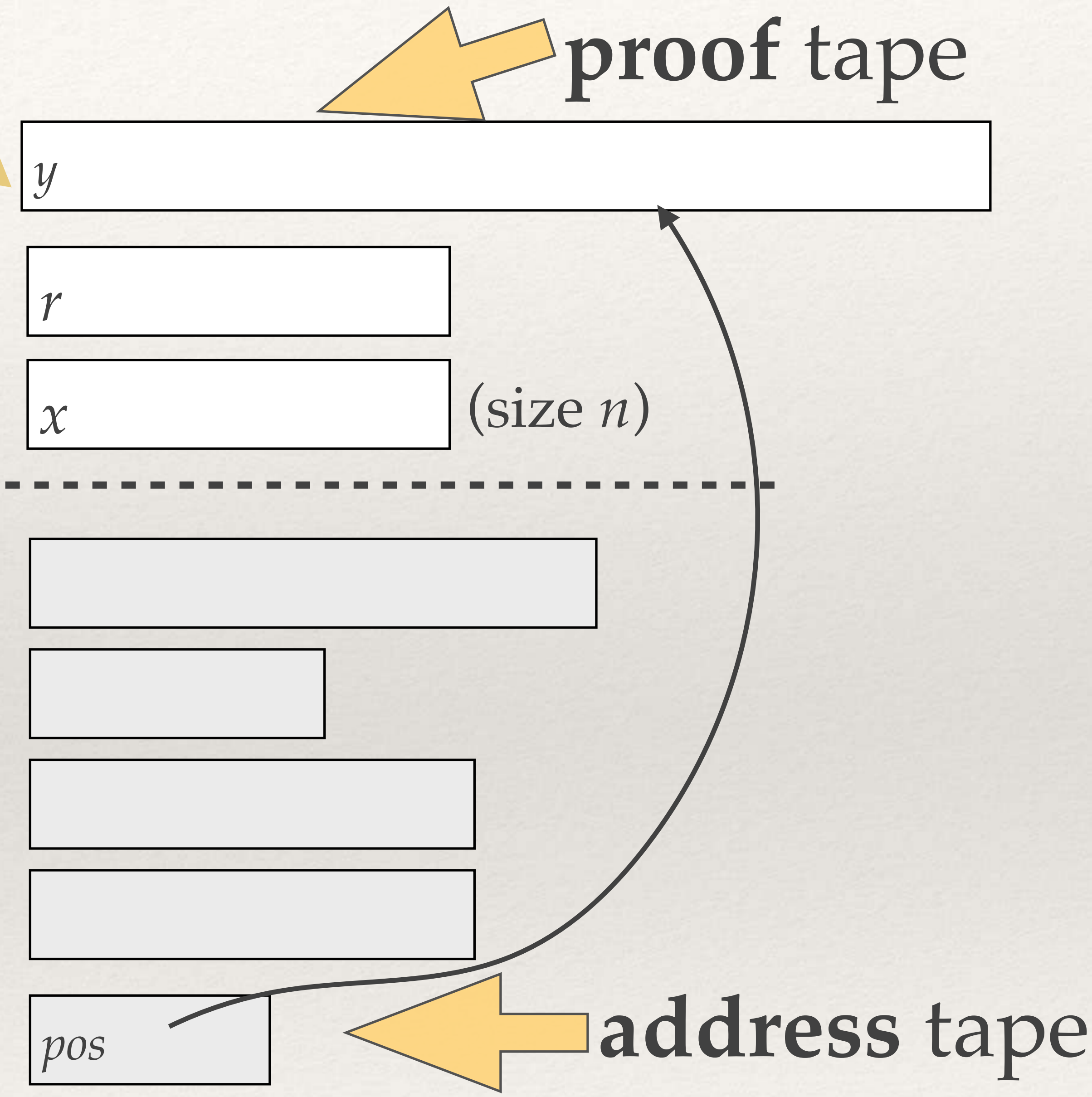
PCP machines

Proof tape is accessed in **random access** mode

Three

- ~~Two~~ **read-only** tapes

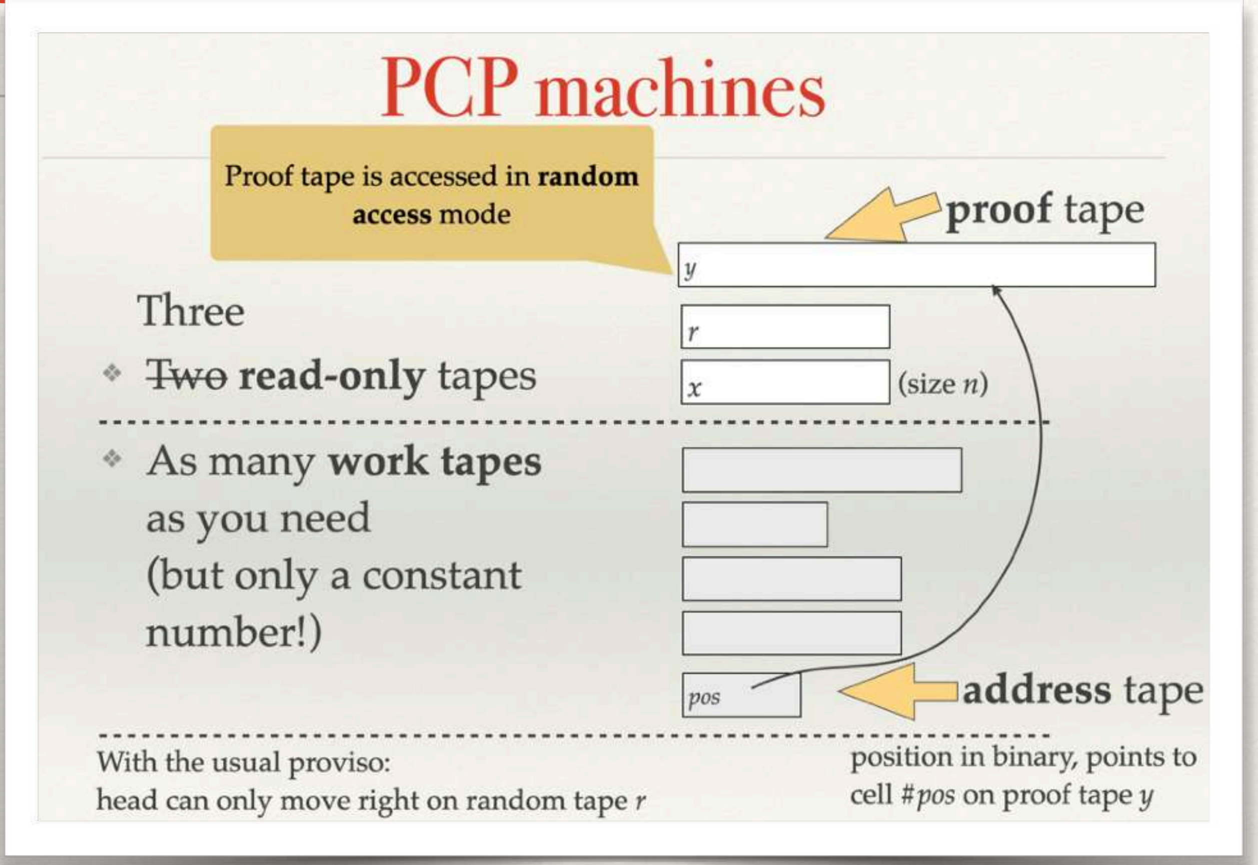
- As many **work tapes** as you need (but only a constant number!)



With the usual proviso:
head can only move right on random tape r

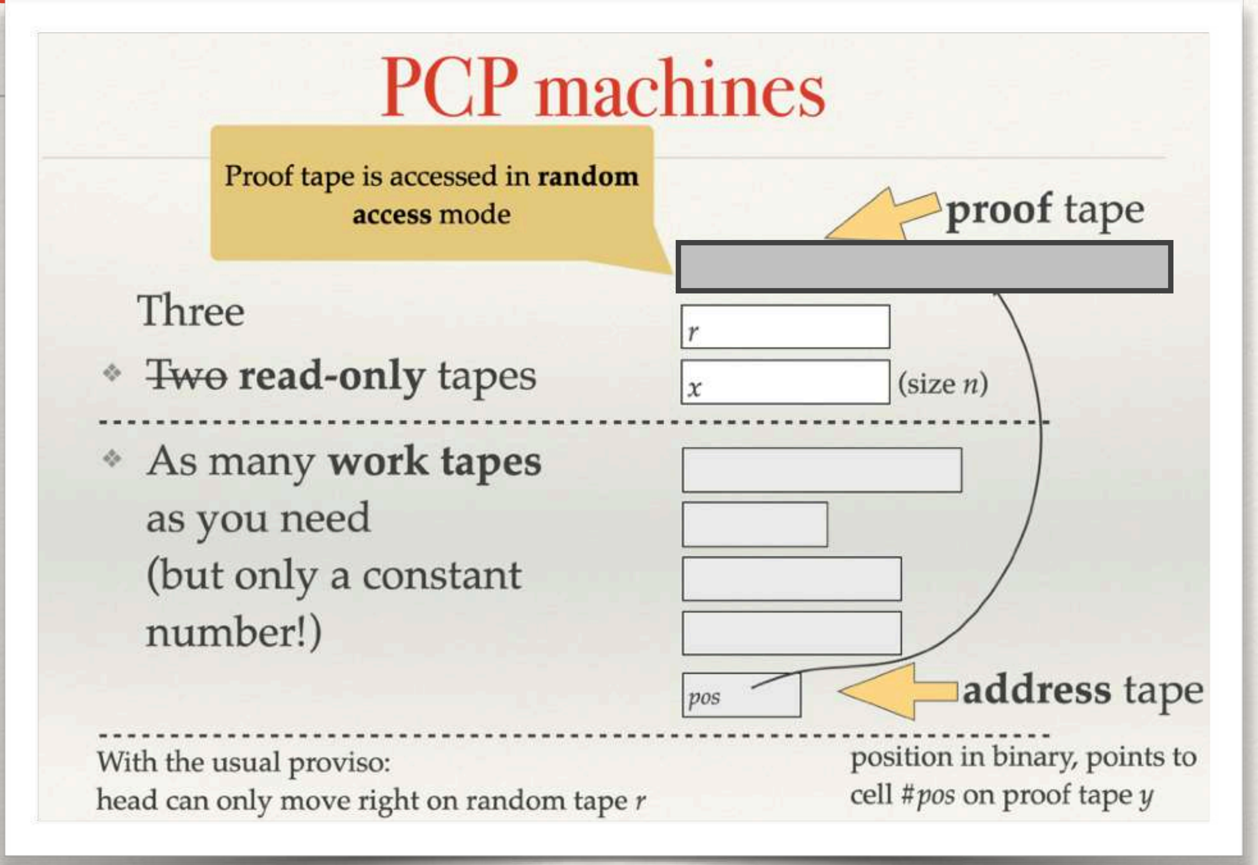
position in binary, points to cell #pos on proof tape y

Running a PCP machine

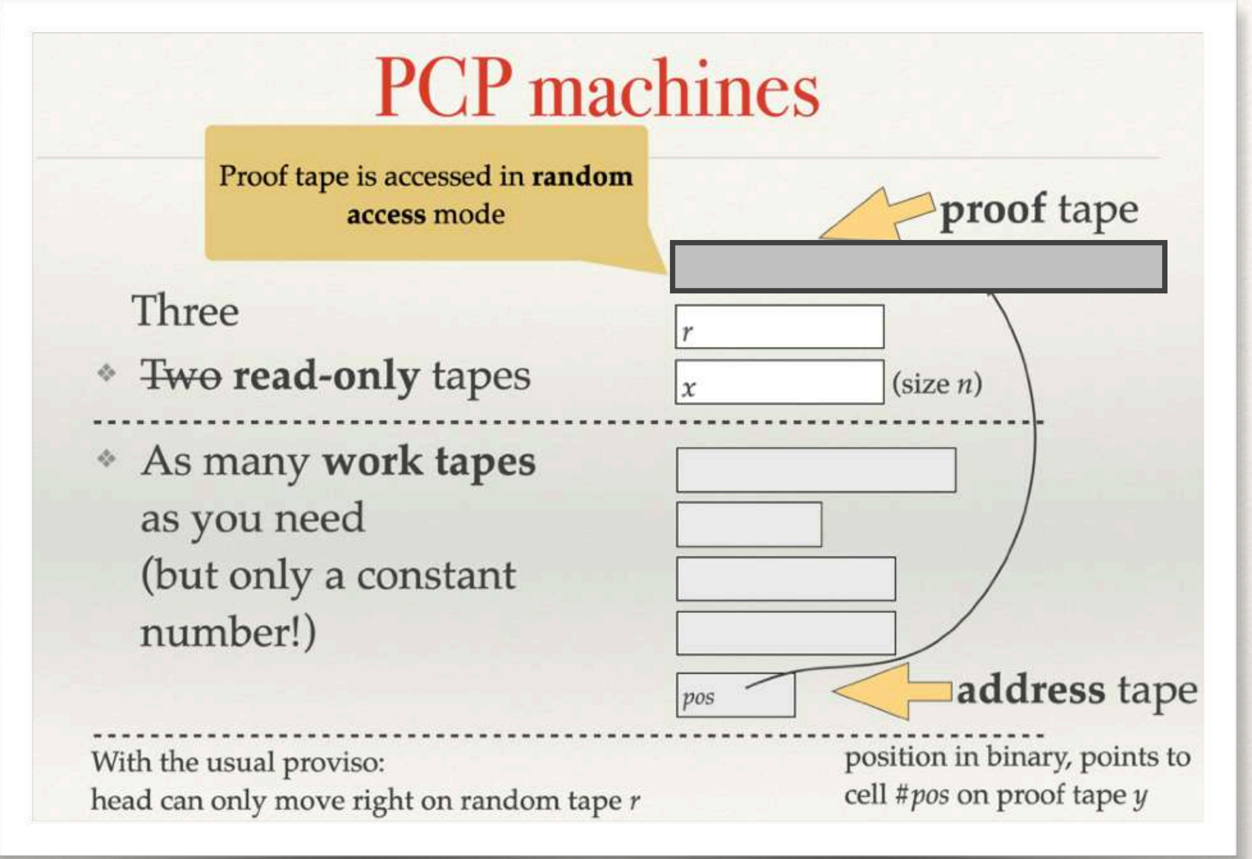


Running a PCP machine

1. On input x, Merlin fills in proof tape y — but keeps it **masked** (= cryptographic **commitment**)

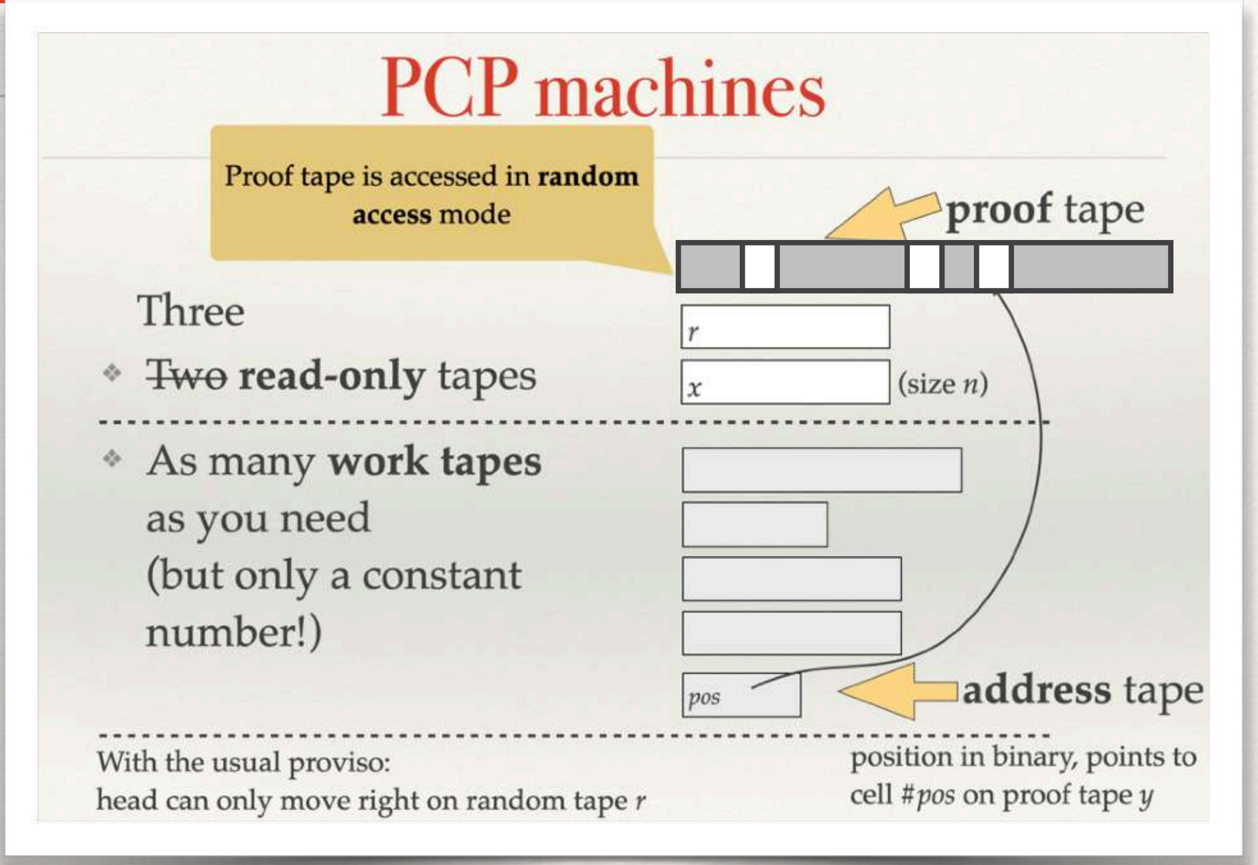


Running a PCP machine



1. On input x, Merlin fills in proof tape y — but keeps it **masked** (= cryptographic **commitment**)
2. Arthur, only knowing $|y|$ (and x), computes k=$\boldsymbol{Q(n)}$ **positions** $p_1, \ldots, p_k$ on the proof tape in binary, in polynomial time, using $\boldsymbol{R(n)}$ **random bits**

Running a PCP machine

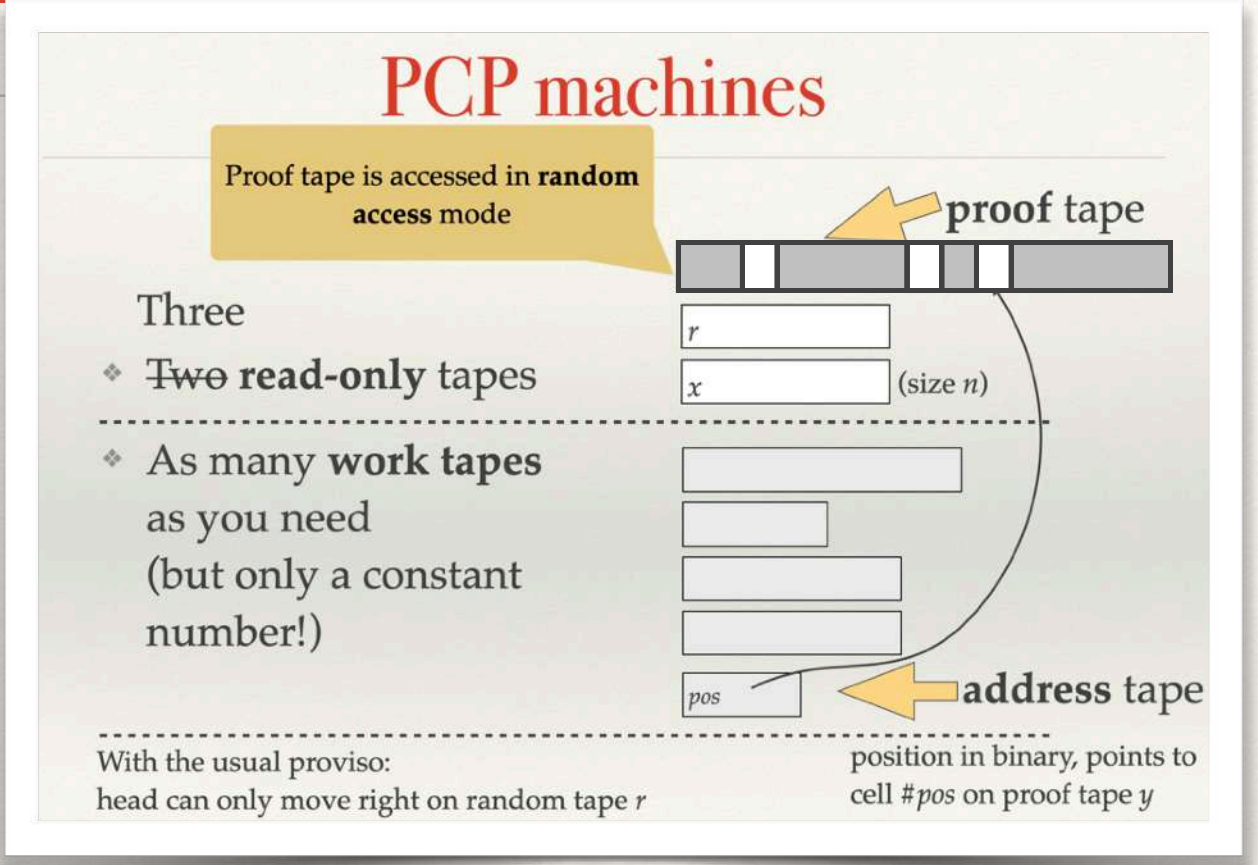


1. On input x, Merlin fills in proof tape y — but keeps it **masked** (= cryptographic **commitment**)

2. Arthur, only knowing $|y|$ (and x), computes k=$\boldsymbol{Q(n)}$ **positions** $p_1, \ldots, p_k$ on the proof tape in binary, in polynomial time, using $\boldsymbol{R(n)}$ **random bits**

3. Merlin **reveals** $y[p_1], \ldots, y[p_k]$

Running a PCP machine



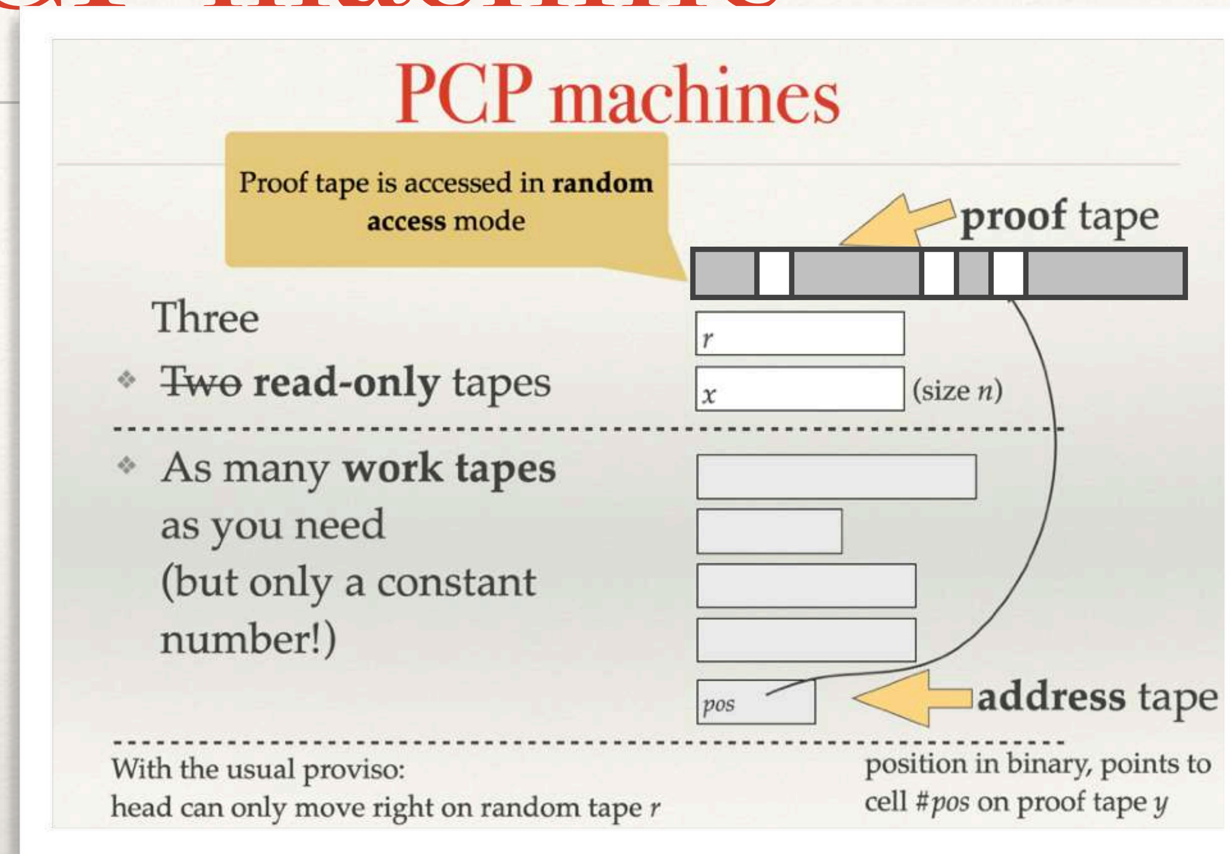
1. On input x, Merlin fills in proof tape y — but keeps it **masked** (= cryptographic **commitment**)

