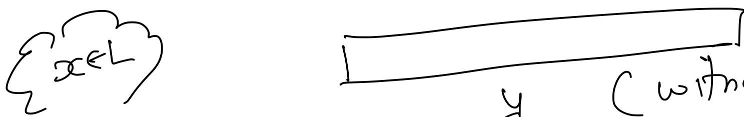
PCP – an alternate view of NP

(and hardness of approximation)

NP definition:

$L \in NP$ if $\exists$ polynomial time verifier (TM) V and an integer $c \geq 0$ s.t

$x \in L \iff \exists y, \; |y| \leq |x|^c$, s.t $V(x,y)$ accepts



y (witness / proof / certificate) to fact that $x \in L$

Given x & y, one can efficiently check y certifies $x \in L$

(It might be hard to come up with such a witness, y).

$L = 3COLOR$

$= \{\langle G \rangle \mid G$ is 3-colorable$\}$

$x = \langle G \rangle$

$y =$ a 3-coloring of G (witness / proof)

Only valid claims "$x \in L$" have proofs.

For Invalid claims ("$x \in L$" when x is in fact not in L), <u>every</u> proof will be rejected

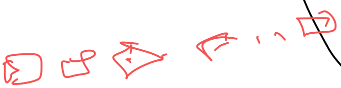
Why is NP called NP?

NP = "nondeterministic" polynomial time

||

Languages that are decided by a nondet. polytime TM. (Nondet. polytime TM:

- ($\delta : Q \times \Sigma \to Q \times \Gamma \times \{L, R\}$
  ↳ det TM: single transition function)
- ↳ δ_0, δ_1 : two transition functions
  (Each step TM can operate according to either one of them, chosen nondeterministically)
- ↳ TM runs for poly(|x|) steps



Depth of tree = nondet runtime = poly(|x|)

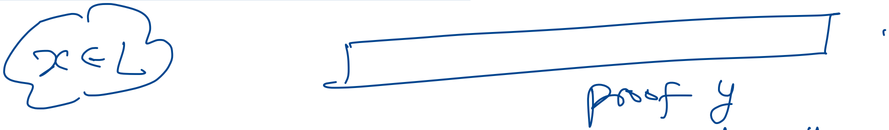
→ x is "accepted" if at least one leaf is an accepting configuration

Exercise: The two definitions of NP, the polytime verifier based one, and the nondeterministic polytime TM based one are both equivalent.

For finite automata: NFA ≡ DFA in power (both recognize regular languages)

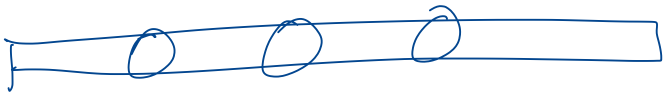
But we believe $P \subsetneq NP$ (nondet. much more powerful for TMs)

Back to verifier view:



NP view of verification requires reading the proof in entirety.

"PCP": – "Quick and dirty" proof verification (Spot checking)



(or few) Spot check proof in 3 locations of giant proof

And if local view "checks out", accept proof.

Busy (Lazy?) grader's dream

Such a "local" proof checker must be randomized / probabilistic.

(Use random coins to sample 3 locations of proof)

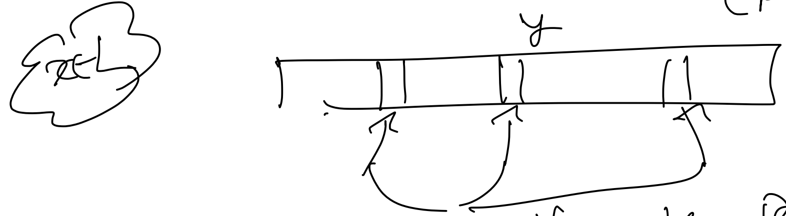
Q: Can this be done so fake claims are caught with good prob. even though the checking is so "spotty"/"local"?

Is there a robust proof writing format such that for false claims, there are bugs in the proof all over the place!

PCP (Probabilistically Checkable proofs) Theorem

(1992): $\exists$ finite q (integer) s.t following holds: $\forall L \in NP$, $\exists c \in \mathbb{N}$, and a randomized polynomial time verifier V s.t

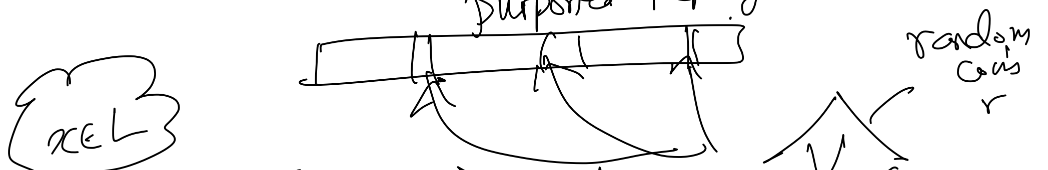
(Completeness) • $x \in L$, $\exists y$, $|y| \le |x|^c$, such that $V(x,y)$ accepts with certainty (prob $= 1$)



V probes proof y in q positions

(Soundness) • If $x \notin L$, $\forall y$, $|y| \le |x|^c$, $V(x,y)$ rejects with prob $\ge 40\%$

Furthermore, V only probes/reads q (randomly chosen) locations of the proof.