2. Arthur, only knowing $|y|$ (and x), computes k=$\boldsymbol{Q(n)}$ **positions** $p_1, \ldots, p_k$ on the proof tape in binary, in polynomial time, using $\boldsymbol{R(n)}$ **random bits**

3. Merlin **reveals** $y[p_1], \ldots, y[p_k]$

4. Arthur computes $f(y[p_1], \ldots, y[p_k]) \in \{$**accept, reject**$\}$ in time $T(n)$

Running a PCP machine



1. On input x, Merlin fills in proof tape y — but keeps it **masked** (= cryptographic **commitment**)

2. Arthur, only knowing $|y|$ (and x), computes k=$\boldsymbol{Q(n)}$ **positions** $p_1, \ldots, p_k$ on the proof tape in binary, in polynomial time, using $\boldsymbol{R(n)}$ **random bits**

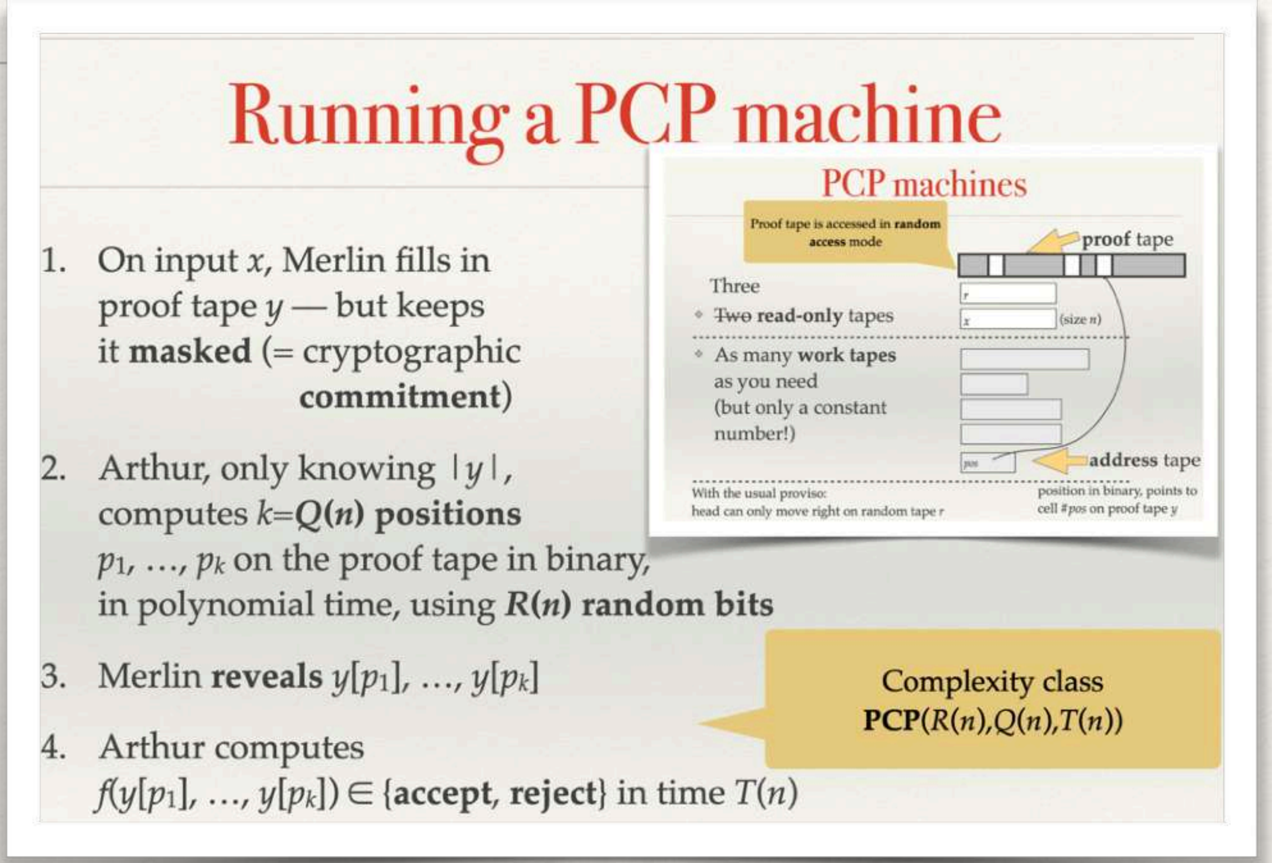
3. Merlin **reveals** $y[p_1], \ldots, y[p_k]$

4. Arthur computes $f(y[p_1], \ldots, y[p_k]) \in \{$**accept**, **reject**$\}$ in time $T(n)$

where f may also depend on x, and on the random bits of step 2

Acceptance conditions

- If $x \in L$, then Merlin can provide a proof tape y such that Arthur will **always** accept
- If $x \notin L$, then whichever proof tape Merlin provides, Arthur will reject **with probability** $\geq$ **½**
- The languages L that can be decided this way form the complexity class **PCP**$(R(n),Q(n),T(n))$



The Arora-Safra theorem

- **Theorem. NP** = **PCP**(O(log n), O(1), O(1))
- I.e., one can decide every language in **NP** by running a PCP machine that:
  — asks $Q(n)$=**O(1) questions** (positions)
  — computed using only $R(n)$=**O(log n) random bits**, in poly time
  — and finally decides in $T(n)$=**O(1) time**.
- Proof would require a whole term!

The Arora-Safra theorem

- **Theorem. NP** = **PCP**(O(log n), O(1), O(1))

NP=PCP,
for short

- I.e., one can decide every language in **NP** by running a PCP machine that:
  — asks $Q(n)$=**O(1) questions** (positions)
  — computed using only $R(n)$=**O(log n) random bits**, in poly time
  — and finally decides in $T(n)$=**O(1) time**.

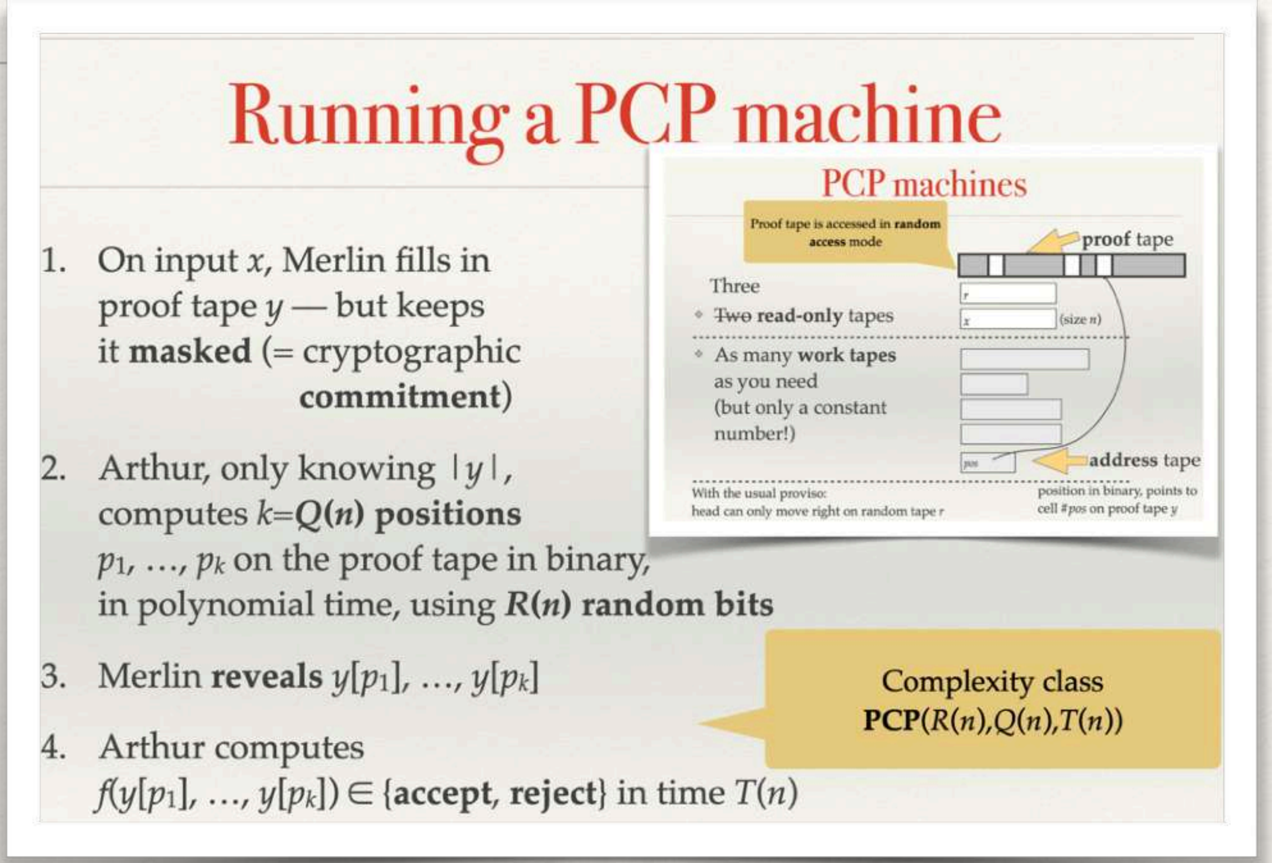
- Proof would require a whole term!

NP=PCP and the hardness of approximation

- What I will explain is that **NP=PCP** is equivalent to the ε-inapproximability of **MAX3SAT** for some ε>0
- Arora-Safra prove **NP=PCP**
- … and there is a simplified (still extremely complex) proof by Irit Dinur

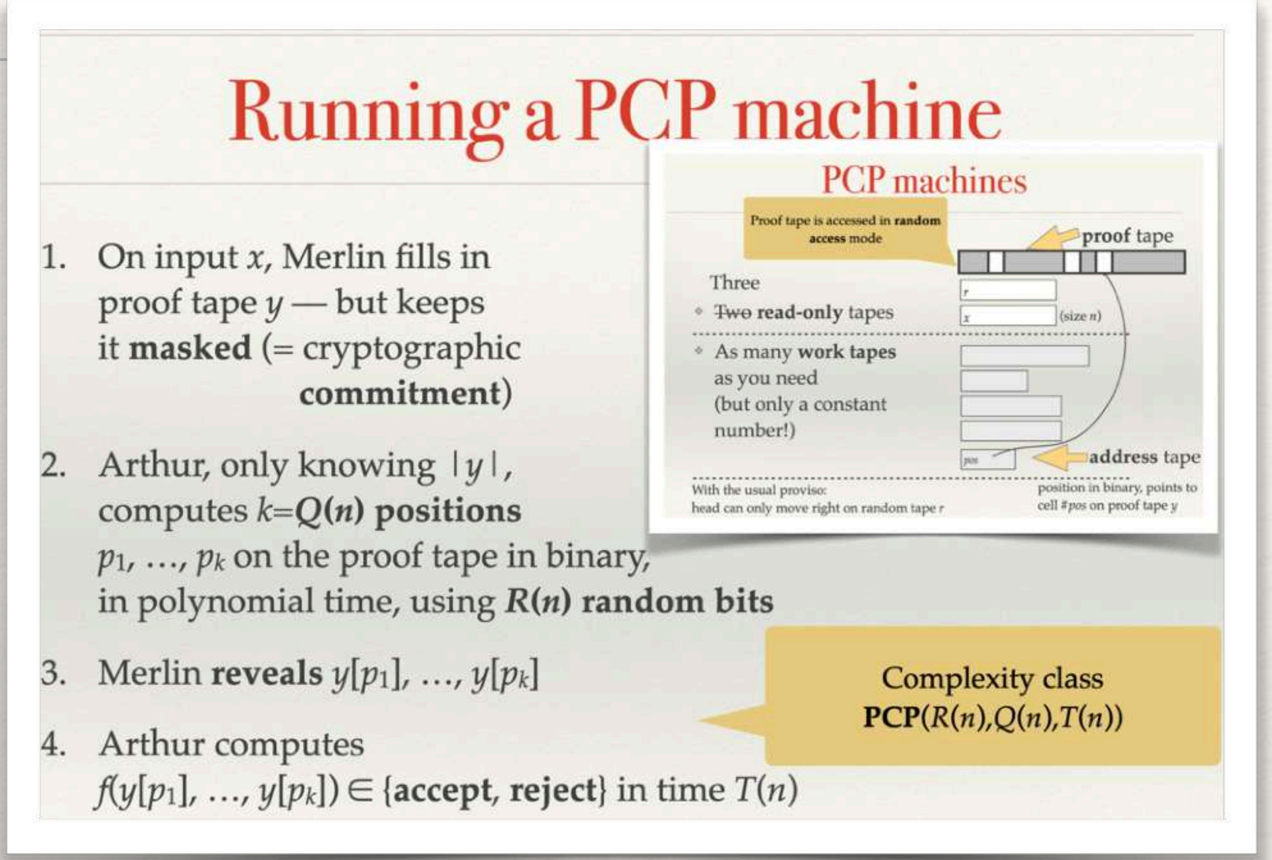
The easy direction: PCP ⊆ NP

- **Derandomize** naively: for every string of $R(n)$ random bits, simulate Arthur's computation
- If more than ½ of the simulations accept, then accept, else reject

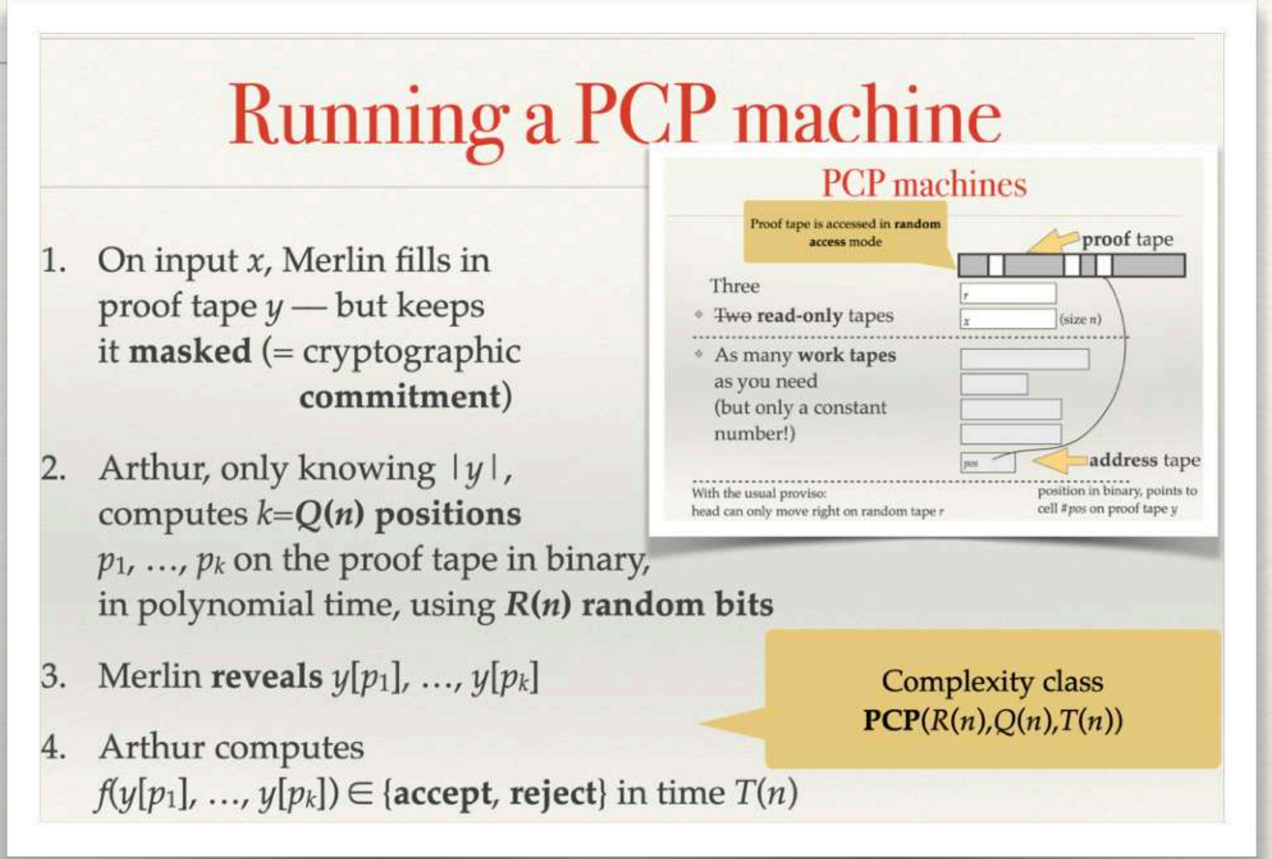


The easy direction: **PCP ⊆ NP**

- **Derandomize** naively: for every string of $R(n)$ random bits, simulate Arthur's computation
- If more than ½ of the simulations accept, then accept, else reject
- Works in time $2^{R(n)} \log R(n)\ \mathrm{poly}(n)+T(n)$



The easy direction: **PCP ⊆ NP**



- **Derandomize** naively: for every string of $R(n)$ random bits, simulate Arthur's computation
- If more than ½ of the simulations accept, then accept, else reject
- Works in time $2^{R(n)}\log R(n)\ \mathrm{poly}(n)+T(n)$
- So **PCP**($R(n) \stackrel{\text{def}}{=} O(\log n)$, $Q(n) \stackrel{\text{def}}{=}$ whatever, $T(n) \stackrel{\text{def}}{=} \mathrm{poly}(n)$) ⊆ **NP**

PCP and the hardness of approximating **SAT**

MAX3SAT(ε)

- … is the following **promise problem**:
  INPUT: a finite set S of m propositional 3-clauses
  PROMISE: S satisfiable / $\mathrm{opt}(S) < (1-\varepsilon)m$
  QUESTION: which is true? [$\mathrm{opt}(S) = \max \#\mathrm{sat.\ clauses}$]

MAX3SAT(ε)

- … is the following **promise problem**:
  INPUT: a finite set S of m propositional 3-clauses
  PROMISE: S satisfiable / $\mathrm{opt}(S) < (1–\varepsilon)m$
  QUESTION: which is true? [$\mathrm{opt}(S)$ = max #sat. clauses]
- We will see that:
  — if **3SAT** is ε-approximable then **MAX3SAT**(ε)
  is polytime decidable
  — (∃ ε>0, **MAX3SAT**(ε) is **NP**-hard) iff **NP**=**PCP**

MAX3SAT(ε)

- … is the following **promise problem**:
  INPUT: a finite set S of m propositional 3-clauses
  PROMISE: S satisfiable / opt(S) < (1–ε)m
  QUESTION: which is true? [opt(S) = max #sat. clauses]
- We will see that:
  — if **3SAT** is ε-approximable then **MAX3SAT**(ε)
  is polytime decidable
  — (∃ ε>0, **MAX3SAT**(ε) is **NP**-hard) iff **NP**=**PCP**
- That was known before Arora-Safra.
  With Arora-Safra: ∃ ε>0, **3SAT** is not ε-approximable, unless **P**=**NP**

MAX3SAT(ε)

- … is the following **promise problem**:
  INPUT: a finite set S of m propositional 3-clauses
  PROMISE: S satisfiable / $\mathrm{opt}(S) < (1-\varepsilon)m$
  QUESTION: which is true? [$\mathrm{opt}(S)$ = max #sat. clauses]
- We will see that:
  — if **3SAT** is ε-approximable then **MAX3SAT**(ε) is polytime decidable
  — (∃ ε>0, **MAX3SAT**(ε) is **NP**-hard) iff **NP**=**PCP**
- That was known before Arora-Safra.
  With Arora-Safra: ∃ ε>0, **3SAT** is not ε-approximable, unless **P**=**NP**

Note: **NP**-*complete* would not make sense for promise problems

If **3SAT** ε-approximable then **MAX3SAT**(ε) polytime

- Given a (polytime) ε-approximation algorithm A for **3SAT**:

If **3SAT** ε-approximable then **MAX3SAT**(ε) polytime

- Given a (polytime) ε-approximation algorithm A for **3SAT**:
- For every instance S of **MAX3SAT**(ε),

  $\varrho \stackrel{\text{def}}{=} A(S)$ satisfies $\geq (1-\varepsilon).\mathrm{opt}(S)$ clauses of S

MAX3SAT(ε)

INPUT: a finite set S of m propositional 3-cla

PROMISE: S satisfiable / $\mathrm{opt}(S) < (1-\varepsilon)m$

QUESTION: which is true?

If **3SAT** ε-approximable then **MAX3SAT(ε)** polytime

- Given a (polytime) ε-approximation algorithm A for **3SAT**:

MAX3SAT(ε)

INPUT: a finite set S of m propositional 3-cla
PROMISE: S satisfiable / opt(S) < (1–ε)m
QUESTION: which is true?

- For every instance S of **MAX3SAT**(ε),

  $\varrho \stackrel{\text{def}}{=} A(S)$ satisfies ≥ (1–ε).opt(S) clauses of S

- If S satisfiable, opt(S)=m, so ϱ satisfies ≥ (1–ε)m clauses

If **3SAT** ε-approximable then **MAX3SAT**(ε) polytime

- Given a (polytime) ε-approximation algorithm A for **3SAT**:

MAX3SAT(ε)

INPUT: a finite set S of m propositional 3-cla
PROMISE: S satisfiable / opt(S) < (1–ε)m
QUESTION: which is true?

- For every instance S of **MAX3SAT**(ε),

  $\varrho \stackrel{\text{def}}{=} A(S)$ satisfies ≥ (1–ε).opt(S) clauses of S

- If S satisfiable, opt(S)=m, so ϱ satisfies ≥ (1–ε)m clauses

- Otherwise, ϱ satisfies < (1–ε)m clauses by the promise

If **3SAT** ε-approximable then **MAX3SAT**(ε) polytime

- Given a (polytime) ε-approximation algorithm A for **3SAT**:

MAX3SAT(ε)

INPUT: a finite set S of m propositional 3-cla
PROMISE: S satisfiable / opt(S) < (1–ε)m
QUESTION: which is true?

- For every instance S of **MAX3SAT**(ε),
  $\varrho \overset{\text{def}}{=} A(S)$ satisfies ≥ (1–ε).opt(S) clauses of S

- If S satisfiable, opt(S)=m, so ϱ satisfies ≥ (1–ε)m clauses

- Otherwise, ϱ satisfies < (1–ε)m clauses by the promise

- Hence comparing #clauses satisfied by $\varrho \overset{\text{def}}{=} A(S)$ with (1–ε)m yields a polytime algorithm deciding **MAX3SAT**(ε). □

($\exists$ ε>0, **MAX3SAT**(ε) **NP**-hard)

iff **NP=PCP**:

the left to right direction

If $\exists\ \varepsilon>0$, MAX3SAT(ε) **NP**-hard then **NP**=**PCP**

- We already know **PCP** $\subseteq$ **NP**. Conversely, let L be any language in **NP**.

If ∃ ε>0, MAX3SAT(ε) NP-hard then NP=PCP

- We already know **PCP** ⊆ **NP**. Conversely, let L be any language in **NP**.
- ∃polytime reduction from L to **MAX3SAT**(ε), since **MAX3SAT**(ε) **NP**-hard by assumption

MAX3SAT(ε)

INPUT: a finite set S of m propositional 3-cla
PROMISE: S satisfiable / opt(S) < (1–ε)m
QUESTION: which is true?

If ∃ ε>0, MAX3SAT(ε) NP-hard then NP=PCP

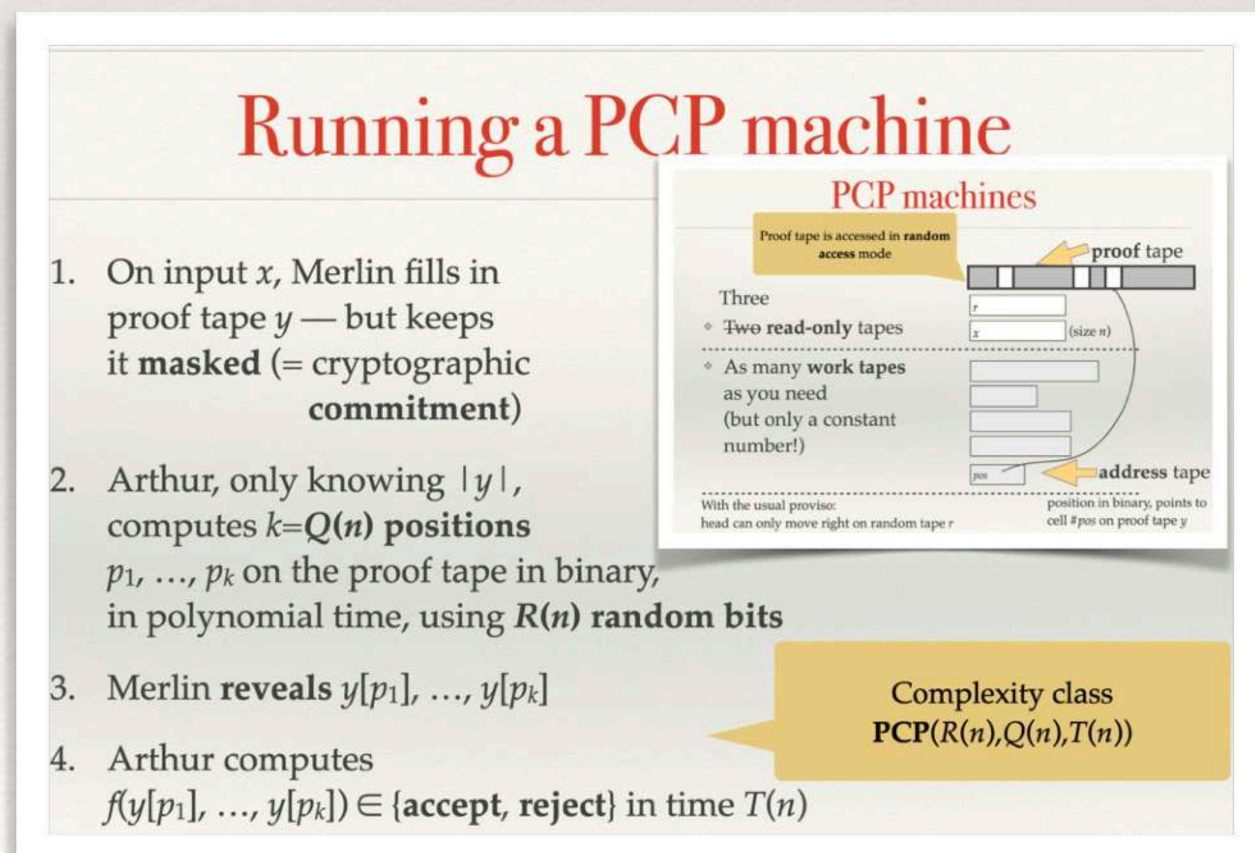
- We already know **PCP** ⊆ **NP**. Conversely, let L be any language in **NP**.

MAX3SAT(ε)

INPUT: a finite set S of m propositional 3-cla
PROMISE: S satisfiable / opt(S) < (1−ε)m
QUESTION: which is true?

- ∃polytime reduction from L to **MAX3SAT**(ε), since **MAX3SAT**(ε) **NP**-hard by assumption

- **PCP** is closed under polytime reductions (important!)



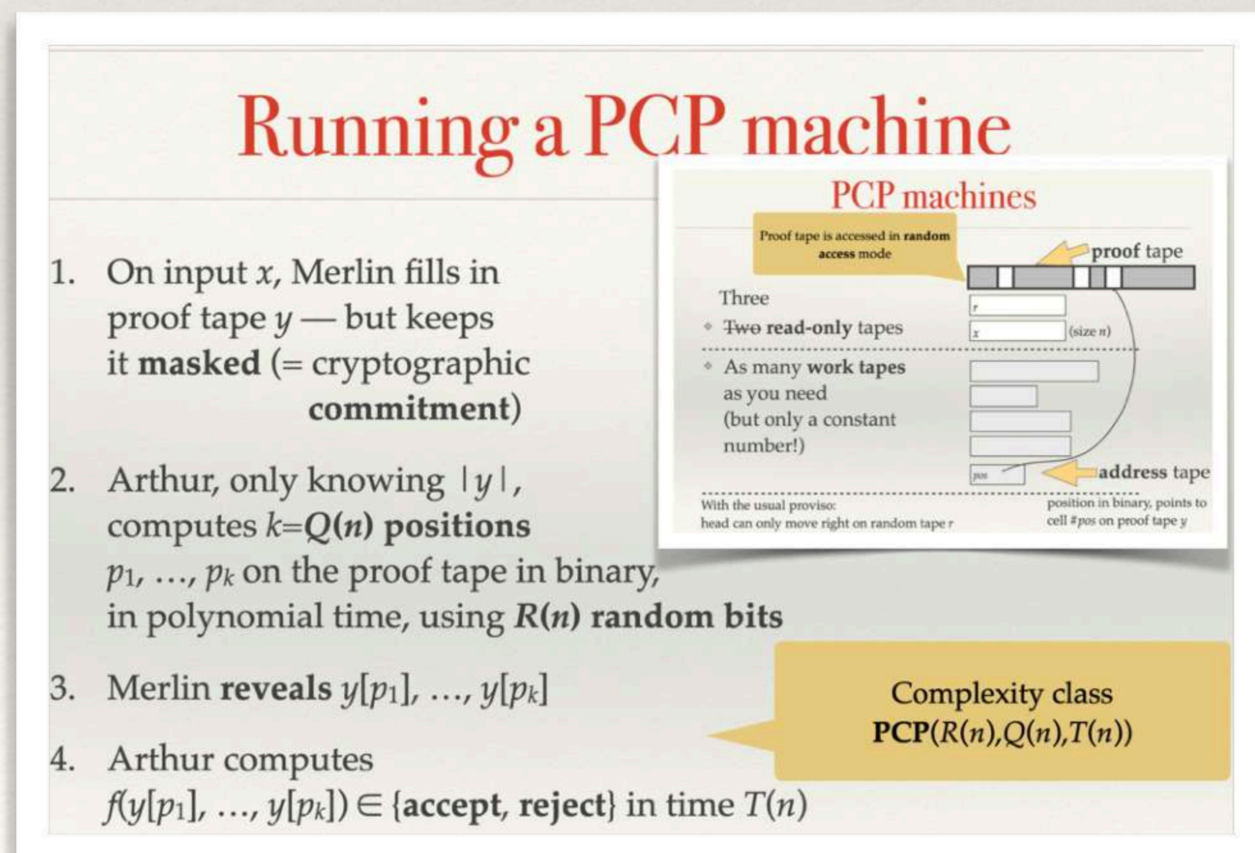
If ∃ ε>0, MAX3SAT(ε) NP-hard then NP=PCP

- We already know **PCP** ⊆ **NP**. Conversely, let L be any language in **NP**.

MAX3SAT(ε)

INPUT: a finite set S of m propositional 3-cla
PROMISE: S satisfiable / opt(S) < (1–ε)m
QUESTION: which is true?

- ∃polytime reduction from L to **MAX3SAT**(ε), since **MAX3SAT**(ε) **NP**-hard by assumption
- **PCP** is closed under polytime reductions (important!)
- So it suffices to exhibit a **PCP** machine deciding **MAX3SAT**(ε)

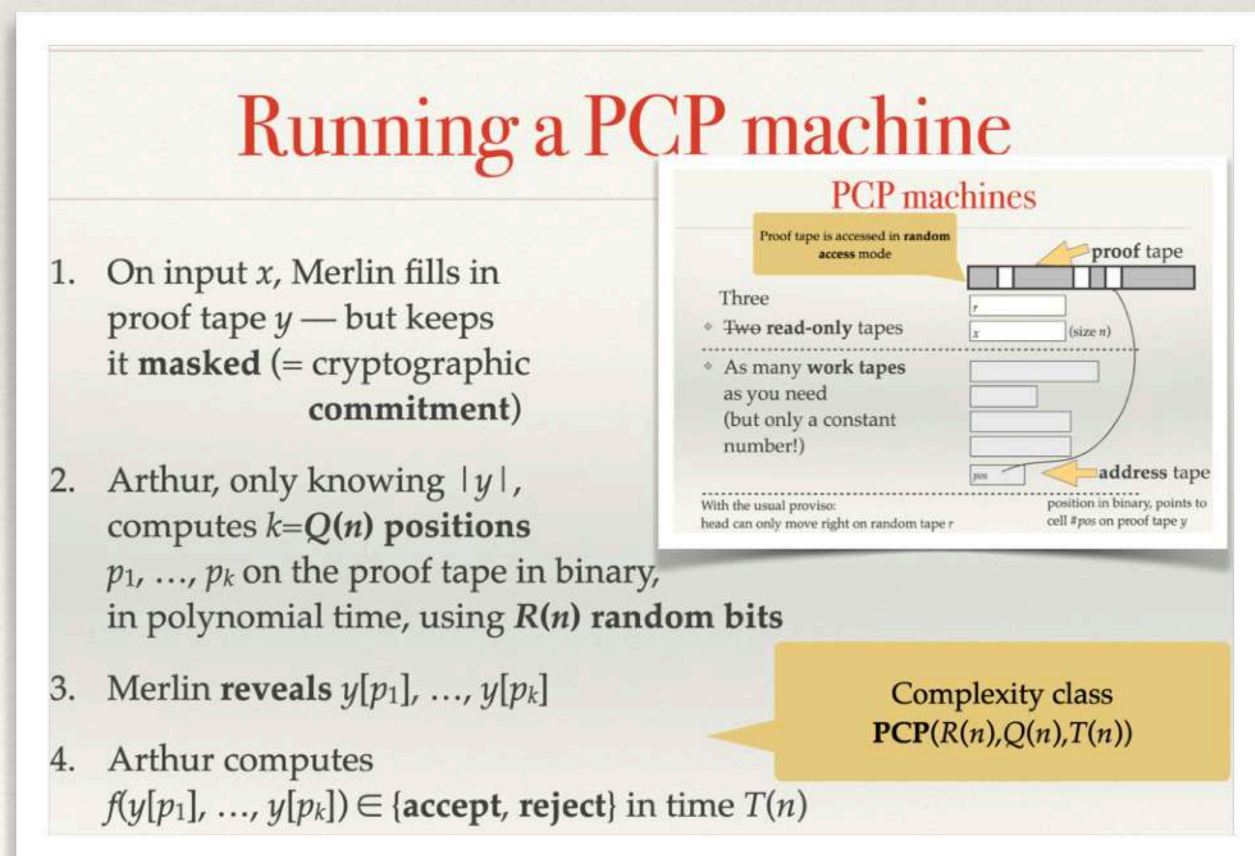


If ∃ ε>0, MAX3SAT(ε) NP-hard then NP=PCP

- A **PCP** machine deciding **MAX3SAT**(ε). Let $S \stackrel{\text{def}}{=} [C_1,\ldots,C_m]$

MAX3SAT(ε)

INPUT: a finite set S of m propositional 3-cla
PROMISE: S satisfiable / opt(S) < (1–ε)m
QUESTION: which is true?



If ∃ ε>0, MAX3SAT(ε) NP-hard then NP=PCP

- A **PCP** machine deciding **MAX3SAT**(ε). Let $S \stackrel{\text{def}}{=} [C_1,\ldots,C_m]$

1. Merlin fills in y with ϱ

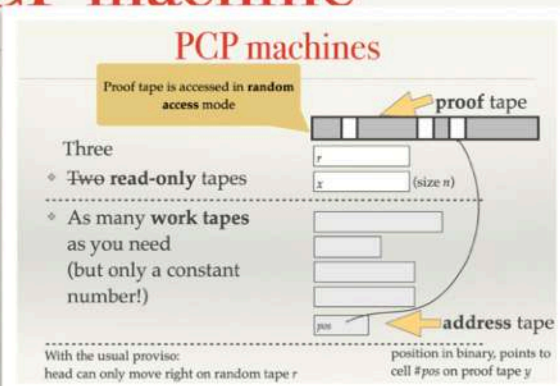
MAX3SAT(ε)

INPUT: a finite set S of m propositional 3-cla
PROMISE: S satisfiable / opt(S) < (1–ε)m
QUESTION: which is true?

Running a PCP machine

1. On input x, Merlin fills in proof tape y — but keeps it **masked** (= cryptographic **commitment**)
2. Arthur, only knowing $|y|$, computes k=**$Q(n)$ positions** $p_1, \ldots, p_k$ on the proof tape in binary, in polynomial time, using **$R(n)$ random bits**
3. Merlin **reveals** $y[p_1], \ldots, y[p_k]$
4. Arthur computes $f(y[p_1], \ldots, y[p_k]) \in$ {**accept**, **reject**} in time $T(n)$

Complexity class
PCP($R(n)$,$Q(n)$,$T(n)$)



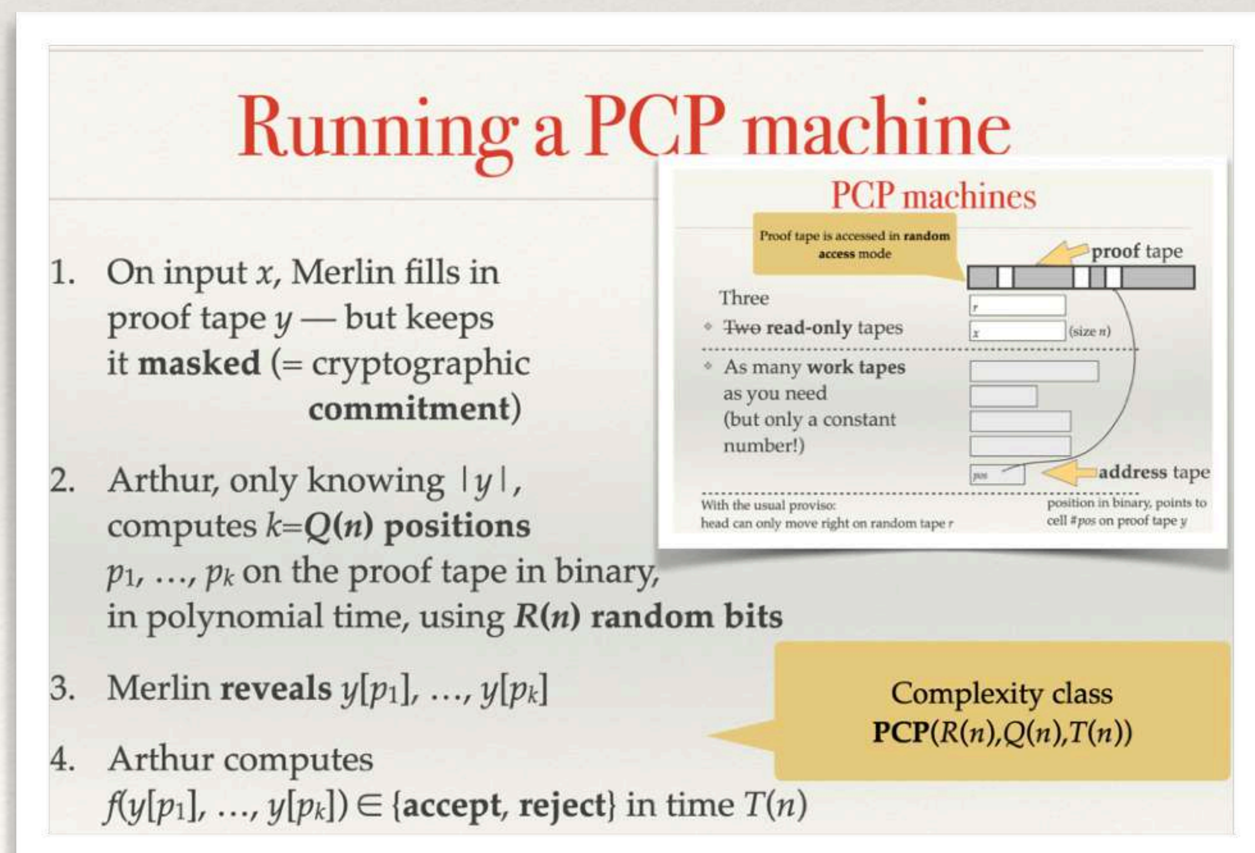
If ∃ ε>0, MAX3SAT(ε) NP-hard then NP=PCP

- A **PCP** machine deciding **MAX3SAT**(ε). Let $S \stackrel{\text{def}}{=} [C_1,\ldots,C_m]$

1. Merlin fills in y with ϱ
2. Arthur chooses C_j at random, (say $+A_{32} \vee -A_{71} \vee -A_{239}$) and gives the corresponding 3 positions (here: 32, 71, 239)

MAX3SAT(ε)

INPUT: a finite set S of m propositional 3-cla
PROMISE: S satisfiable / $\mathrm{opt}(S) < (1-\varepsilon)m$
QUESTION: which is true?



Running a PCP machine

1. On input x, Merlin fills in proof tape y — but keeps it **masked** (= cryptographic **commitment**)
2. Arthur, only knowing $|y|$, computes k=***Q*(*n*) positions** $p_1, \ldots, p_k$ on the proof tape in binary, in polynomial time, using ***R*(*n*) random bits**
3. Merlin **reveals** $y[p_1], \ldots, y[p_k]$
4. Arthur computes $f(y[p_1], \ldots, y[p_k]) \in$ {**accept**, **reject**} in time $T(n)$

Complexity class
PCP($R(n)$,$Q(n)$,$T(n)$)

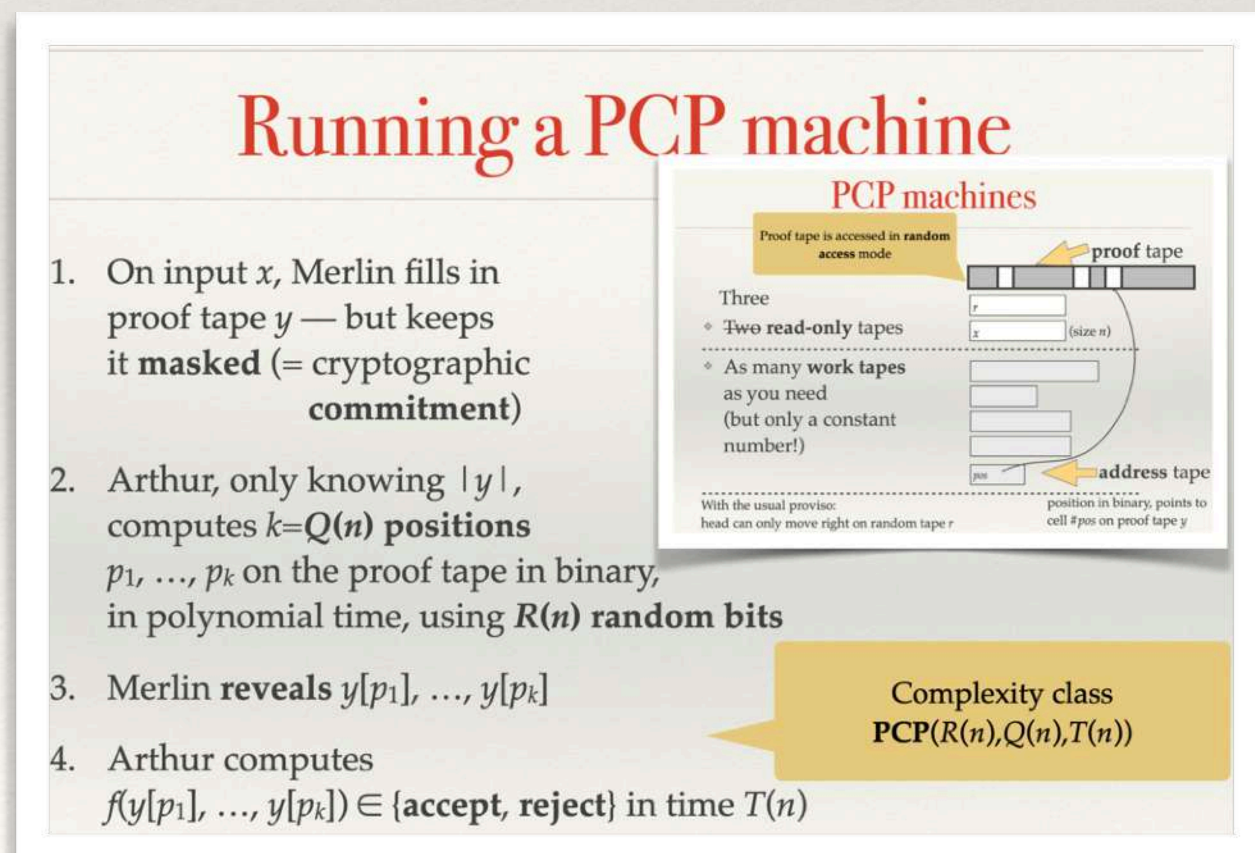
If ∃ ε>0, MAX3SAT(ε) NP-hard then NP=PCP

- A **PCP** machine deciding **MAX3SAT**(ε). Let $S \stackrel{\text{def}}{=} [C_1,\ldots,C_m]$

1. Merlin fills in y with ϱ
2. Arthur chooses C_j at random, (say $+A_{32} \vee -A_{71} \vee -A_{239}$) and gives the corresponding 3 positions (here: 32, 71, 239)
3. Merlin reveals the corresponding truth values

MAX3SAT(ε)

INPUT: a finite set S of m propositional 3-cla
PROMISE: S satisfiable / opt(S) < (1–ε)m
QUESTION: which is true?


Running a PCP machine

1. On input x, Merlin fills in proof tape y — but keeps it **masked** (= cryptographic **commitment**)
2. Arthur, only knowing $|y|$, computes k=$Q(n)$ **positions** $p_1, \ldots, p_k$ on the proof tape in binary, in polynomial time, using **$R(n)$ random bits**
3. Merlin **reveals** $y[p_1], \ldots, y[p_k]$
4. Arthur computes $f(y[p_1], \ldots, y[p_k]) \in$ {**accept**, **reject**} in time $T(n)$

Complexity class **PCP**($R(n)$,$Q(n)$,$T(n)$)

PCP machines

Proof tape is accessed in **random access** mode

proof tape

Three
- ~~Two~~ **read-only** tapes
- As many **work tapes** as you need (but only a constant number!)

r

x (size n)

pos

address tape

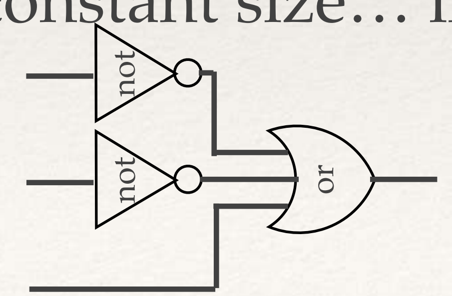
With the usual proviso: head can only move right on random tape r

position in binary, points to cell #pos on proof tape y

If ∃ ε>0, MAX3SAT(ε) NP-hard then NP=PCP

- A **PCP** machine deciding **MAX3SAT**(ε). Let $S \stackrel{\text{def}}{=} [C_1,\ldots,C_m]$

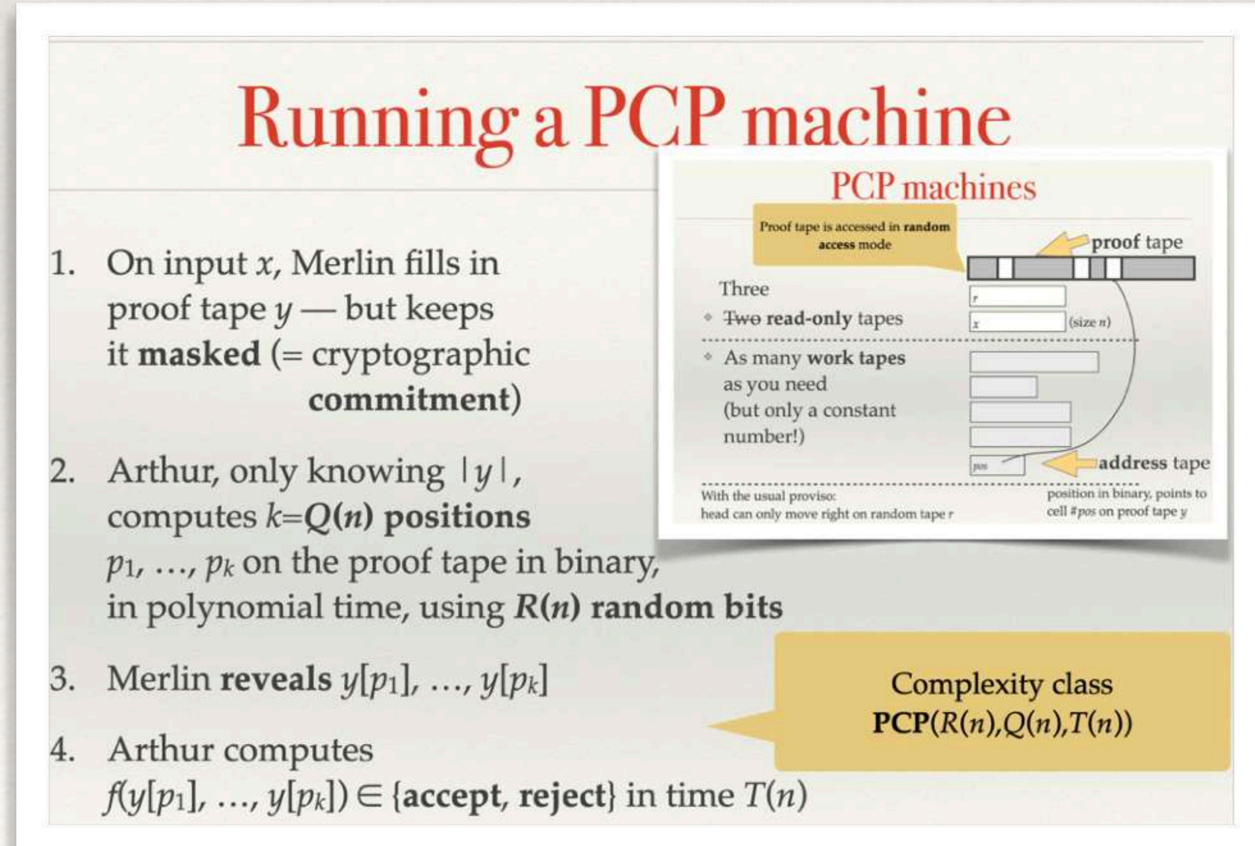
1. Merlin fills in y with ϱ
2. Arthur chooses C_j at random, (say $+A_{32} \vee -A_{71} \vee -A_{239}$) and gives the corresponding 3 positions (here: 32, 71, 239)
3. Merlin reveals the corresponding truth values
4. Arthur evaluates C_j using a precompiled circuit, of constant size… in time O(1) (here,



) accepts if true, rejects if false

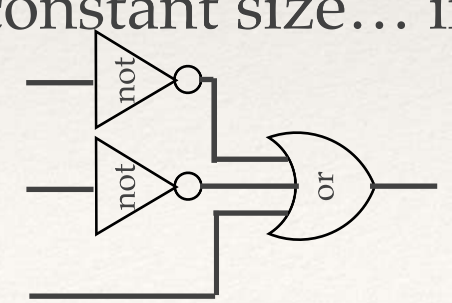
MAX3SAT(ε)

INPUT: a finite set S of m propositional 3-cla
PROMISE: S satisfiable / opt(S) < $(1-\varepsilon)m$
QUESTION: which is true?