(Acc/rej based on value of q bits read)

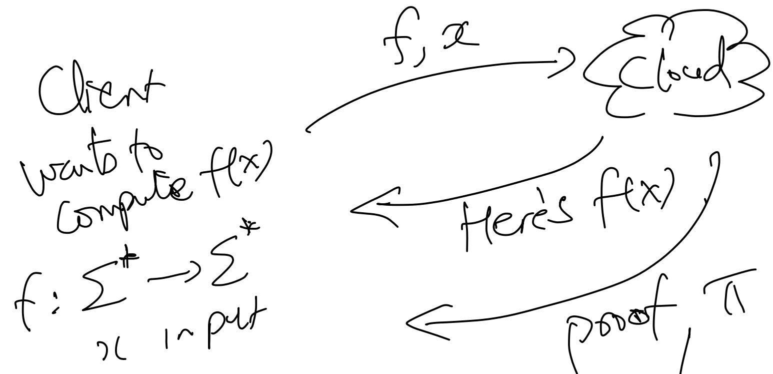
(In fact, can take $q=3$, (shown later by optimizing the PCP)

but can't take $q=2$)

(3SAT is NP-complete; 2SAT is in P)

Verifier only needs $O(\log n)$ random bits.

- PCP theorem is a $\boxed{\text{major}}$ result in TCS.
- New perspective on proofs / verification of proofs
- Can potentially use this in numerous verification applications, eg delegating computations to cloud & checking results.



How to trust this?

"proof of correct work"

Could be a PCP that can be quickly spot checked!

- Applications in Cryptography, block chains

Connections to Approximation algorithms ("showing "hardness of approximation")

Approximation connection

3SAT is NP-complete

$$\begin{cases} (x_1 \vee x_2 \vee \bar{x}_3) \\ \wedge (x_2 \vee \bar{x}_3 \vee x_4) \\ \wedge \ldots \wedge (x_7 \vee \bar{x}_4 \vee x_8) \\ \wedge \ldots (\ldots) \end{cases}$$

Given a satisfiable 3SAT formula, there is likely no polytime algo to find a satisfying assignment.

How about finding an approximately good assignment, say one which satisfies 99% of the clauses?

[OPTIMIZATION VERSION]

Also seems hard ...

How might one prove such a hardness?

A "gap producing" reduction:

CIRCUITSAT $\leq_p$ APPROX-3SAT

Circuit C $\xrightarrow[\text{map}]{\text{poly time}}$ 3CNF formula ϕ.

C satisfiable $\Longrightarrow$ ϕ is satisfiable

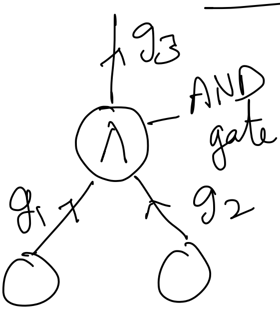
C not satisfiable $\Rightarrow$ ϕ does not have an assignment that satisfies even 99% of the clauses

(Note: Much stronger requirement than in usual NP-completeness reduction)

<u>Exercise</u>: Such a reduction as above implies that finding a 99% satisfying assignment to a fully satisfiable 3CNF formula is NP-hard.

Revisit "conventional" reduction:

<u>enforce</u>: $g_3 = g_1 \wedge g_2$



Snippet of circuit

$$\xmapsto{\text{Reduction}} \quad (\bar{g}_1 \vee \bar{g}_2 \vee g_3) \wedge (\bar{g}_1 \vee g_2 \vee \bar{g}_3) \wedge (g_1 \vee \bar{g}_2 \vee \bar{g}_3) \wedge (g_1 \vee g_2 \vee \bar{g}_3)$$

(Do above for every gate)

But in usual reduction as above, can just violate functioning of one gate of circuit and make it satisfiable.

(So violating just one or a few 3SAT clauses might be consistent with C being unsatisfiable).

That's why a gap producing reduction is tricky.

(Can't be fully local in reduction)

<u>Theorem:</u>

PCP theorem $\Longleftrightarrow$ Existence of such a gap producing reduction

PCP theorem was proved using algebraic methods & then $\Longrightarrow$ hardness of approximation

Later in 2005, a direct proof in other direction, via hardness of approx. was given

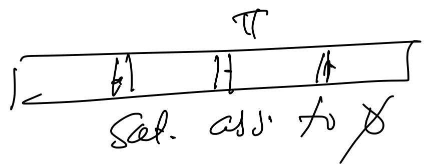
---

$\Longleftarrow$: How can a reduction give a PCP?

To prove circuit C is satisfiable:

Conventional pf: Sat. assignment to C.

PCP proof: A purported satisfying assignment π to ϕ, which is the output of above gap producing reduction applied to C



$$C \longmapsto \phi$$

How to verify? (lazily)
- Pick random clause of ϕ
- Check it is satisfied under π

Only need to read 3 bits!

If $C \notin$ CIRCUITSAT

verifier rejects with $\geq 1\%$ probability

To reject with prob $\geq 40\%$.

repeat t times so that

$$(0.99)^t \leq 0.6$$

$q = 3t$ queries. $\square$