If ∃ ε>0, MAX3SAT(ε) NP-hard then NP=PCP

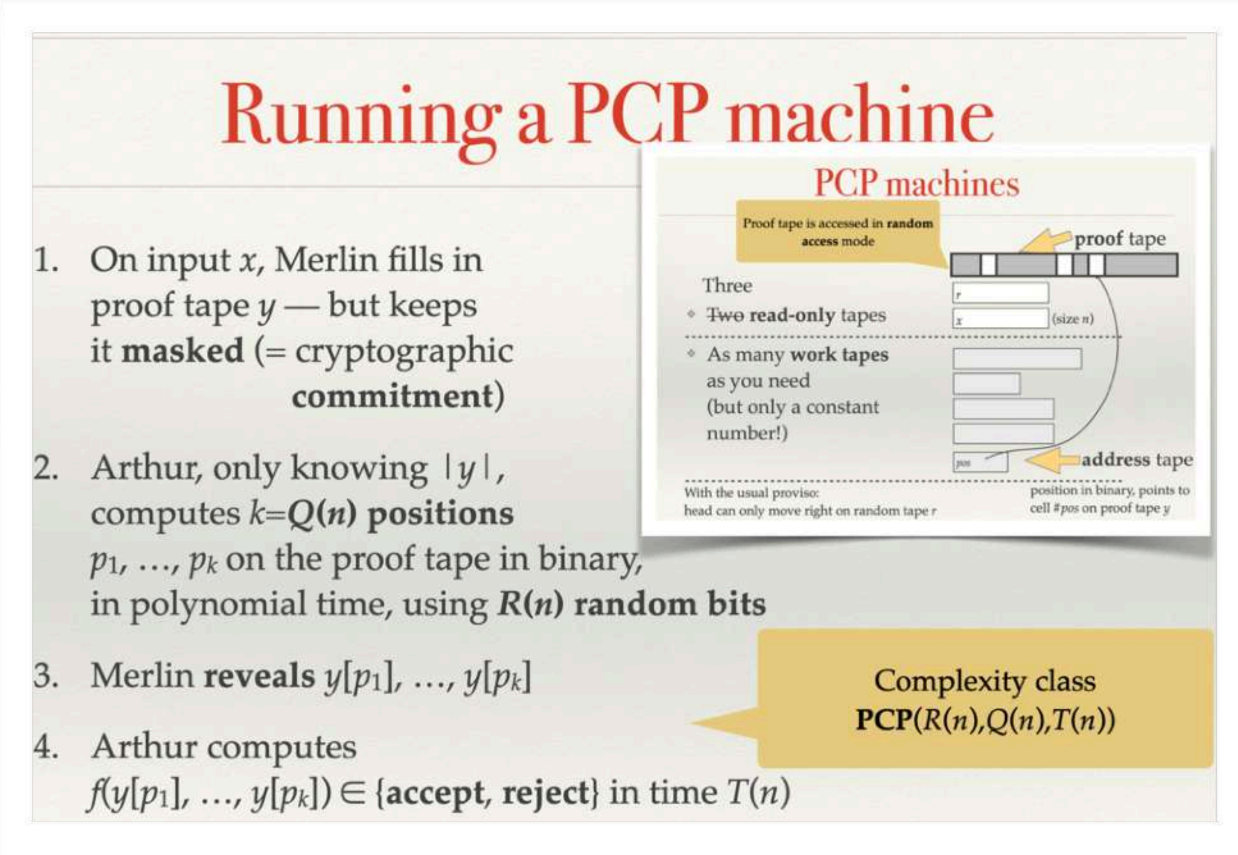
- A **PCP** machine deciding **MAX3SAT**(ε). Let $S \stackrel{\text{def}}{=} [C_1,\ldots,C_m]$

1. Merlin fills in y with ϱ
2. Arthur chooses C_j at random, (say $+A_{32} \vee -A_{71} \vee -A_{239}$) and gives the corresponding 3 positions (here: 32, 71, 239)
3. Merlin reveals the corresponding truth values
4. Arthur evaluates C_j using a precompiled circuit, of constant size… in time O(1) (here, ) accepts if true, rejects if false

MAX3SAT(ε)

INPUT: a finite set S of m propositional 3-cla
PROMISE: S satisfiable / opt(S) < $(1-\varepsilon)m$
QUESTION: which is true?

Oops… and precompiles a circuit that evaluates C_j, to be used in step 4



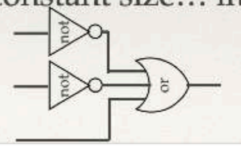
If ∃ $\varepsilon>0$, MAX3SAT(ε) NP-hard then NP=PCP

- Uses $R(n)$=O(log n) random bits:
  just one number j ($1 \leq j \leq m$) at random
- $Q(n)$=O(1) (indeed, =3)
  $T(n)$=O(1)

MAX3SAT(ε)

INPUT: a finite set S of m propositional 3-cla
PROMISE: S satisfiable / opt(S) < $(1-\varepsilon)m$
QUESTION: which is true?

- A **PCP** machine deciding **MAX3SAT**(ε). Let $S \triangleq [C_1,\ldots,C_m]$

1. Merlin fills in y with ϱ
2. Arthur chooses C_j at random, (say $+A_{32} \vee -A_{71} \vee -A_{239}$) and gives the corresponding 3 positions (here: 32, 71, 239)
3. Merlin reveals the corresponding truth values
4. Arthur decides using a precompiled circuit, of constant size… in time O(1) (here, )

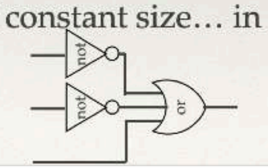
If ∃ ε>0, MAX3SAT(ε) NP-hard then NP=PCP

- Uses $R(n)$=O(log n) random bits:
  just one number j ($1 \le j \le m$) at random
- $Q(n)$=O(1) (indeed, =3)
  $T(n)$=O(1)
- If S satisfiable, then Merlin can produce
  a satisfying assignment, so Arthur will accept

MAX3SAT(ε)

INPUT: a finite set S of m propositional 3-cla
PROMISE: S satisfiable / opt(S) < $(1-\varepsilon)m$
QUESTION: which is true?

- A **PCP** machine deciding **MAX3SAT**(ε). Let $S \stackrel{\text{def}}{=} [C_1,\ldots,C_m]$

1. Merlin fills in y with ϱ
2. Arthur chooses C_j at random, (say $+A_{32} \vee -A_{71} \vee -A_{239}$) and gives the corresponding 3 positions (here: 32, 71, 239)
3. Merlin reveals the corresponding truth values
4. Arthur decides using a precompiled circuit, of constant size… in time O(1) (here, )

If ∃ $\varepsilon>0$, MAX3SAT(ε) NP-hard then NP=PCP

- Uses $R(n)$=O(log n) random bits:
just one number j ($1 \le j \le m$) at random

- $Q(n)$=O(1) (indeed, =3)
$T(n)$=O(1)

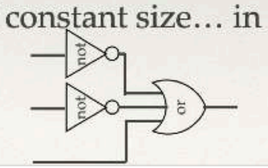
- If S satisfiable, then Merlin can produce a satisfying assignment, so Arthur will accept

- If opt(S) < $(1-\varepsilon)m$, then whatever ϱ is given,
$\Pr_j(\varrho \vDash C_j) < (1-\varepsilon)$

MAX3SAT(ε)

INPUT: a finite set S of m propositional 3-cla
PROMISE: S satisfiable / opt(S) < $(1-\varepsilon)m$
QUESTION: which is true?

- A **PCP** machine deciding **MAX3SAT**(ε). Let $S \triangleq [C_1,\ldots,C_m]$

1. Merlin fills in y with ϱ
2. Arthur chooses C_j at random, (say $+A_{32} \vee -A_{71} \vee -A_{239}$) and gives the corresponding 3 positions (here: 32, 71, 239)
3. Merlin reveals the corresponding truth values
4. Arthur decides using a precompiled circuit, of constant size… in time O(1) (here, )

If ∃ ε>0, MAX3SAT(ε) NP-hard then NP=PCP

- Uses $R(n)$=O(log n) random bits:
  just one number j ($1 \leq j \leq m$) at random

- $Q(n)$=O(1) (indeed, =3)
  $T(n)$=O(1)

- If S satisfiable, then Merlin can produce
  a satisfying assignment, so Arthur will accept

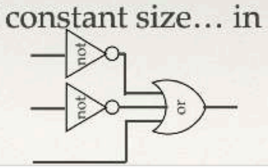
- If opt(S) < (1–ε)m, then whatever ϱ is given,
  $\Pr_j(\varrho \models C_j) < (1-\varepsilon)$

Note: j is random here,
not ϱ as in Johnson's algorithm

MAX3SAT(ε)

INPUT: a finite set S of m propositional 3-cla
PROMISE: S satisfiable / opt(S) < (1–ε)m
QUESTION: which is true?

- A **PCP** machine deciding **MAX3SAT**(ε). Let $S \triangleq [C_1,\ldots,C_m]$

1. Merlin fills in y with ϱ
2. Arthur chooses C_j at random, (say $+A_{32} \vee -A_{71} \vee -A_{239}$) and gives the corresponding 3 positions (here: 32, 71, 239)
3. Merlin reveals the corresponding truth values
4. Arthur decides using a precompiled circuit, of constant size… in time O(1) (here, )

If ∃ ε>0, MAX3SAT(ε) NP-hard then NP=PCP

- Uses $R(n)$=O(log n) random bits:
  just one number j ($1 \leq j \leq m$) at random

- $Q(n)$=O(1) (indeed, =3)
  $T(n)$=O(1)

- If S satisfiable, then Merlin can produce a satisfying assignment, so Arthur will accept

- If opt(S) < (1–ε)m, then whatever ϱ is given,
  $\Pr_j(\varrho \vDash C_j) < (1-\varepsilon)$

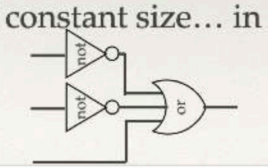
Shoot! We needed ½ here…

Note: j is random here,
not ϱ as in Johnson's algorithm

MAX3SAT(ε)

INPUT: a finite set S of m propositional 3-cla
PROMISE: S satisfiable / opt(S) < (1–ε)m
QUESTION: which is true?

- A **PCP** machine deciding **MAX3SAT**(ε). Let $S \triangleq [C_1,\ldots,C_m]$

1. Merlin fills in y with ϱ
2. Arthur chooses C_j at random, (say $+A_{32} \vee -A_{71} \vee -A_{239}$) and gives the corresponding 3 positions (here: 32, 71, 239)
3. Merlin reveals the corresponding truth values
4. Arthur decides using a precompiled circuit, of constant size… in time O(1) (here, 

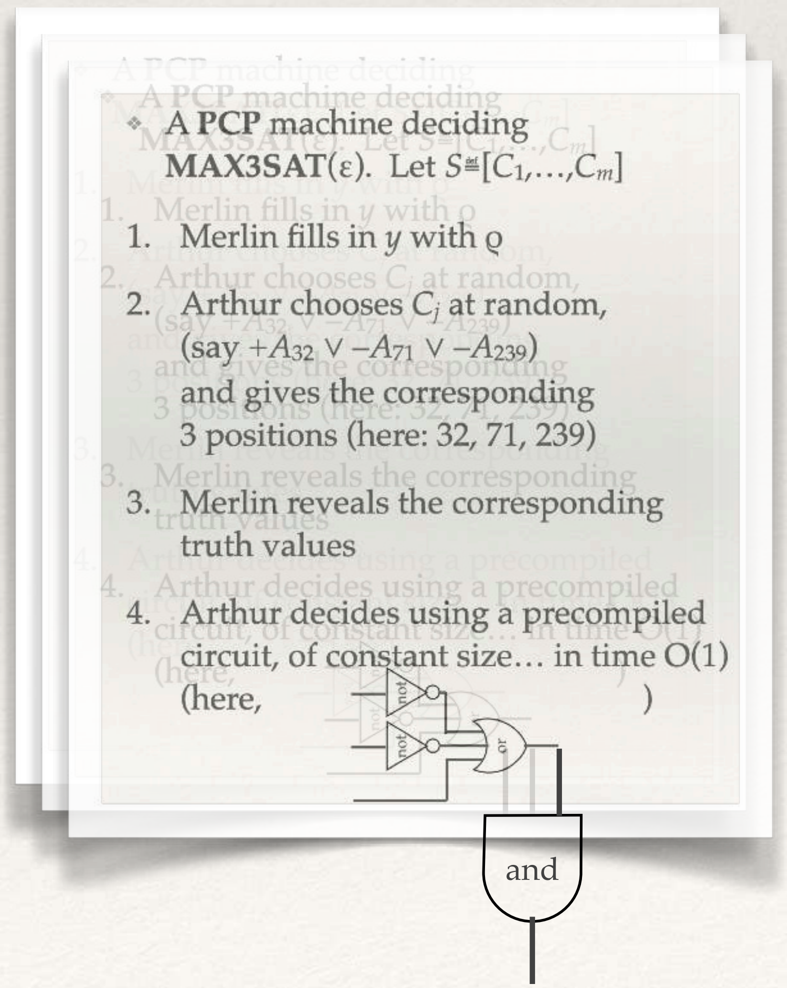
)

If ∃ ε>0, MAX3SAT(ε) NP-hard then NP=PCP

- We solve the problem using **parallel repetition** (*k* times)

MAX3SAT(ε)

INPUT: a finite set S of m propositional 3-cla
PROMISE: S satisfiable / opt(S) < (1–ε)m
QUESTION: which is true?

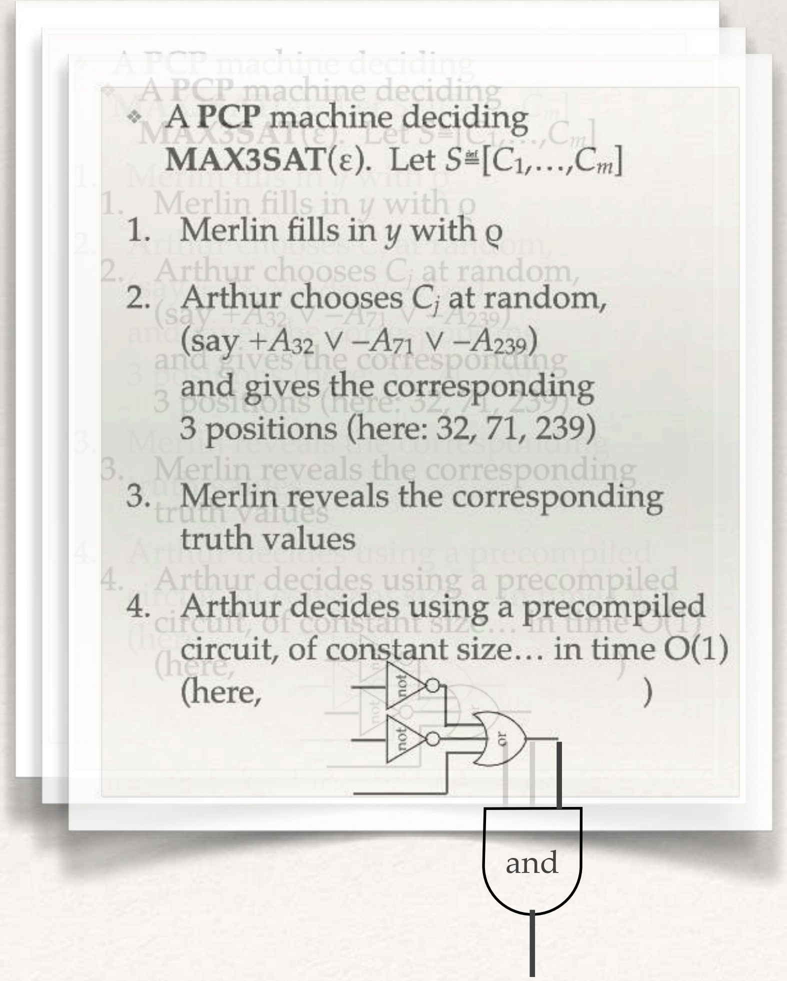


If ∃ ε>0, MAX3SAT(ε) NP-hard then NP=PCP

- We solve the problem using **parallel repetition** (k times)
- Uses $R(n)$=O(log n) random bits: just k numbers j ($1 \leq j \leq m$) at random

MAX3SAT(ε)

INPUT: a finite set S of m propositional 3-cla
PROMISE: S satisfiable / opt(S) < $(1-\varepsilon)m$
QUESTION: which is true?



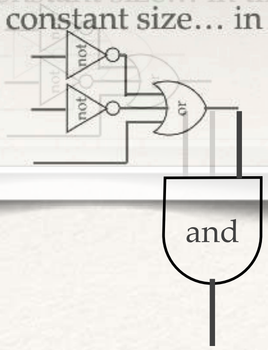
If ∃ ε>0, MAX3SAT(ε) NP-hard then NP=PCP

- We solve the problem using **parallel repetition** (k times)
- Uses $R(n)$=O(log n) random bits: just k numbers j ($1 \leq j \leq m$) at random
- $Q(n)$=O(1) (indeed, $3k$)
  $T(n)$=O(1)

MAX3SAT(ε)

INPUT: a finite set S of m propositional 3-cla
PROMISE: S satisfiable / opt(S) < $(1-\varepsilon)m$
QUESTION: which is true?

- A PCP machine deciding MAX3SAT(ε). Let $S \stackrel{\text{def}}{=} [C_1,\dots,C_m]$
1. Merlin fills in y with ϱ
2. Arthur chooses C_j at random, (say $+A_{32} \vee -A_{71} \vee -A_{239}$) and gives the corresponding 3 positions (here: 32, 71, 239)
3. Merlin reveals the corresponding truth values
4. Arthur decides using a precompiled circuit, of constant size… in time O(1) (here,)

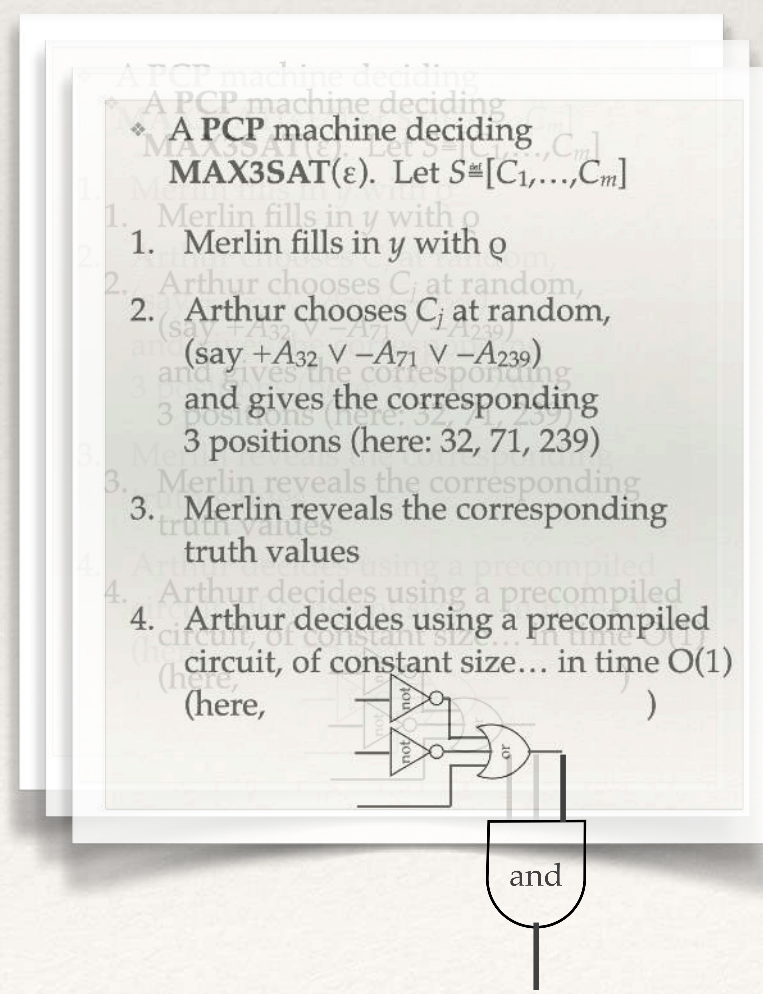


If ∃ ε>0, MAX3SAT(ε) NP-hard then NP=PCP

- We solve the problem using **parallel repetition** (k times)
- Uses $R(n)$=O(log n) random bits: just k numbers j ($1 \leq j \leq m$) at random
- $Q(n)$=O(1) (indeed, $3k$) $T(n)$=O(1)
- If S satisfiable, then Merlin can produce a satisfying assignment, so Arthur will accept

MAX3SAT(ε)

INPUT: a finite set S of m propositional 3-cla
PROMISE: S satisfiable / opt(S) < (1–ε)m
QUESTION: which is true?



If ∃ ε>0, MAX3SAT(ε) NP-hard then NP=PCP

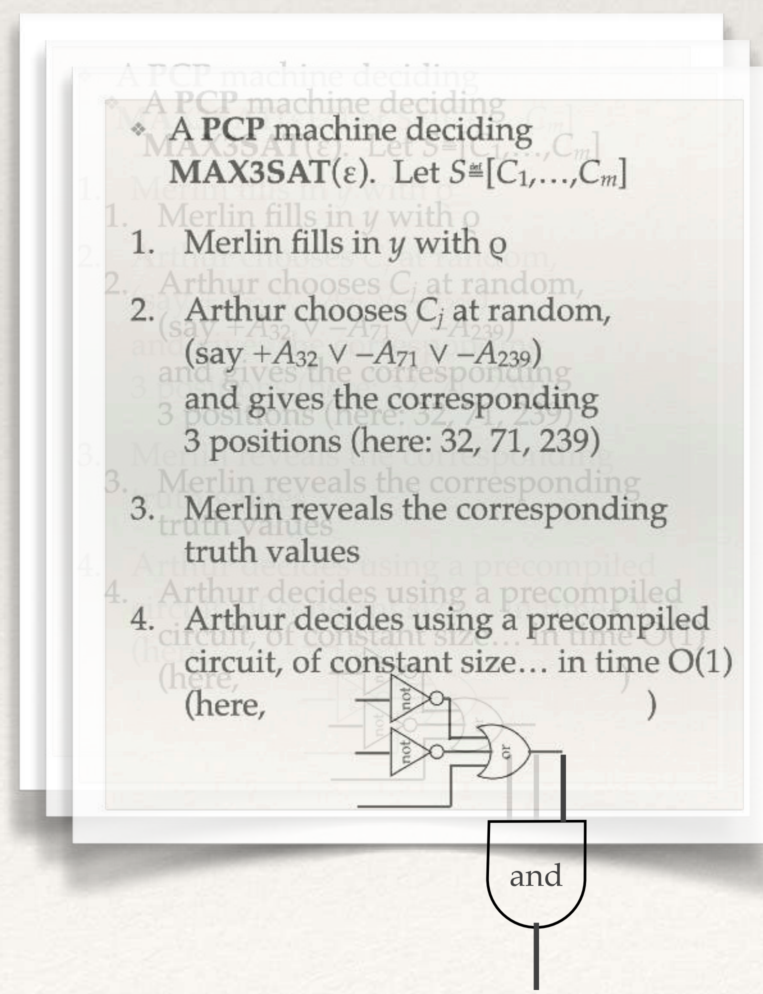
- We solve the problem using **parallel repetition** (k times)
- Uses $R(n)$=O(log n) random bits: just k numbers j ($1 \le j \le m$) at random
- $Q(n)$=O(1) (indeed, $3k$) $T(n)$=O(1)
- If S satisfiable, then Merlin can produce a satisfying assignment, so Arthur will accept
- If opt(S) < $(1-\varepsilon)m$, then whatever ϱ is given, $\Pr_{j_1,\ldots,j_k}(\varrho \models C_{j_1}$ and … and $\varrho \models C_{j_k}) \le (1-\varepsilon)^k$

MAX3SAT(ε)

INPUT: a finite set S of m propositional 3-cla

PROMISE: S satisfiable / opt(S) < $(1-\varepsilon)m$

QUESTION: which is true?



- A PCP machine deciding MAX3SAT(ε). Let $S \triangleq [C_1,\ldots,C_m]$

1. Merlin fills in y with ϱ
2. Arthur chooses C_j at random, (say $+A_{32} \vee -A_{71} \vee -A_{239}$) and gives the corresponding 3 positions (here: 32, 71, 239)
3. Merlin reveals the corresponding truth values
4. Arthur decides using a precompiled circuit, of constant size… in time O(1) (here,)

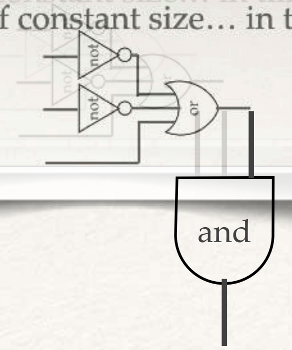
If ∃ ε>0, MAX3SAT(ε) NP-hard then NP=PCP

- We solve the problem using **parallel repetition** (k times)
  $k \stackrel{\text{def}}{=} \lceil -\log 2 / \log (1-\varepsilon) \rceil$
- Uses $R(n)=O(\log n)$ random bits:
  just k numbers j ($1 \leq j \leq m$) at random
- $Q(n)=O(1)$ (indeed, $3k$)
  $T(n)=O(1)$
- If S satisfiable, then Merlin can produce a satisfying assignment, so Arthur will accept
- If $\text{opt}(S) < (1-\varepsilon)m$, then whatever ϱ is given,
  $\Pr_{j_1,\ldots,j_k}(\varrho \vDash C_{j_1} \text{ and } \ldots \text{ and } \varrho \vDash C_{j_k}) \leq (1-\varepsilon)^k \leq ½$

MAX3SAT(ε)

INPUT: a finite set S of m propositional 3-cla
PROMISE: S satisfiable / $\text{opt}(S) < (1-\varepsilon)m$
QUESTION: which is true?

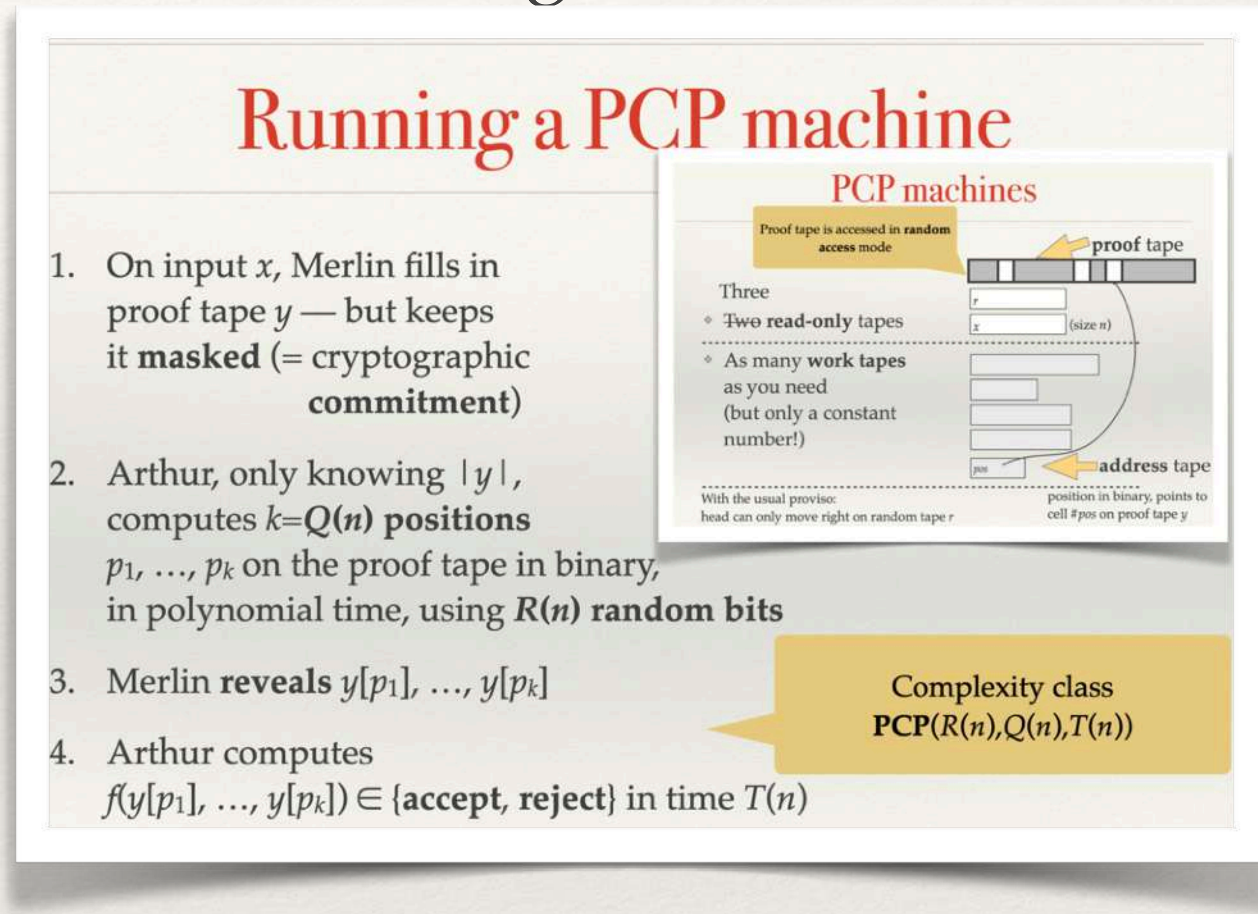
- A PCP machine deciding MAX3SAT(ε). Let $S \stackrel{\text{def}}{=} [C_1,\ldots,C_m]$
  1. Merlin fills in y with ϱ
  2. Arthur chooses C_j at random, (say $+A_{32} \vee -A_{71} \vee -A_{239}$) and gives the corresponding 3 positions (here: 32, 71, 239)
  3. Merlin reveals the corresponding truth values
  4. Arthur decides using a precompiled circuit, of constant size… in time O(1) (here,)



$(\exists\, \varepsilon>0$, MAX3SAT(ε) **NP**-hard$)$ iff **NP**=**PCP**: the right to left direction

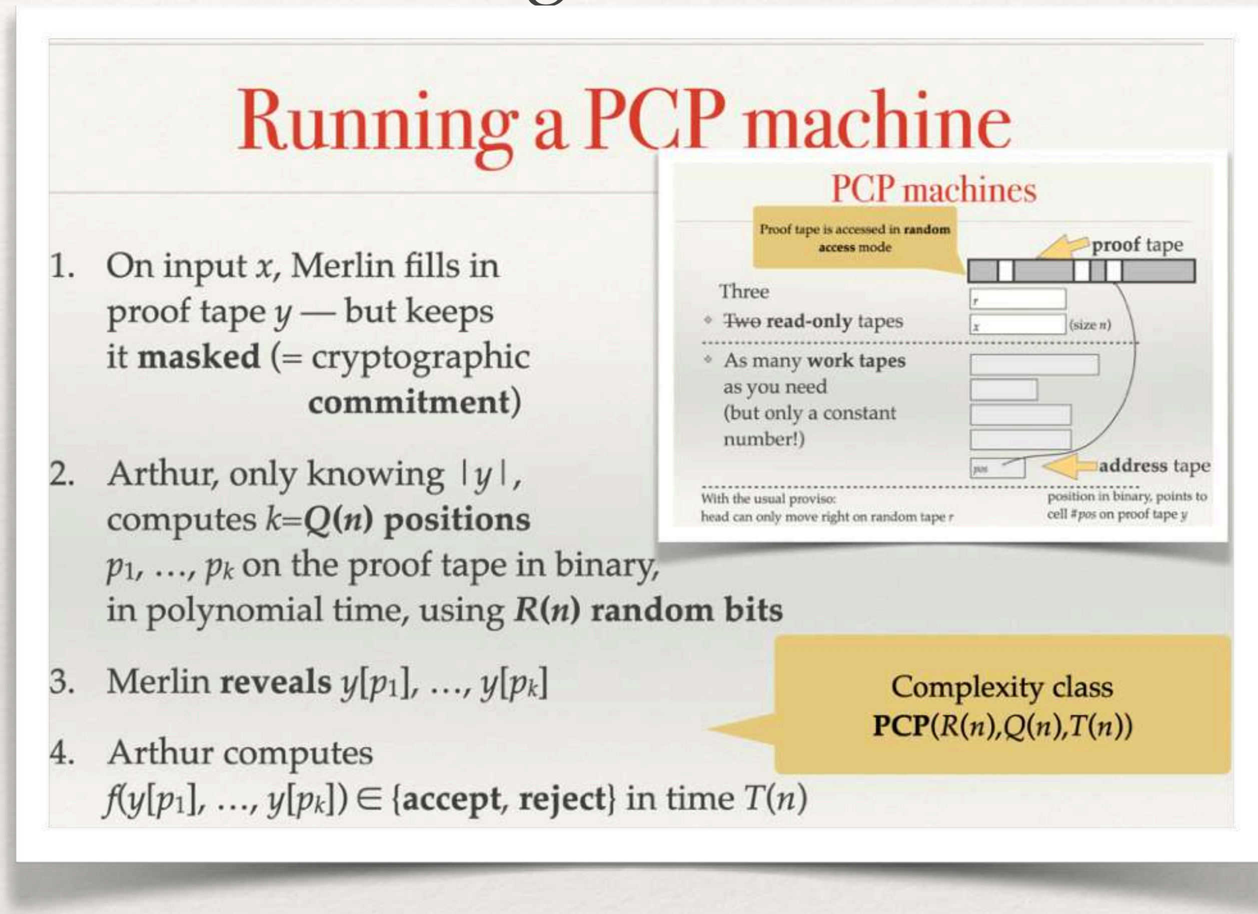
If NP=PCP then ∃ ε>0, MAX3SAT(ε) NP-hard

- Assume **NP=PCP**. Then there is a **PCP**($R(n) \stackrel{\text{def}}{=} k \log n$,O(1),O(1)) machine $\mathcal{M}$ deciding **SAT**



If NP=PCP then ∃ ε>0, MAX3SAT(ε) NP-hard

- Assume **NP**=**PCP**. Then there is a **PCP**($R(n) \stackrel{\text{def}}{=} k \log n$,O(1),O(1)) machine $\mathcal{M}$ deciding **SAT**
- We look for a polytime reduction from **SAT** to **MAX3SAT**(ε), for some ε>0

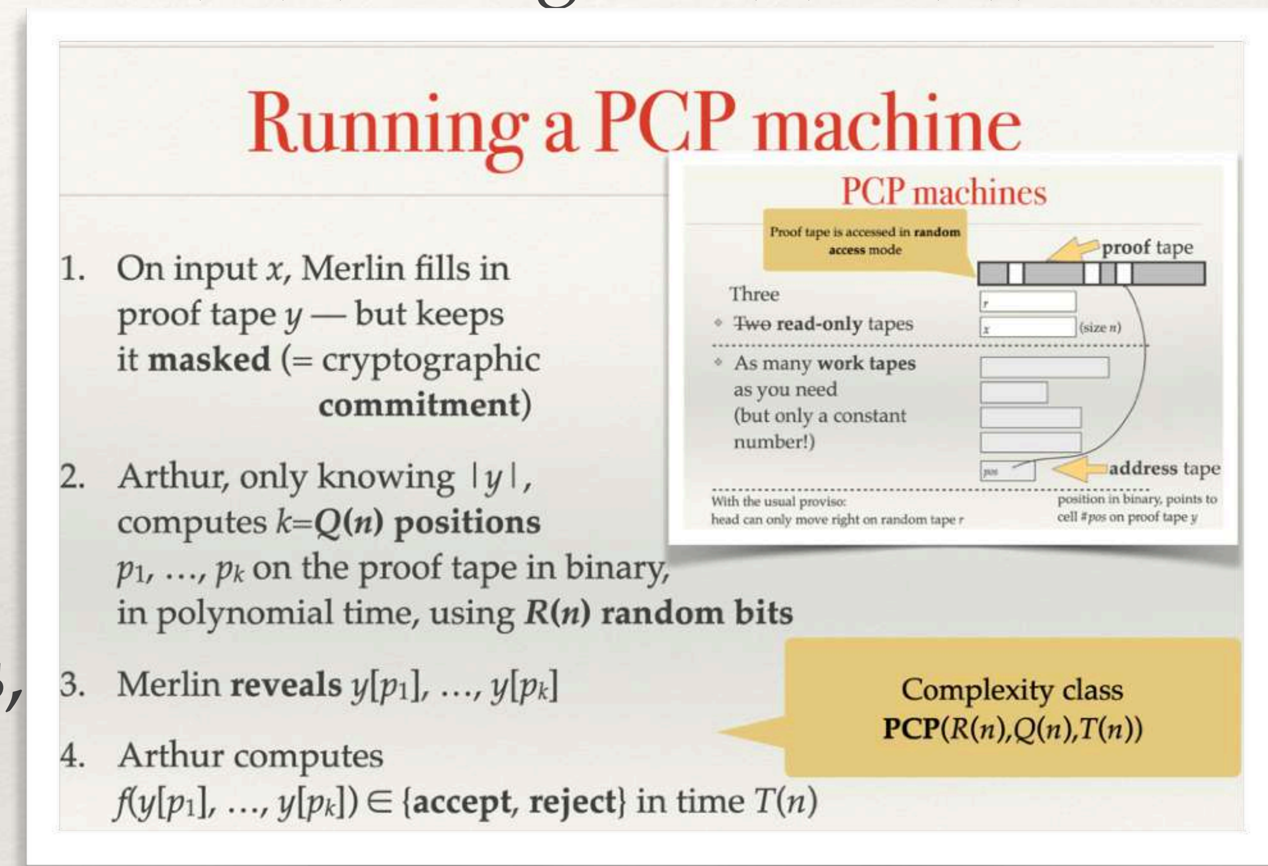


If NP=PCP then ∃ ε>0, MAX3SAT(ε) NP-hard

- Assume **NP**=**PCP**. Then there is a **PCP**($R(n) \stackrel{\text{def}}{=} k \log n$,O(1),O(1)) machine $\mathcal{M}$ deciding **SAT**

- We look for a polytime reduction from **SAT** to **MAX3SAT**(ε), for some ε>0

- Let us look at $\mathcal{M}(x)$'s possible runs, for each $R(n)$-bit word r drawn at random in step 2

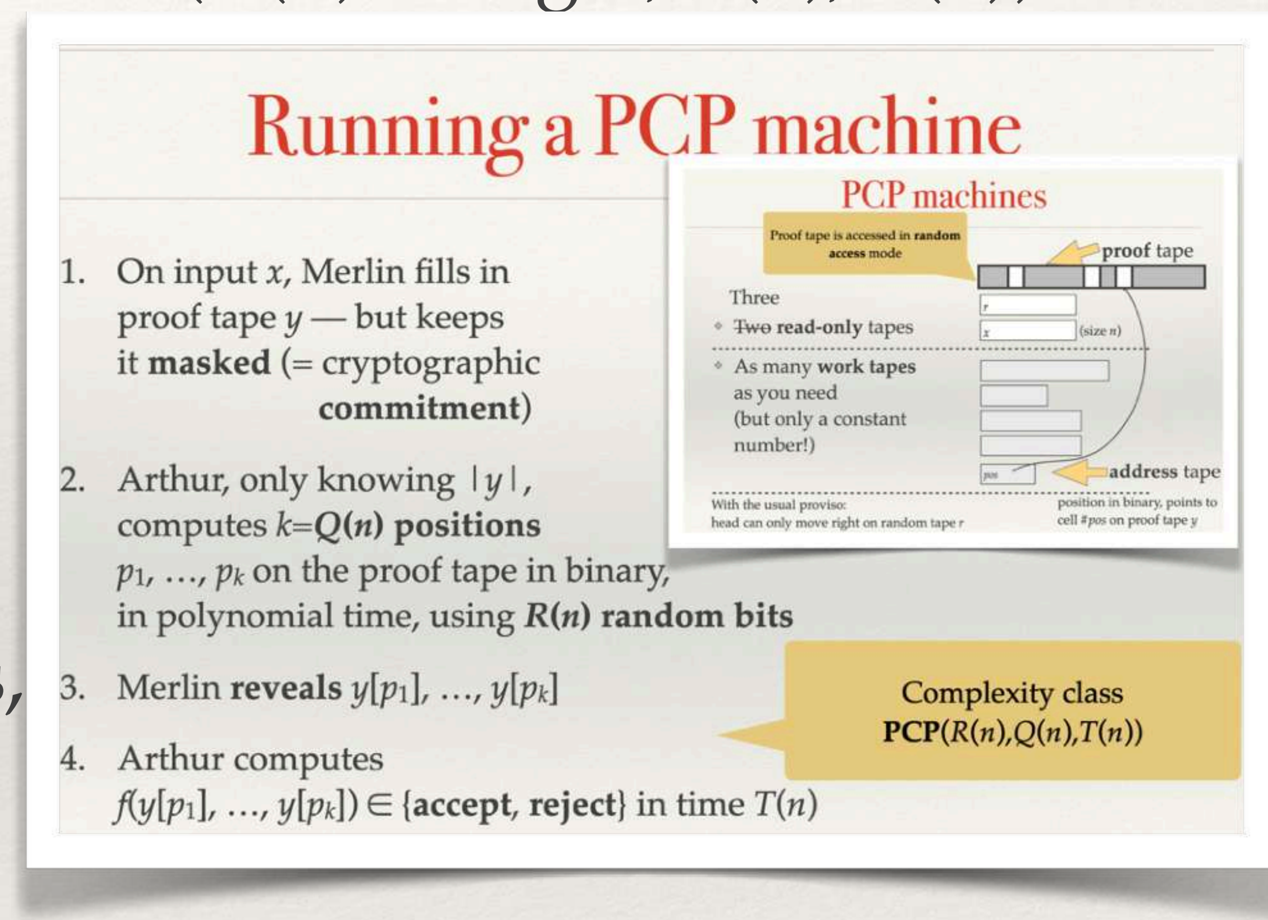


If NP=PCP then ∃ ε>0, MAX3SAT(ε) NP-hard

- Assume **NP**=**PCP**. Then there is a **PCP**($R(n) \stackrel{\text{def}}{=} k \log n$,O(1),O(1)) machine $\mathcal{M}$ deciding **SAT**

- We look for a polytime reduction from **SAT** to **MAX3SAT**(ε), for some ε>0

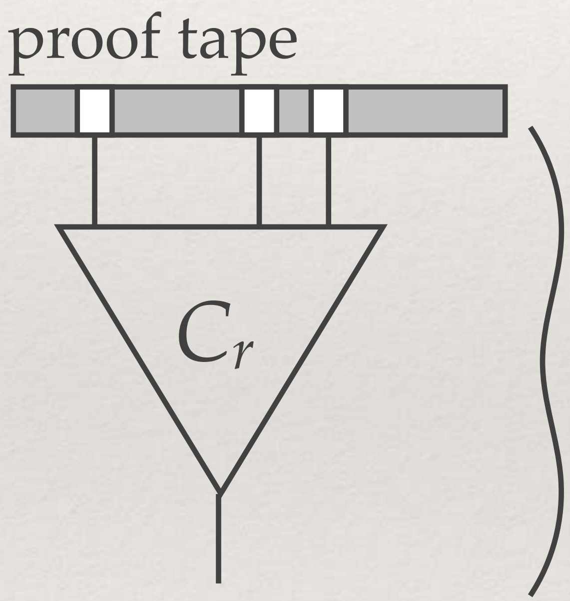
- Let us look at $\mathcal{M}(x)$'s possible runs, for each $R(n)$-bit word r drawn at random in step 2



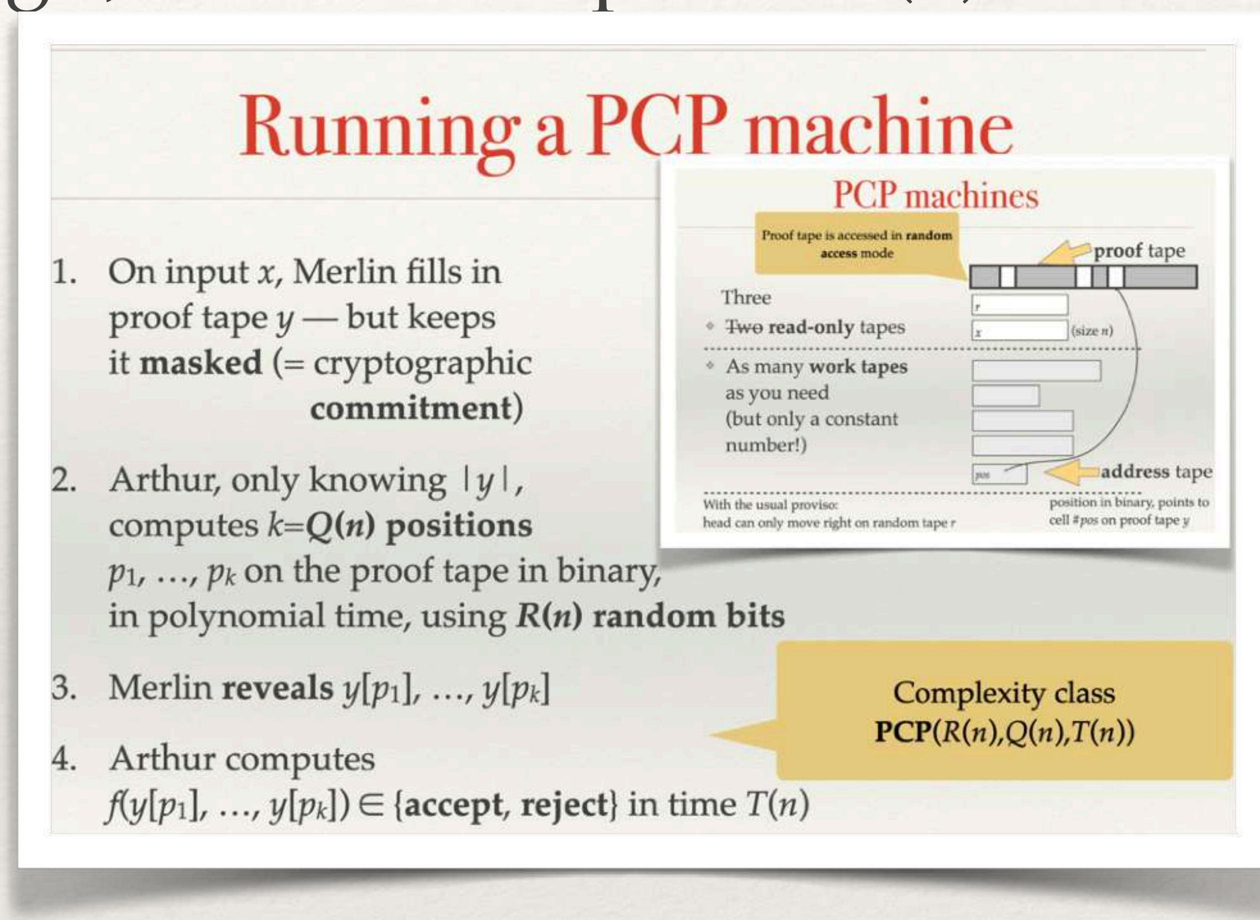
- (Note that the assumption that $T(n)$=O(1) in step 4 is superfluous:
  f only has a constant #inputs, and can always be encoded by a constant-size circuit, evaluated in time O(1)…)

If NP=PCP then ∃ ε>0, MAX3SAT(ε) NP-hard

- For each $k \log_2 n$-bit random string r, Arthur computes O(1) positions, and a constant-size circuit C_r (say, ≤G fan-in 2 gates)



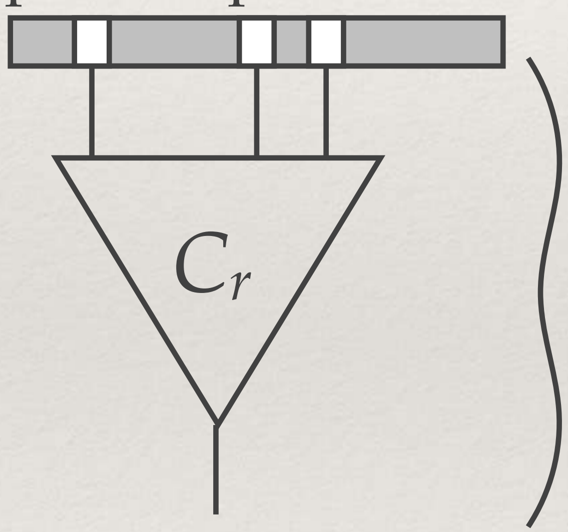
$2^{k \log_2 n} = n^k$ instances



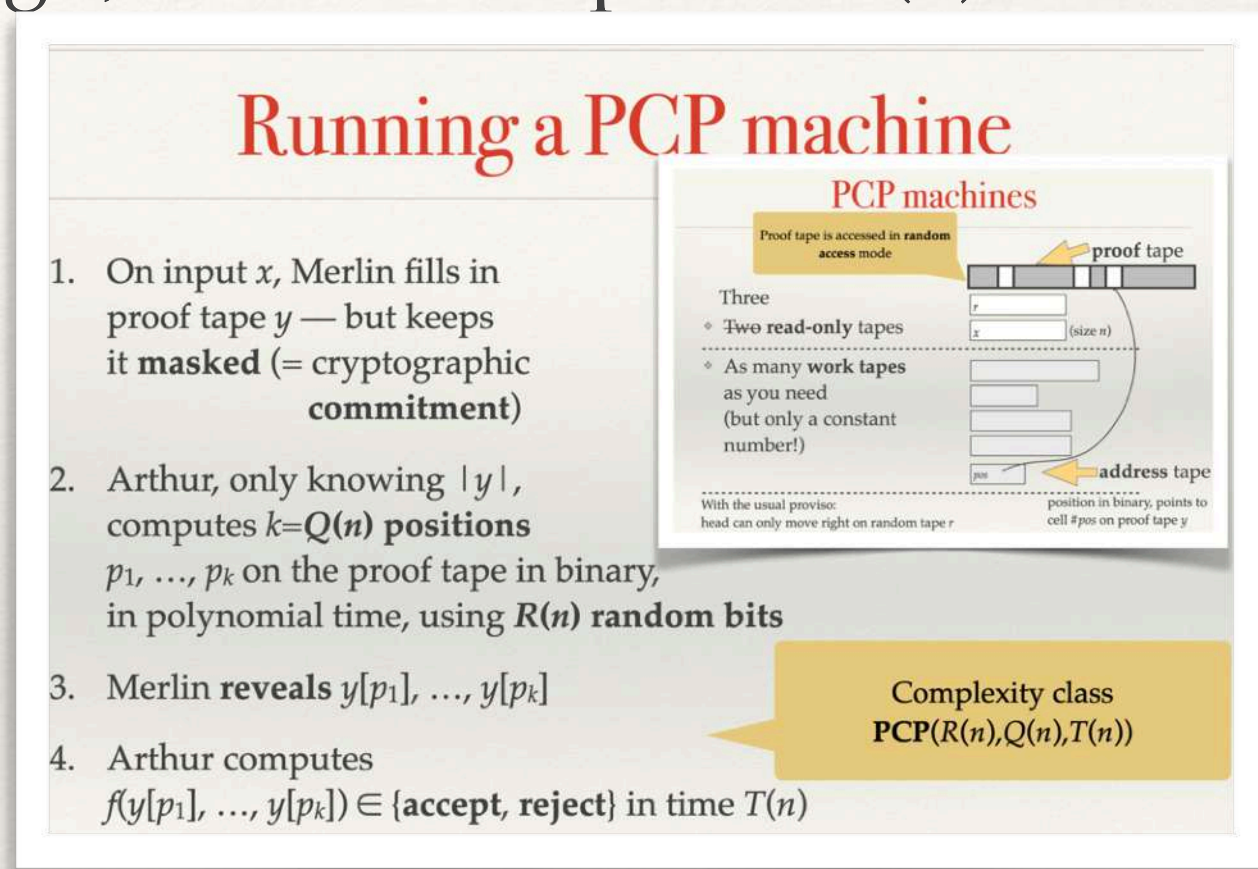
If NP=PCP then ∃ ε>0, MAX3SAT(ε) NP-hard

- For each $k \log_2 n$-bit random string r, Arthur computes O(1) positions, and a constant-size circuit C_r (say, $\leq G$ fan-in 2 gates)

proof tape



$2^{k \log_2 n} = n^k$ instances



Running a PCP machine

1. On input x, Merlin fills in proof tape y — but keeps it **masked** (= cryptographic **commitment**)
2. Arthur, only knowing $|y|$, computes k=$\mathbf{Q(n)}$ **positions** $p_1, \ldots, p_k$ on the proof tape in binary, in polynomial time, using $\mathbf{R(n)}$ **random bits**
3. Merlin **reveals** $y[p_1], \ldots, y[p_k]$
4. Arthur computes $f(y[p_1], \ldots, y[p_k]) \in \{$**accept**, **reject**$\}$ in time $T(n)$

Complexity class **PCP**$(R(n),Q(n),T(n))$

PCP machines

Proof tape is accessed in **random access** mode

proof tape

Three
- ~~Two~~ **read-only** tapes

r

x (size n)

- As many **work tapes** as you need (but only a constant number!)

pos **address** tape

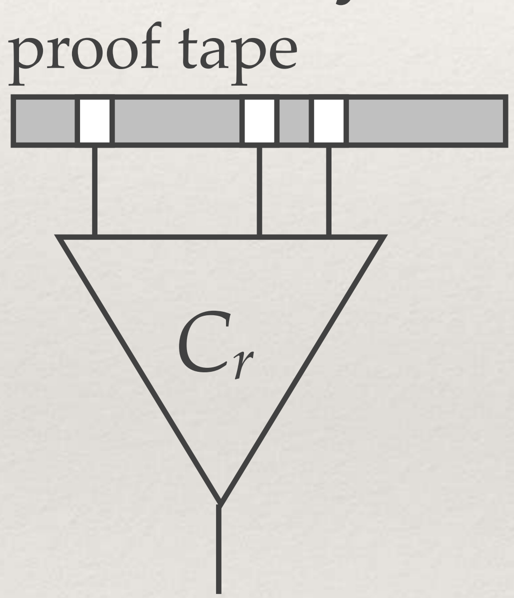
With the usual proviso: head can only move right on random tape r

position in binary, points to cell #pos on proof tape y

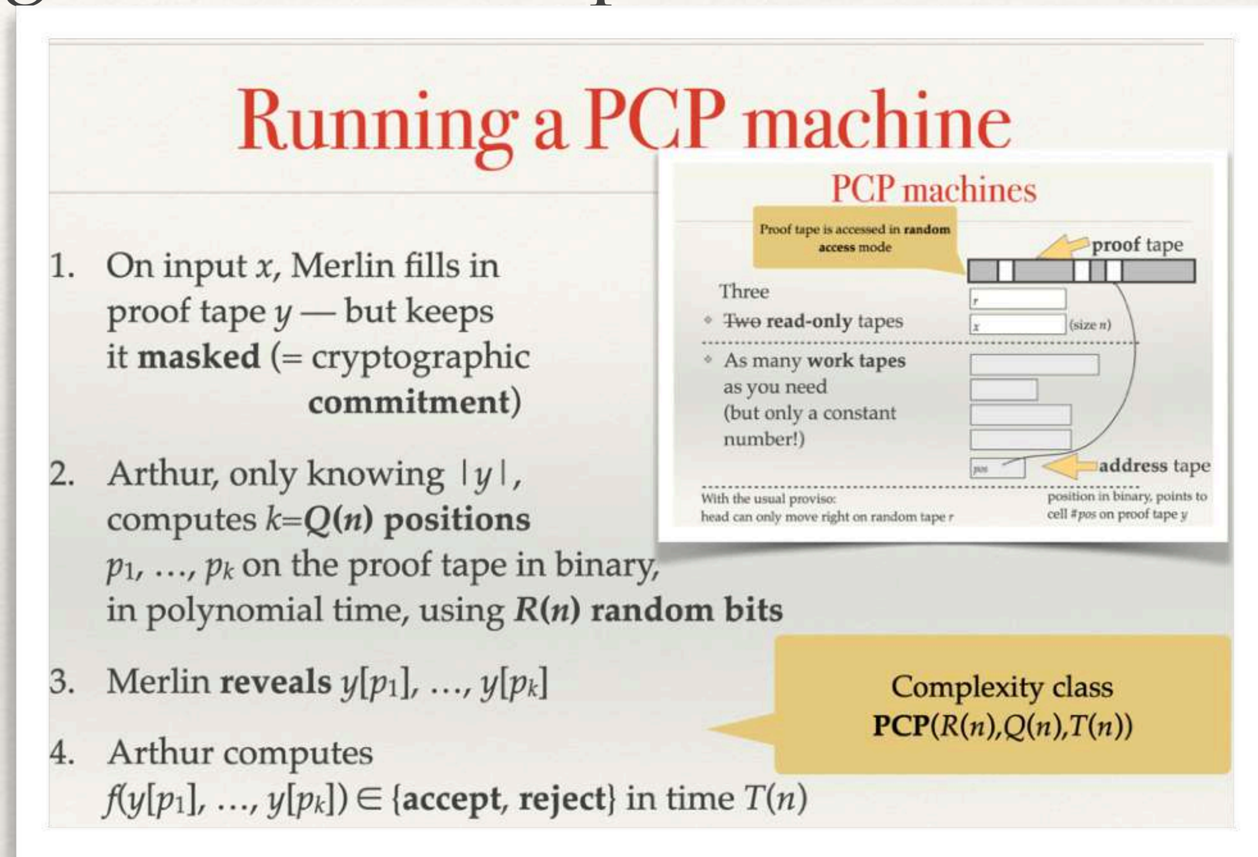
- If input S satisfiable, then Merlin can provide y / output wire of C_r is true, for every r

If NP=PCP then ∃ ε>0, MAX3SAT(ε) NP-hard

- For each $k \log_2 n$-bit random string r, Arthur computes O(1) positions, and a constant-size circuit C_r (say, ≤G fan-in 2 gates)



$2^{k \log_2 n} = n^k$ instances

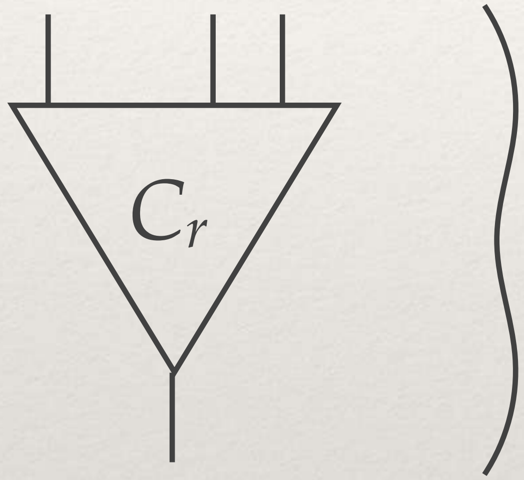


- If input S satisfiable, then Merlin can provide y / output wire of C_r is true, for every r
- Otherwise, whatever y, ≥ ½n^k output wires false

If NP=PCP then ∃ ε>0, MAX3SAT(ε) NP-hard

- We now encode those circuits as a **3SAT** formula, e.g.:

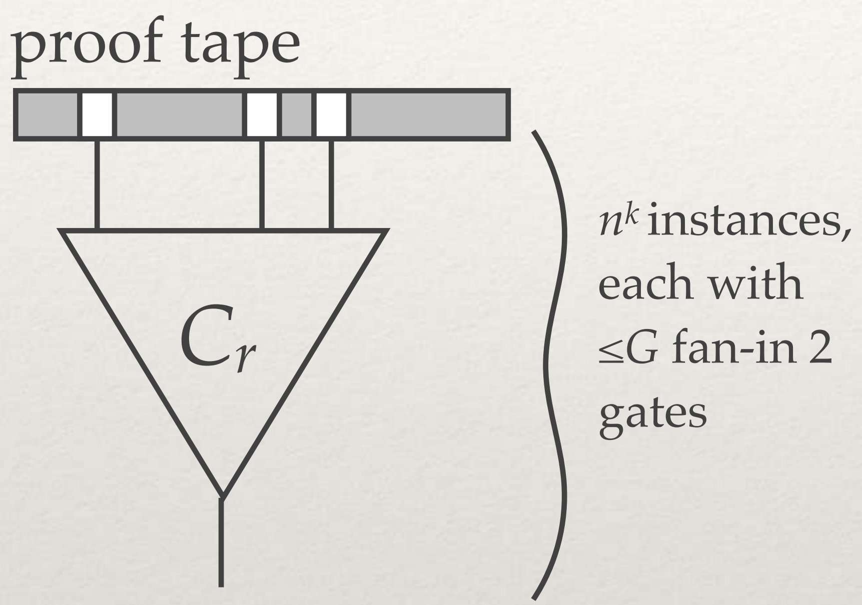
proof tape



n^k instances,
each with
$\leq G$ fan-in 2
gates

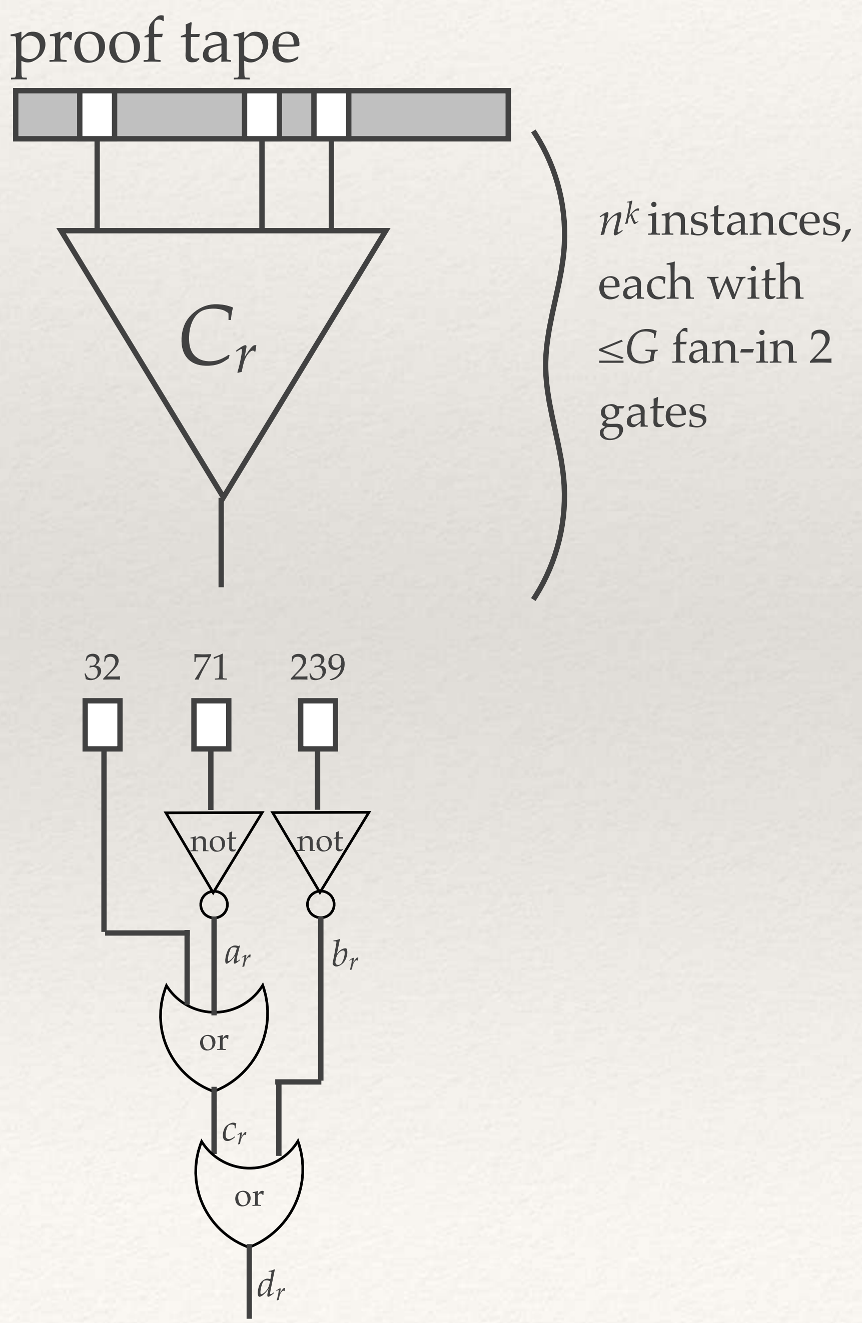
If **NP**=**PCP** then ∃ ε>0, **MAX3SAT**(ε) **NP**-hard

- We now encode those circuits as a **3SAT** formula, e.g.:



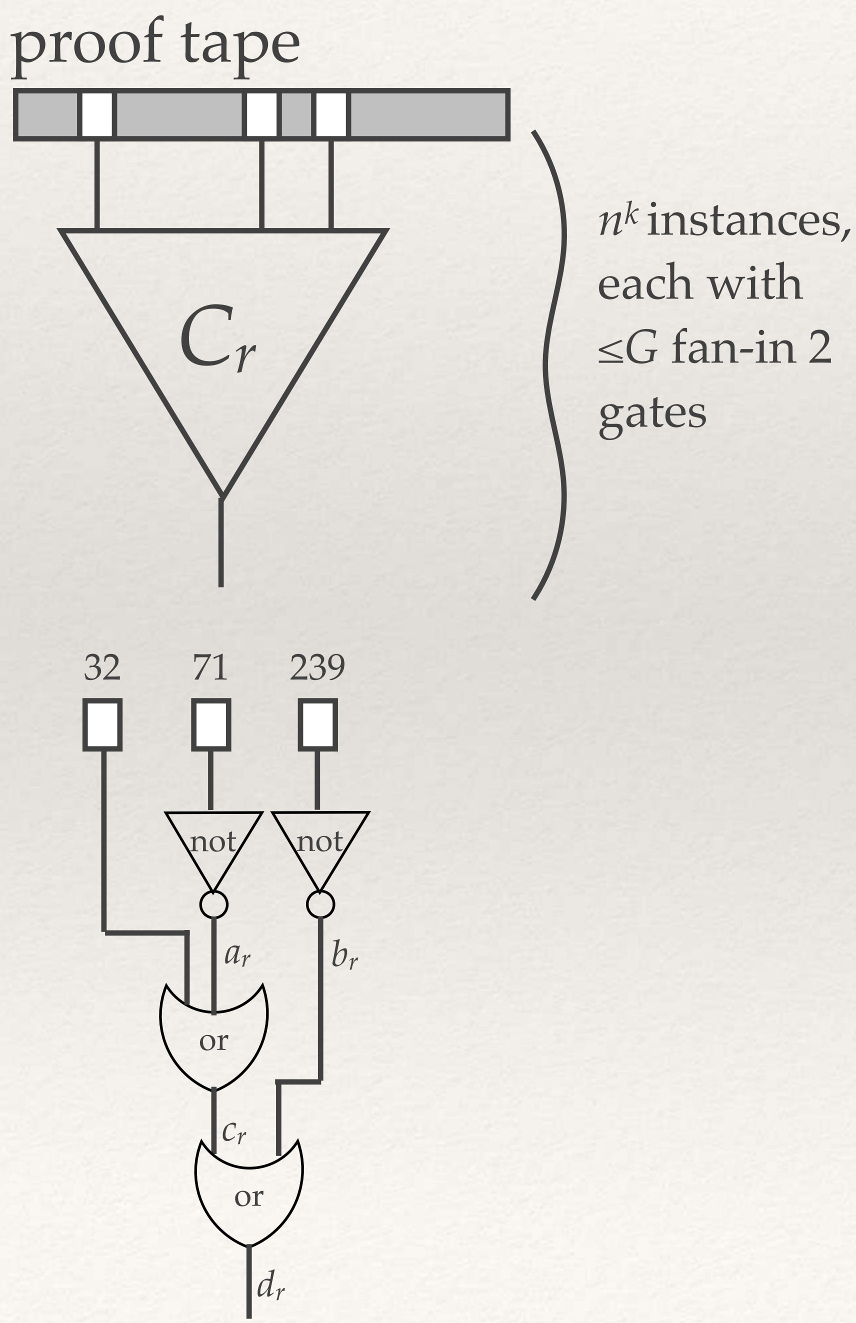
If NP=PCP then ∃ ε>0, MAX3SAT(ε) NP-hard

- We now encode those circuits as a **3SAT** formula, e.g.:



If NP=PCP then ∃ ε>0, MAX3SAT(ε) NP-hard

- We now encode those circuits as a **3SAT** formula, e.g.:



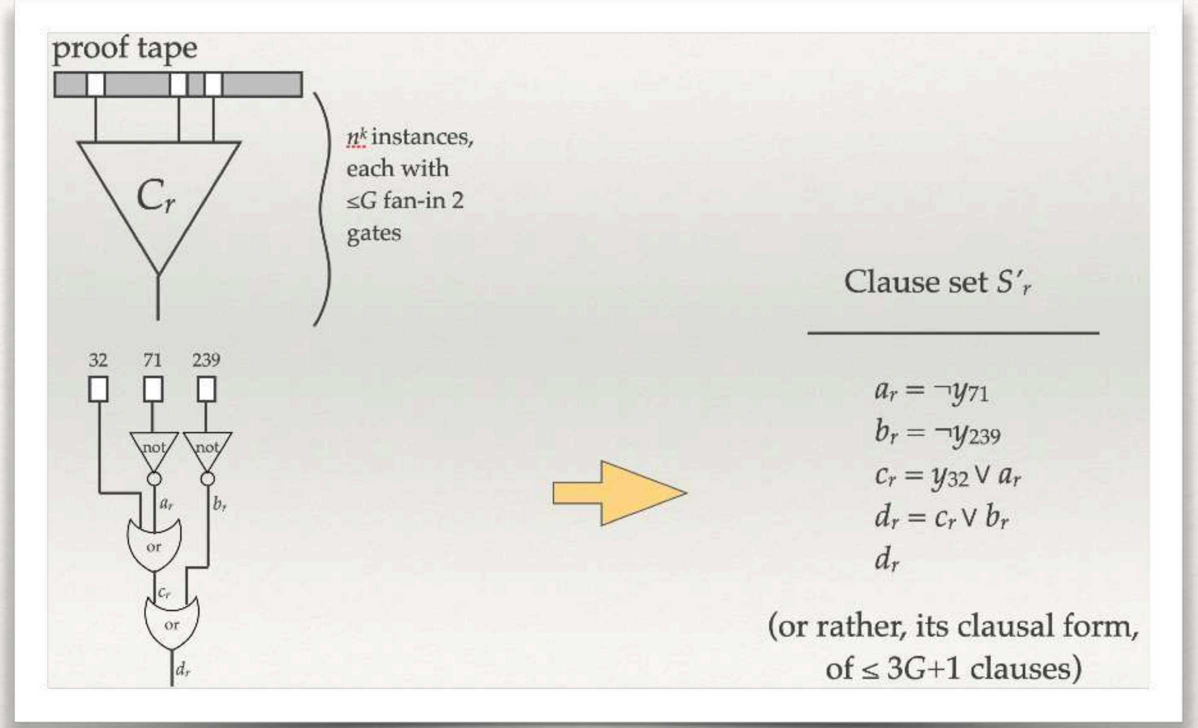
Clause set S'_r

$$a_r = \neg y_{71}$$
$$b_r = \neg y_{239}$$
$$c_r = y_{32} \vee a_r$$
$$d_r = c_r \vee b_r$$
$$d_r$$

(or rather, its clausal form,
of ≤ 3G+1 clauses)

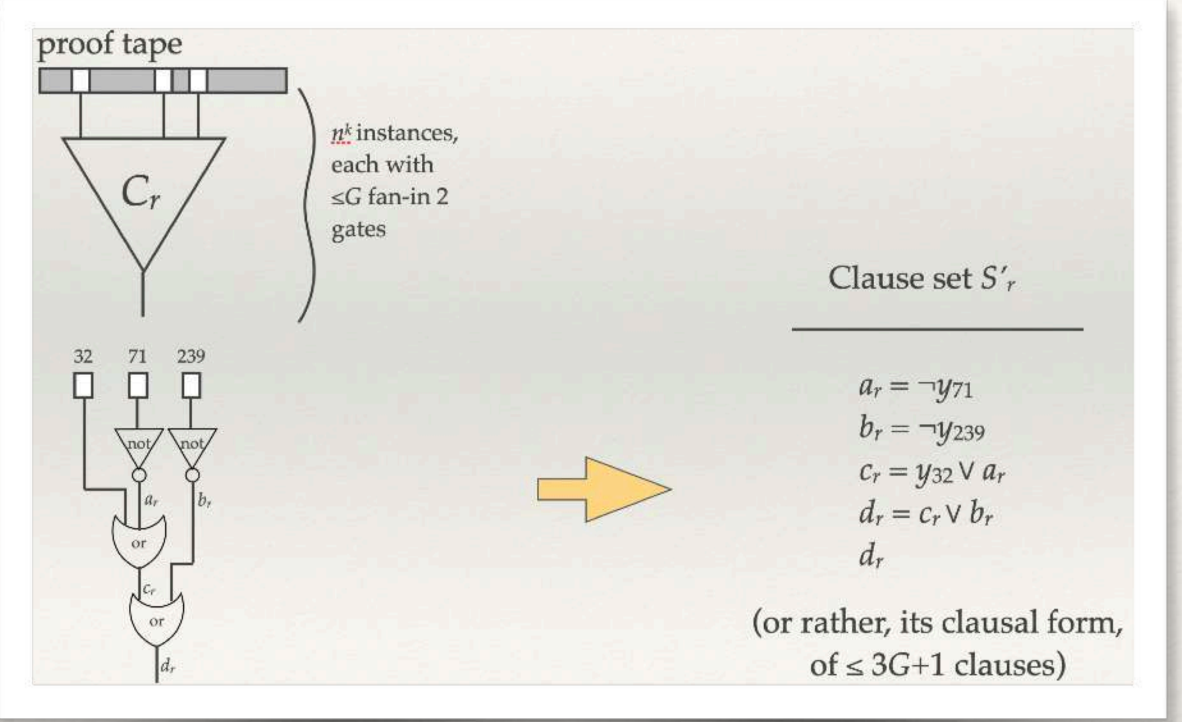
If NP=PCP then ∃ ε>0, MAX3SAT(ε) NP-hard

- If S satisfiable, then Merlin can provide y / $\forall r$, output(C_r) true, so $S' \overset{\text{def}}{=} \bigwedge_r S_r$ is satisfiable



If NP=PCP then ∃ ε>0, MAX3SAT(ε) NP-hard

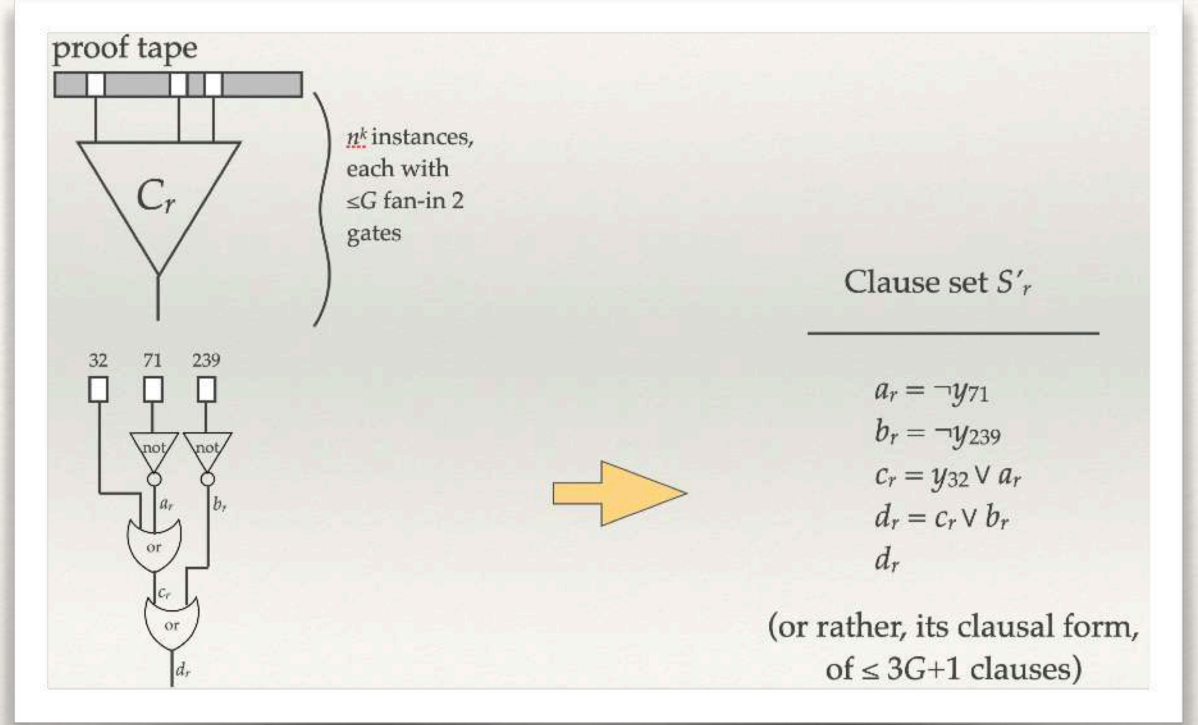
- If S satisfiable, then Merlin can provide y / $\forall r$, output(C_r) true, so $S' \stackrel{\text{def}}{=} \bigwedge_r S_r$ is satisfiable
- Otherwise, whatever y, there is a set I of $\geq \frac{1}{2}n^k$ values of r / $\forall r \in I$, output(C_r) false



If NP=PCP then ∃ ε>0, MAX3SAT(ε) NP-hard

- If S satisfiable, then Merlin can provide y / $\forall r$, output(C_r) true, so $S' \stackrel{\text{def}}{=} \bigwedge_r S_r$ is satisfiable

- Otherwise, whatever y, there is a set I of $\geq \frac{1}{2}n^k$ values of r / $\forall r \in I$, output(C_r) false

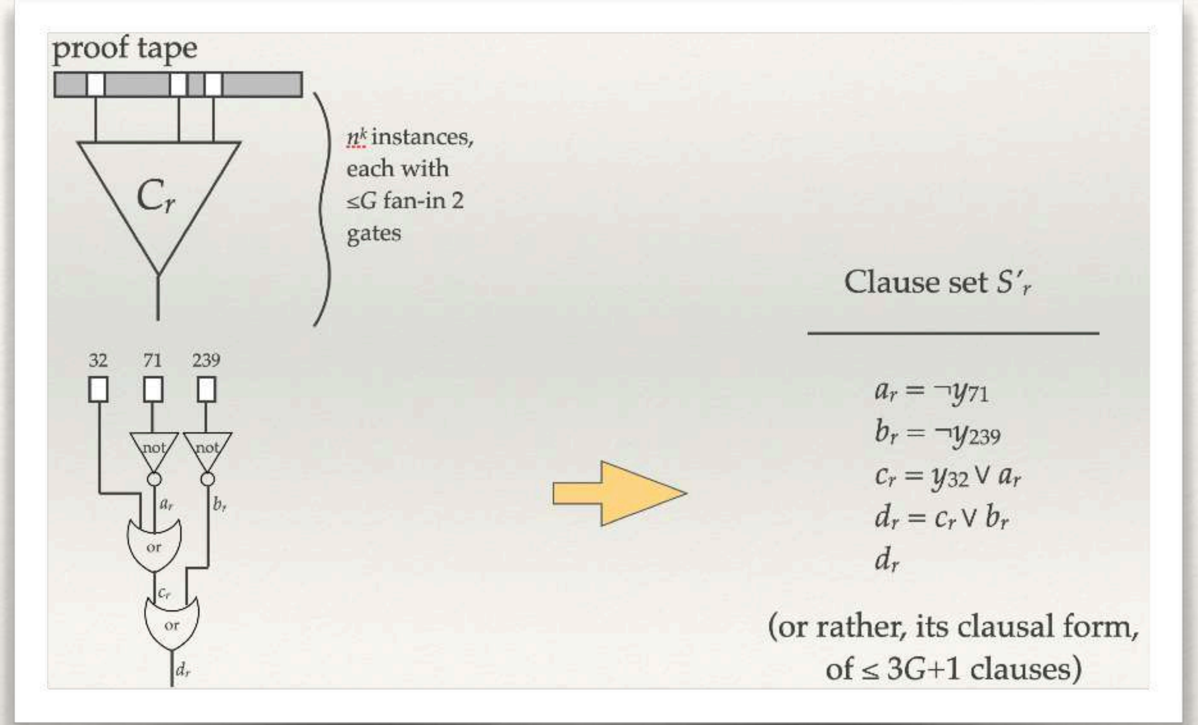


- hence $\forall \varrho$ (giving truth values to each y_i and to every auxiliary var.), at least one clause in each S'_r, $r \in I$, must be unsatisfied by ϱ

If NP=PCP then ∃ ε>0, MAX3SAT(ε) NP-hard

- If S satisfiable, then Merlin can provide y / $\forall r$, output(C_r) true, so $S' \stackrel{\text{def}}{=} \bigwedge_r S_r$ is satisfiable

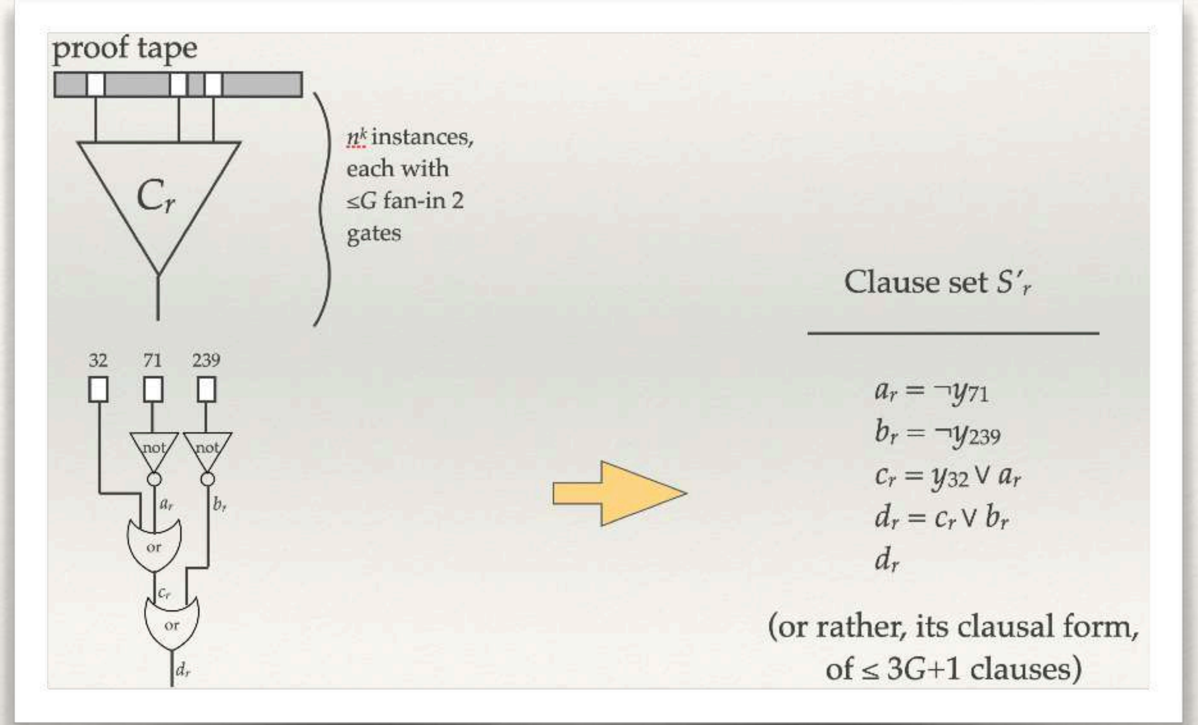
- Otherwise, whatever y, there is a set I of $\geq \frac{1}{2}n^k$ values of r / $\forall r \in I$, output(C_r) false



- hence $\forall \varrho$ (giving truth values to each y_i and to every auxiliary var.), at least one clause in each S'_r, $r \in I$, must be unsatisfied by ϱ

- so opt(S') ≤ #clauses in S' – $\frac{1}{2}n^k$, and #clauses in $S' \leq (3G+1)n^k$, so opt(S') / #clauses in $S' \leq 1-(\frac{1}{2}n^k)/((3G+1)n^k) = 1-1/(6G+2)$

If NP=PCP then ∃ ε>0, MAX3SAT(ε) NP-hard



- If S satisfiable, then Merlin can provide y / $\forall r$, output(C_r) true, so $S' \stackrel{\text{def}}{=} \bigwedge_r S_r$ is satisfiable
- Otherwise, whatever y, there is a set I of $\geq \frac{1}{2} n^k$ values of r / $\forall r \in I$, output(C_r) false
- hence $\forall \varrho$ (giving truth values to each y_i and to every auxiliary var.), at least one clause in each S'_r, $r \in I$, must be unsatisfied by ϱ
- so opt(S') ≤ #clauses in S' – $\frac{1}{2} n^k$, and #clauses in $S' \leq (3G+1)n^k$, so opt(S')/#clauses in $S' \leq 1-(\frac{1}{2}n^k)/((3G+1)n^k) = 1-1/(6G+2)$
- Therefore S' is an instance of **MAX3SAT**(ε), with $\varepsilon \stackrel{\text{def}}{=} 1/(6G+2)$

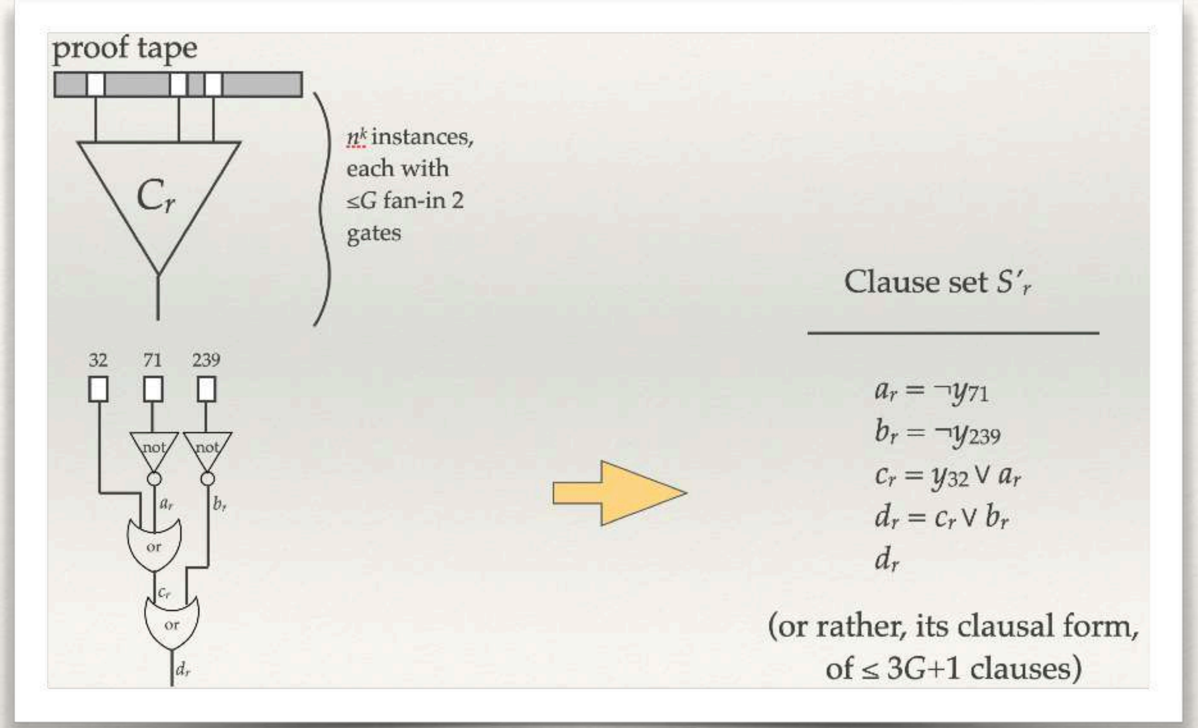
If NP=PCP then ∃ ε>0, MAX3SAT(ε) NP-hard

- **Summary**: If S satisfiable,

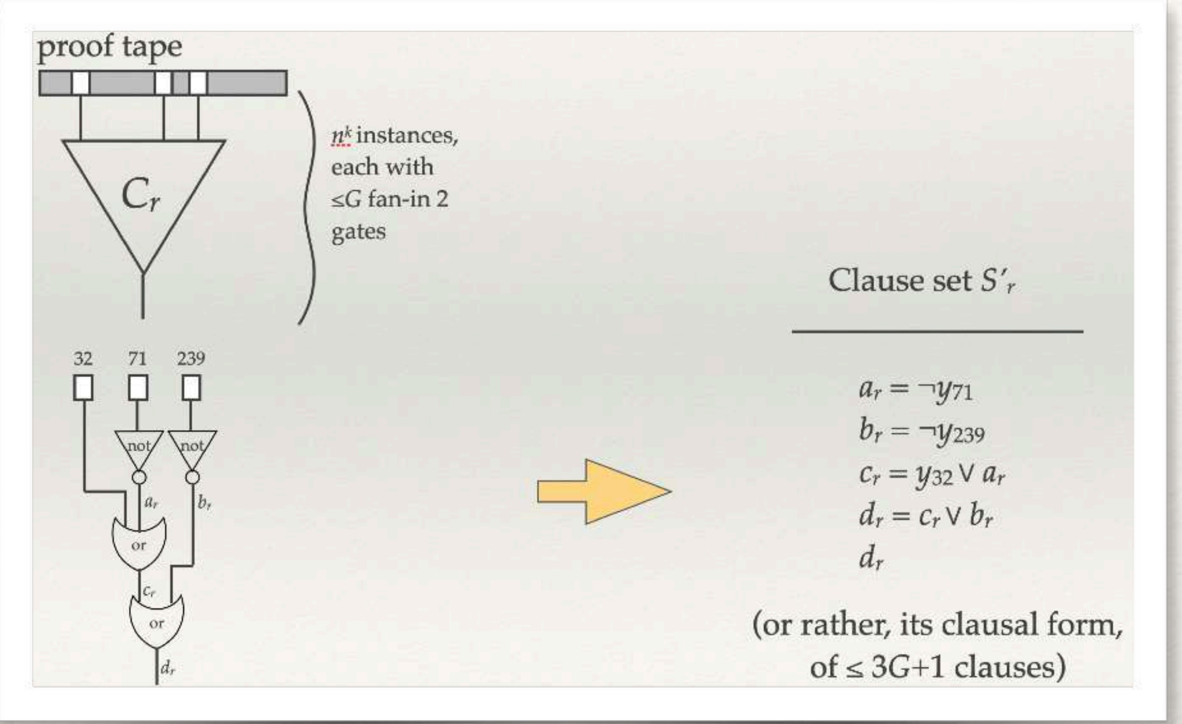
  then $S' \stackrel{\text{def}}{=} \bigwedge_r S_r$ is satisfiable

  Else, opt(S') ≤ (1–ε).#clauses in S'

  where $\varepsilon \stackrel{\text{def}}{=} 1/(6G+2)$



If NP=PCP then ∃ ε>0, MAX3SAT(ε) NP-hard



- **Summary**: If S satisfiable,

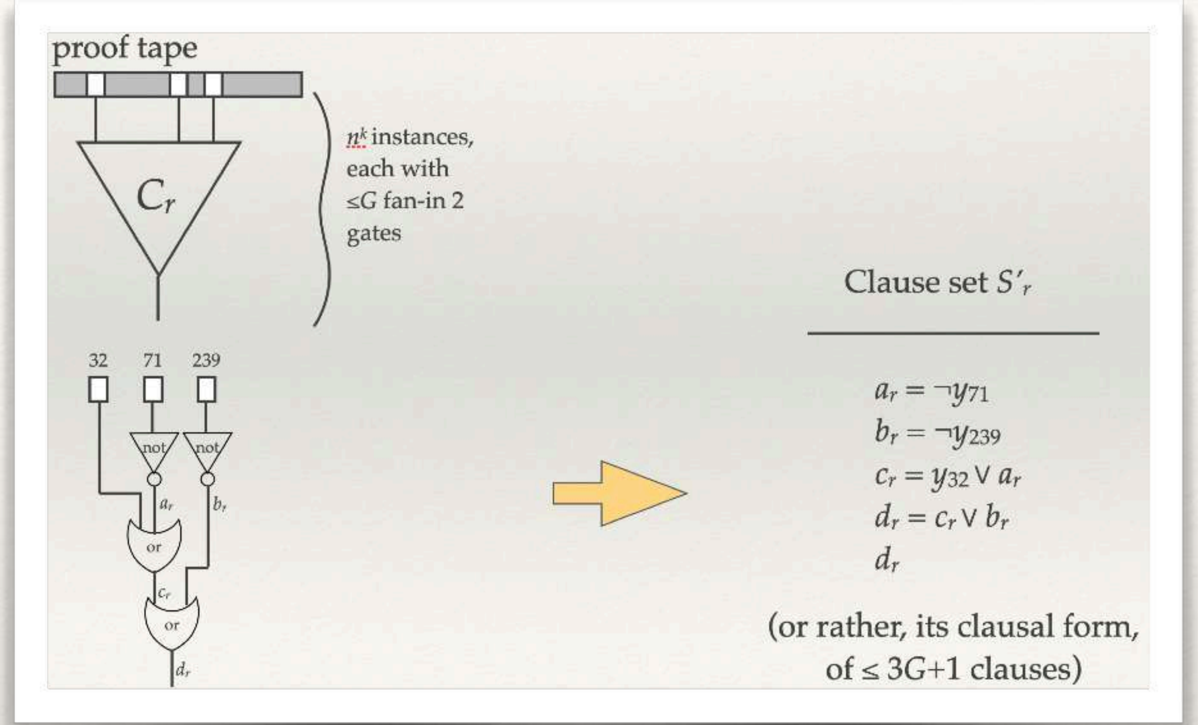
  then $S' \overset{\text{def}}{=} \bigwedge_r S_r$ is satisfiable

  Else, opt(S') ≤ (1–ε).#clauses in S'

  where $\varepsilon \overset{\text{def}}{=} 1/(6G+2)$

- Additionally, each C_r can be computed in polynomial time (simulating Arthur's computation),
  and computing S'_r from C_r also takes polynomial time

If NP=PCP then ∃ ε>0, MAX3SAT(ε) NP-hard



- **Summary**: If S satisfiable,

  then $S' \stackrel{\text{def}}{=} \bigwedge_r S_r$ is satisfiable

  Else, $\text{opt}(S') \leq (1-\varepsilon).\#$clauses in S'

  where $\varepsilon \stackrel{\text{def}}{=} 1/(6G+2)$

- Additionally, each C_r can be computed in polynomial time (simulating Arthur's computation),
  and computing S'_r from C_r also takes polynomial time

- Hence we have found a polytime reduction from **SAT** to **MAX3SAT**(ε) (assuming **NP**=**PCP**). □

Irit Dinur

Par נובאמת — Travail personnel,

https://commons.wikimedia.org/w/index.php?curid=72395593

Home > ACM Journals > Journal of the ACM > Vol. 54, No. 3 > The PCP theorem by gap amplification

(2007)

ARTICLE

The PCP theorem by gap amplification

Author: Irit Dinur Authors Info & Affiliations

Publication: Journal of the ACM • June 2007 • https://doi.org/10.1145/1236457.1236459

The PCP Theorem by Gap Amplification

Irit Dinur*

September 26, 2005

Abstract

We describe a new proof of the PCP theorem that is based on a combinatorial amplification lemma. The *unsat value* of a set of constraints $\mathcal{C} = \{c_1, \ldots, c_n\}$, denoted $\text{UNSAT}(\mathcal{C})$, is the smallest fraction of unsatisfied constraints, ranging over all possible assignments for the underlying variables.

We prove a new combinatorial amplification lemma that doubles the unsat-value of a constraint-system, with only a linear blowup in the size of the system. Iterative application of this lemma yields a proof for the PCP theorem.

The amplification lemma relies on a new notion of "graph powering" that can be applied to systems of constraints. This powering amplifies the unsat-value of a constraint system provided that the underlying graph structure is an expander.

We also apply the amplification lemma to construct PCPs and locally-testable codes whose length is linear up to a *polylog* factor, and whose correctness can be probabilistically verified by making a *constant* number of queries. Namely, we prove $SAT \in PCP_{\frac{1}{2},1}[\log_2(n \cdot \text{poly}\log n), O(1)]$. This answers an open question of Ben-Sasson et al. (STOC '04).

- Simplified proof…
  I will only give a rough sketch
- Uses expander graphs, « powering » on random walks, Hadamard codes, etc.

Constraint graph satisfiability

- Instead of **MAX3SAT**(ε), Dinur uses:
- **Defn.** A **constraint graph** is an undirected graph (V, E) plus a set of constraints $c(e) \subseteq \Sigma\times\Sigma$, one for each edge e … where Σ is a finite set of values, or **colors**,
  that each vertex may assume under a **color assignment**

Constraint graph satisfiability

- Instead of **MAX3SAT**(ε), Dinur uses:
- **Defn.** A **constraint graph** is an undirected graph (V, E) plus a set of constraints $c(e) \subseteq \Sigma\times\Sigma$, one for each edge e … where Σ is a finite set of values, or **colors**, that each vertex may assume under a **color assignment**
- Question: is there a color assignment satisfying all the edge constraints?

Constraint graph satisfiability

- Instead of **MAX3SAT**(ε), Dinur uses:
- **Defn.** A **constraint graph** is an undirected graph (V, E) plus a set of constraints $c(e) \subseteq \Sigma\times\Sigma$, one for each edge e … where Σ is a finite set of values, or **colors**, that each vertex may assume under a **color assignment**
- Question: is there a color assignment satisfying all the edge constraints?
- **NP**-complete, generalizes **3-COLORABILITY**

The gap

- The **gap** of an unsatisfiable contraint graph is
  min (#unsatisfied edge constraints) / m [$m \stackrel{\text{def}}{=}$ #edges]
- We start with an unsatisfiable constraint graph G

The gap

- The **gap** of an unsatisfiable contraint graph is min (#unsatisfied edge constraints) / m [$m \stackrel{\text{def}}{=}$ #edges]
- We start with an unsatisfiable constraint graph G
- … of gap $\geq 1/m$

The gap

- The **gap** of an unsatisfiable contraint graph is min (#unsatisfied edge constraints) / m [$m \stackrel{\text{def}}{=}$ #edges]
- We start with an unsatisfiable constraint graph G
- … of gap $\geq 1/m$
- and we modify it so as to increase its gap until we reach a **constant** non-zero number

The gap

- The **gap** of an unsatisfiable contraint graph is min (#unsatisfied edge constraints) / m [$m \stackrel{\text{def}}{=}$ #edges]
- We start with an unsatisfiable constraint graph G
- … of gap $\geq 1/m$
- and we modify it so as to increase its gap until we reach a **constant** non-zero number
- Applied to a **satisfiable** constraint graph, the modifications will preserve satisfiability.

Graph expanders

- A **graph expander** is a family of undirected graphs with « good connectivity »
- **Defn.** The **edge expansion** $h(G)$ of a graph G is min (#edges between S and its complement / #S) over subsets S of $<n/2$ vertex of G [$n \stackrel{\text{def}}{=}$ #vertices]

Graph expanders

- A **graph expander** is a family of undirected graphs
  with « good connectivity »

- **Defn.** The **edge expansion** $h(G)$ of a graph G is
  min (#edges between S and its complement / #S)
  over subsets S of $<n/2$ vertex of G [$n \stackrel{\text{def}}{=}$ #vertices]

- A **graph expander** is a family of graphs G_n, $n \in \mathbf{N}$,
  — each regular of constant degree d_0
  — with n vertices each
  — such that $\boxed{h(G_n) \geq h_0, \text{ a positive constant}}$

Graph expanders

- A **graph expander** is a family of undirected graphs with « good connectivity »

- **Defn.** The **edge expansion** $h(G)$ of a graph G is
  min (#edges between S and its complement / #S)
  over subsets S of $<n/2$ vertex of G [$n \stackrel{\text{def}}{=}$ #vertices]

- A **graph expander** is a family of graphs G_n, $n \in \mathbf{N}$,
  — each regular of constant degree d_0
  — with n vertices each
  — such that $\boxed{h(G_n) \geq h_0\text{, a positive constant}}$

This exists, and G_n can even be produced in polynomial time (in n)

Graph expanders

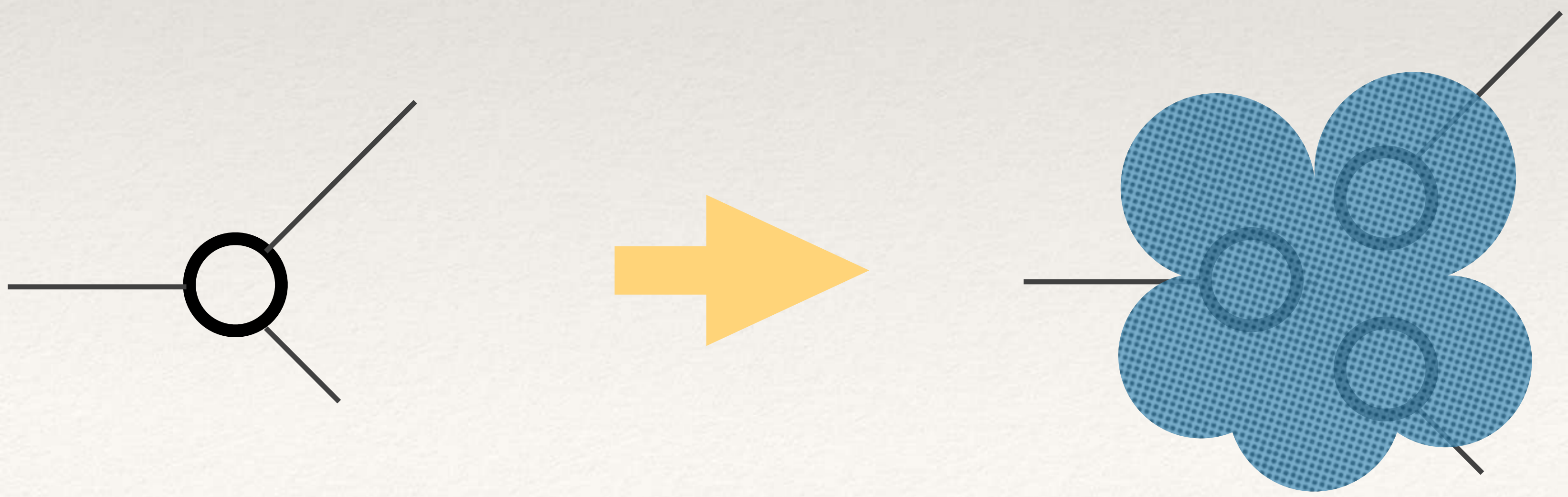
- A **graph expander** is a family of undirected graphs with « good connectivity »
- A **random walk** on a graph expander is rapidly mixing, namely: just doing a few steps gets you exponentially close to the stationary distribution

1. Sparsification

- First step: make G **sparse enough**

  (so as to allow step 2 to apply; the important step is step 3)

  precisely: make it **regular** and of small enough degree d

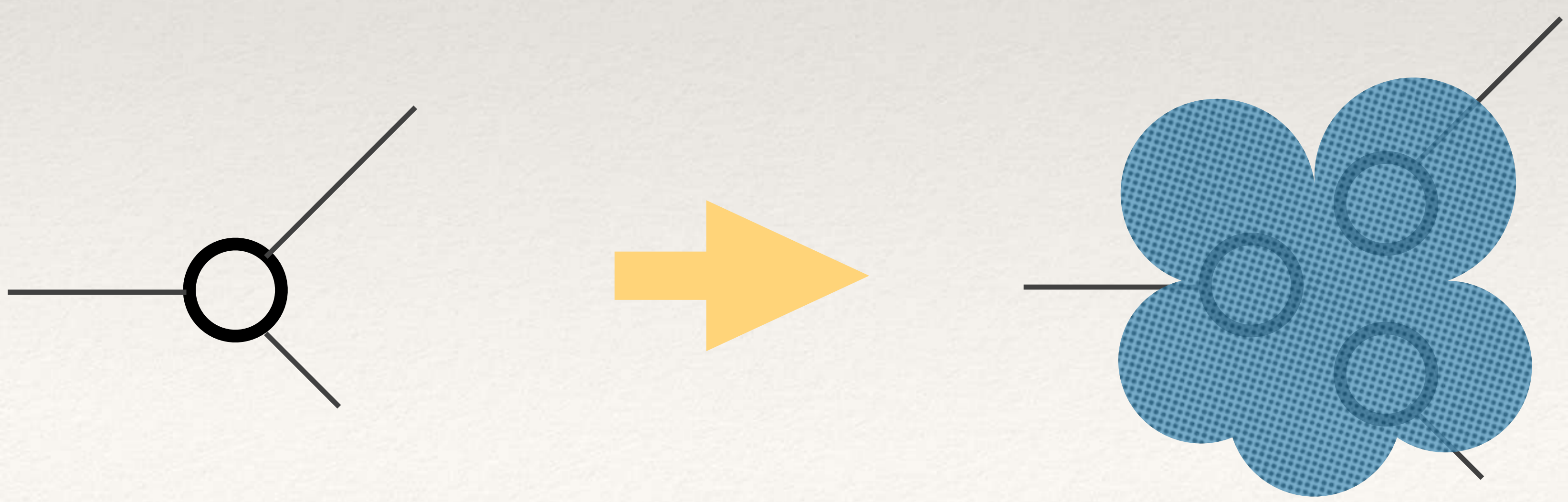


1. Sparsification

- First step: make G **sparse enough**

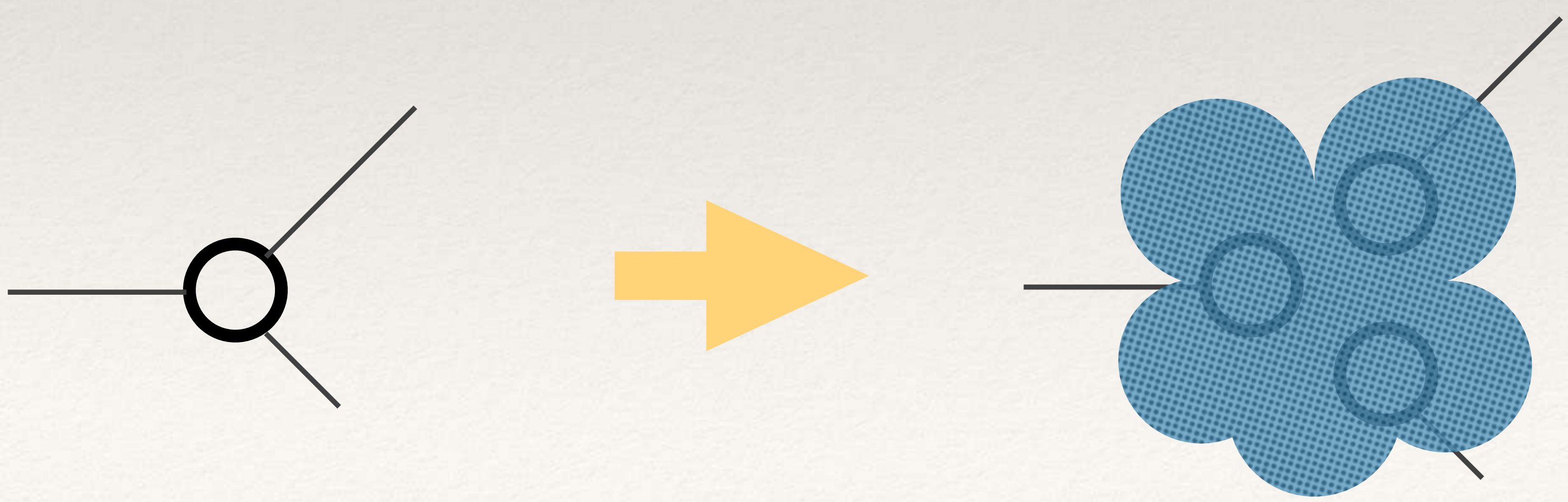
  (so as to allow step 2 to apply; the important step is step 3)

  precisely: make it **regular** and of small enough degree d
- Gap decreases by a **constant** factor only



1. Sparsification

- First step: make G **sparse enough**
  (so as to allow step 2 to apply; the important step is step 3)
  precisely: make it **regular** and of small enough degree d
- Gap decreases by a **constant** factor only
- Replace every vertex (degree, say, k) by
  a **graph expander** of degree d–1 with k vertices



2. Expanderize

- First step: make G **an expander**

  (so as to allow step 3 to apply)

2. Expanderize

- First step: make G **an expander**

  (so as to allow step 3 to apply)

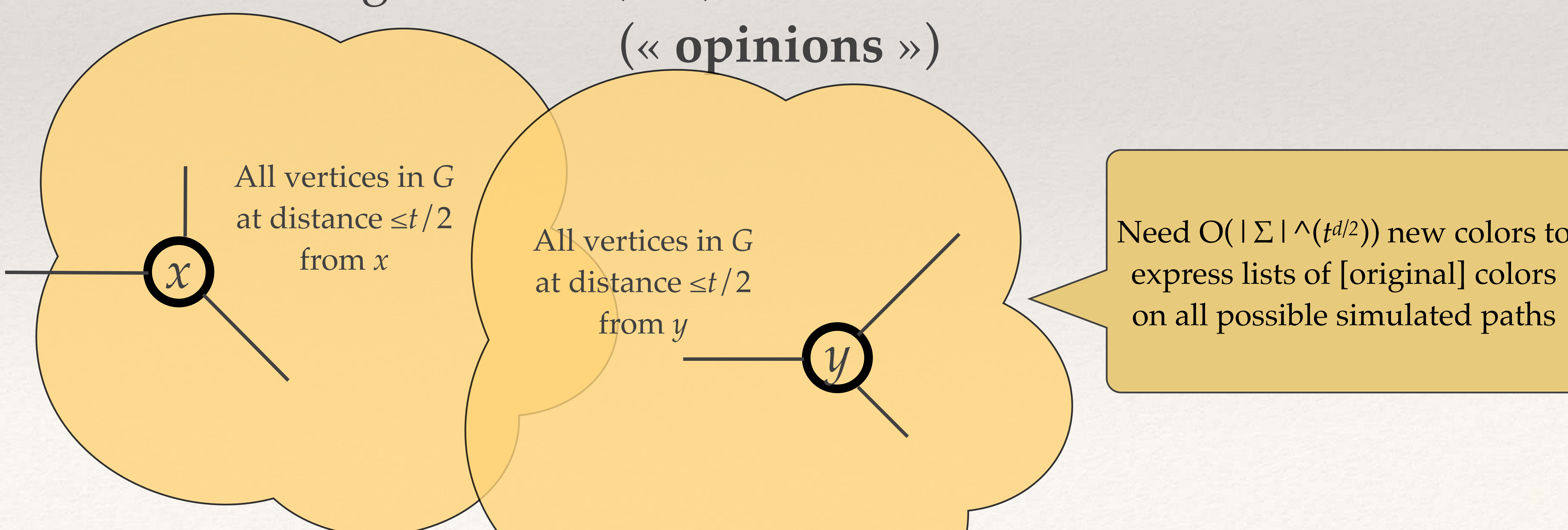
- By taking the union with a good expander

2. Expanderize

- First step: make G **an expander**
  (so as to allow step 3 to apply)
- By taking the union with a good expander
- Gap (also) decreases by a **constant** factor (only)

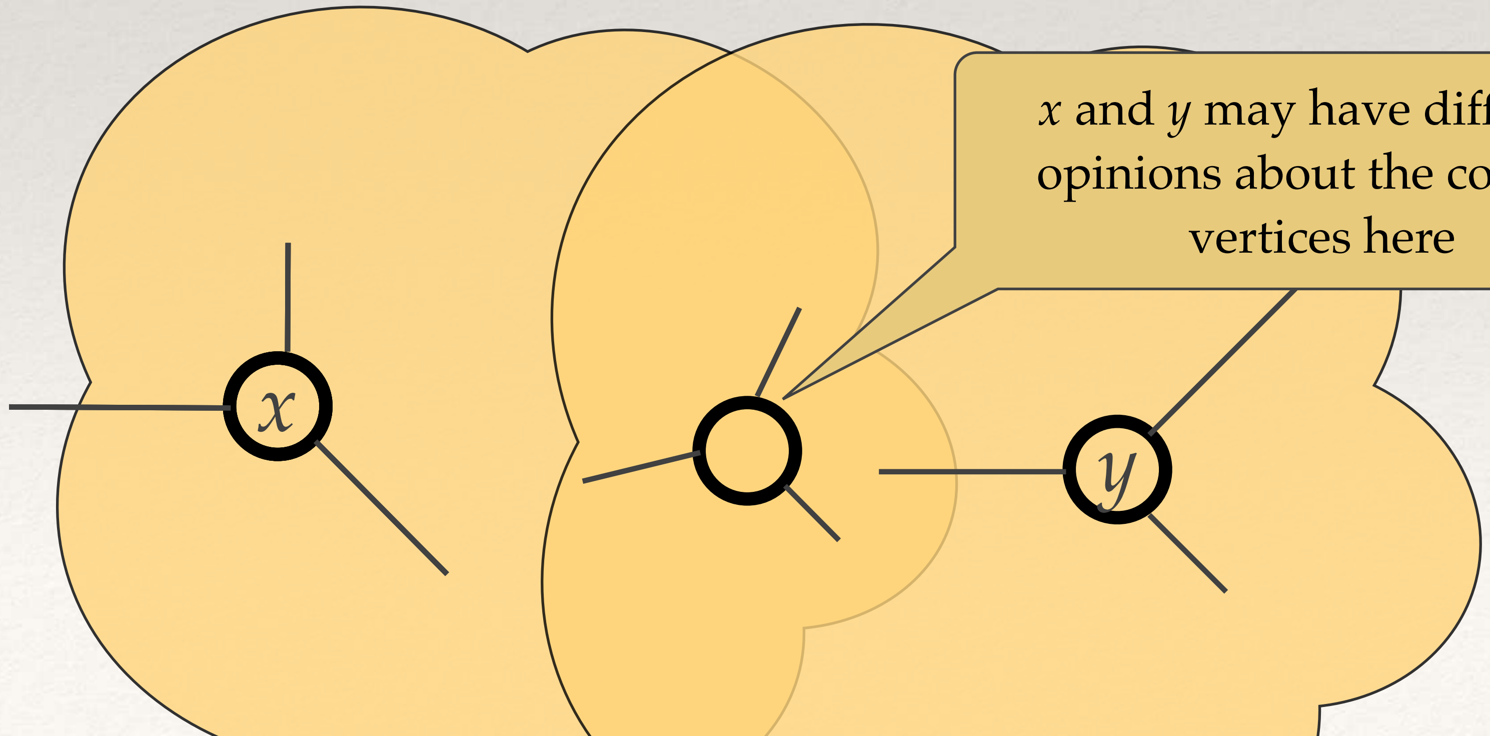
3. Amplify the gap

- This is the difficult step.
- Fix a constant $t>0$, and build a new constraint graph G^t
  whose single edges simulate **paths** of t edges in G
  (there are as many edges between x and y in G^t as paths in G)
- Encode distance $\leq t/2$ neighborhoods around each vertex
  New colors = assignment of (old) colors to vertices in those nbds
  (« **opinions** »)



3. Amplify the gap

- Problem: close vertices in G may be assigned incompatible opinions (**consistency** problem)

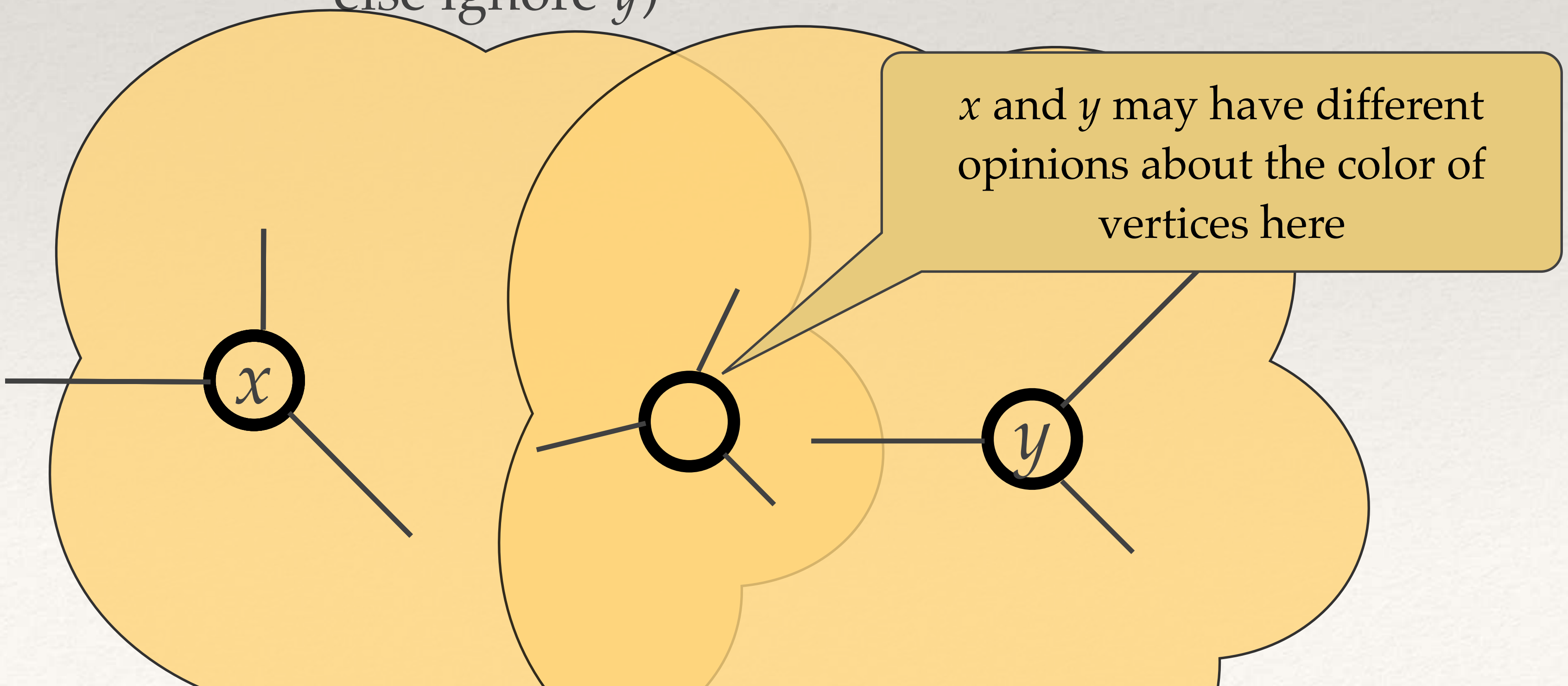


3. Amplify the gap

- Problem: close vertices in G may be assigned incompatible opinions (**consistency** problem)
- Correctness proof: given a color assignment on G^t,
  build back a color assignment on G:

  color of x (in G) $\stackrel{\text{def}}{=}$ **most likely** result as given by:

  (do **random walk** in G starting from x;
  stops at y with probability 1;
  if y is in neighborhood of x
  then return opinion of y on what the color of x should be
  else ignore y)



3. Amplify the gap

- The analysis is a bit complex, but:
- **Gap** is (finally) amplified, by roughly $\sqrt{t}$ while gap$\leq 1/t$

3. Amplify the gap

- The analysis is a bit complex, but:
- **Gap** is (finally) amplified, by roughly $\sqrt{t}$ while gap$\leq 1/t$

3. Amplify the gap

- The analysis is a bit complex, but:
- **Gap** is (finally) amplified, by roughly $\sqrt{t}$ while gap$\leq 1/t$
- … although we need O(|Σ|^($t^{d/2}$)) new colors to solve consistency (express lists of [original] colors on all possible simulated paths)

4. Alphabet reduce

- Reduce back the alphabet of colors **constant size** (2^6, i.e. 64)
- By encoding constraints through assignment testers
  assignments are encoded by **Hadamard error-correcting** codes
  [correct many errors, but exponentially large —
  which is not a problem here because this will be the exponential of a constant…]

4. Alphabet reduce

- Reduce back the alphabet of colors **constant size** (2^6, i.e. 64)
- By encoding constraints through assignment testers assignments are encoded by **Hadamard error-correcting** codes
  [correct many errors, but exponentially large —
  which is not a problem here because this will be the exponential of a constant…]
- Decreases back gap by some constant factor

4. Alphabet reduce

- Reduce back the alphabet of colors **constant size** (2^6, i.e. 64)
- By encoding constraints through assignment testers
  assignments are encoded by **Hadamard error-correcting** codes
  [correct many errors, but exponentially large —
  which is not a problem here because this will be the exponential of a constant…]
- Decreases back gap by some constant factor
- … and repeat steps 1—4 until gap becomes larger than a constant
  (requires $O(\log m)$ iterations)

Dinur's algorithm summarized

Step	Main Ideas	Effects	Proof Techniques
Degree Reduce	Split every vertex in to many vertices,and introduce an Expander cloud with equality constraints among the split vertices.	Size $\uparrow$ a $O(d)$ factor, Gap decreases by a constant factor, Alphabet remains same	Basic expansion property of expanders
Expanderize	Superimpose a constant degree expander with trivial constraints, on to the constraint graph G	Size $\uparrow$ a factor of 2 to 3, Gap decreases by a constant factor, Alphabet remains same	Existence of constant degree expanders and Property that Expander + Graph gives an expander.
Gap-Amplification	Each vertex's value is its opinion,on the values of vertices at a distance $< t$,Add edges corresponding to consistency on random walks	Size $\uparrow$ by a large constant factor ,Gap increases by $O(t)$, Alphabet size becomes $\lvert\Sigma\rvert^{O(d^t)}$	Properties of random walks on the graph
Alphabet-Reduce	Encode the assignment with error correcting codes, Build a circuit that checks if assignment satisfies and is a valid codeword, Use an assignment tester for the circuit	Size $\uparrow$ a constant factor, Gap decreases by a constant factor,Alphabet size reduced to 2^6	Hadamard codes, Linearity Testing, Fourier Analysis

Table 1: Proof of PCP

https://courses.cs.washington.edu/courses/cse533/05au/

and…

That's it, folks!

- I hope you enjoyed the material of the course!