Introduction to
Approximation Algorithms

Mong-Jen Kao (高孟駿)

Friday 13:20 – 15:10

The Complexity Class NP

&

Proof Checking

The classic point of view

The Complexity Class NP

- A language L is in NP
  if there is a ***nondeterministic Turing machine*** (NTM) M
  that decides it in polynomial-time.

  For any string x,

  - If $x \in L$, then *there exists a computation path* of M that accepts x.
  - If $x \notin L$, then all computation paths of M reject x.

An equivalent point of view

The Complexity Class NP

- A problem Π is in NP,

  if there is a ***polynomial-time algorithm*** $\boldsymbol{A}$ such that

  for any instance I of Π,

  – If I is a "Yes"-instance, then

    there is a ***proof*** $\pi \in \{0,1\}^{poly(n)}$ such that A accepts on (I, π).

  – If I is a "No"-instance, then A rejects (I, π) for all $\pi \in \{0,1\}^{poly(n)}$.

An equivalent point of view

The Complexity Class NP

- A problem Π is in NP if there is a ***proof system***

  for its ***yes answers*** *to be verified efficiently* in polynomial-time.

  - (Completeness)

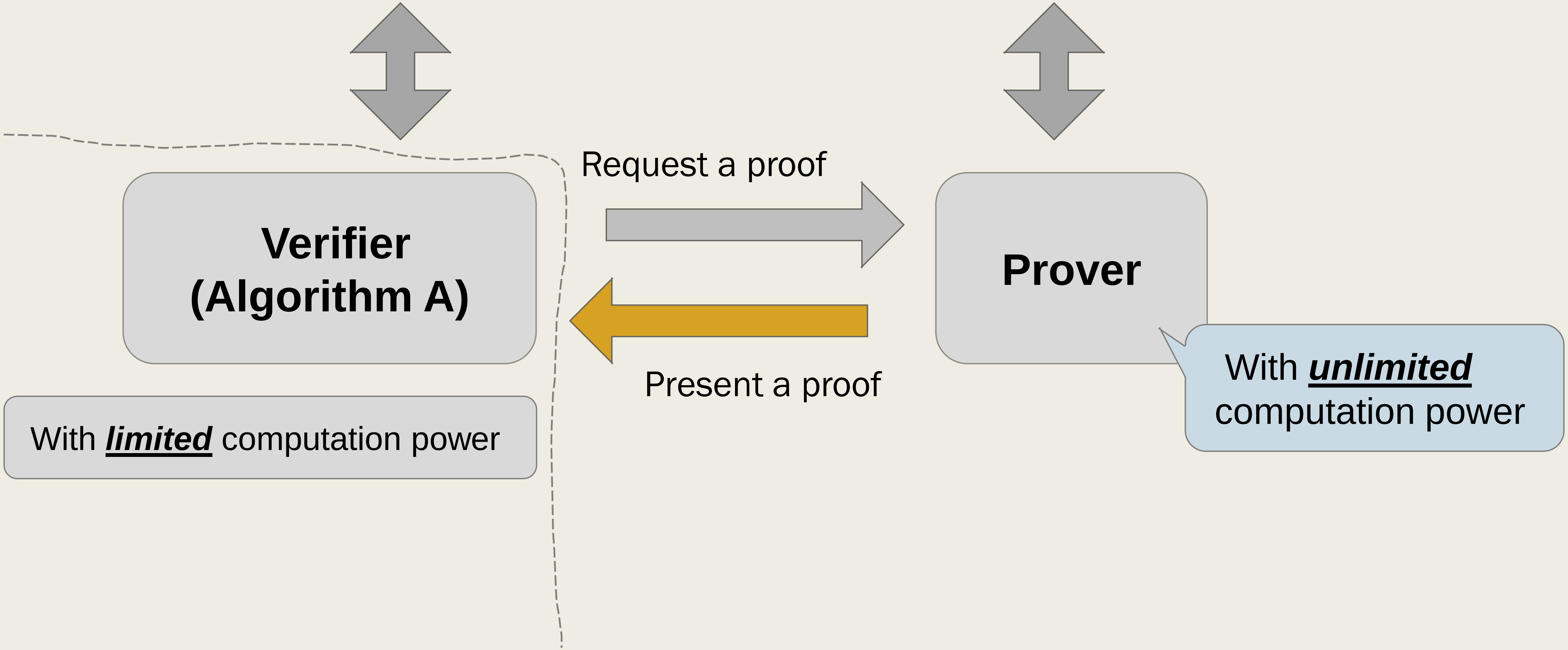
    For each “yes”-instance, there is a proof that leads to accept.

  - (Soundness)

    For each “no”-instance, no proof leads to accepts.

Assume a ***valid proof system*** for Π

that can be <u>*efficiently verified*</u> by an ***algorithm*** $\boldsymbol{A}$.



The classic point of view

The Complexity Class P

- A language L is in P

  if there is a deterministic Turing machine M that decides it

  in polynomial-time.

  For any string x,

  - If $x \in L$, then M accepts x in polynomial-time.
  - If $x \notin L$, then M rejects x in polynomial-time.

A ***Turing machine*** is actually an ***algorithm***, so…

The classic point of view

The Complexity Class P

- A problem Π is in P

  if there is a polynomial-time algorithm A that decides it.

  For any instance I,

  - A answers "Yes" if I is a "Yes"-instance, and

    "No" if I is a "No"-instance.

- The complexity class **P** consists of

  problems that can be solved efficiently in polynomial-time.

The Complexity Classes P vs NP

- From the proof-verifying perspective,
  - Problems in P are those, whose proof can be computed (composed) efficiently in polynomial-time.

- Obviously, $\boldsymbol{P} \subseteq \boldsymbol{NP}$.

- Whether or not $\boldsymbol{NP} \subseteq \boldsymbol{P}$ is *a major open problem* in CS.
  - Is *writing proofs* **as easy as** *verifying them*?

Do you believe so? :)

Probabilistically Checkable Proofs

(PCP)

How much effort does it require to check a proof for a problem in NP?

The Complexity Class PCP(r(n),q(n))

- A language L is in PCP(r(n),q(n))

  if there is a polynomial-time randomized algorithm V such that

  on any input string $x \in \{0,1\}^n$,

  - (Efficiency)

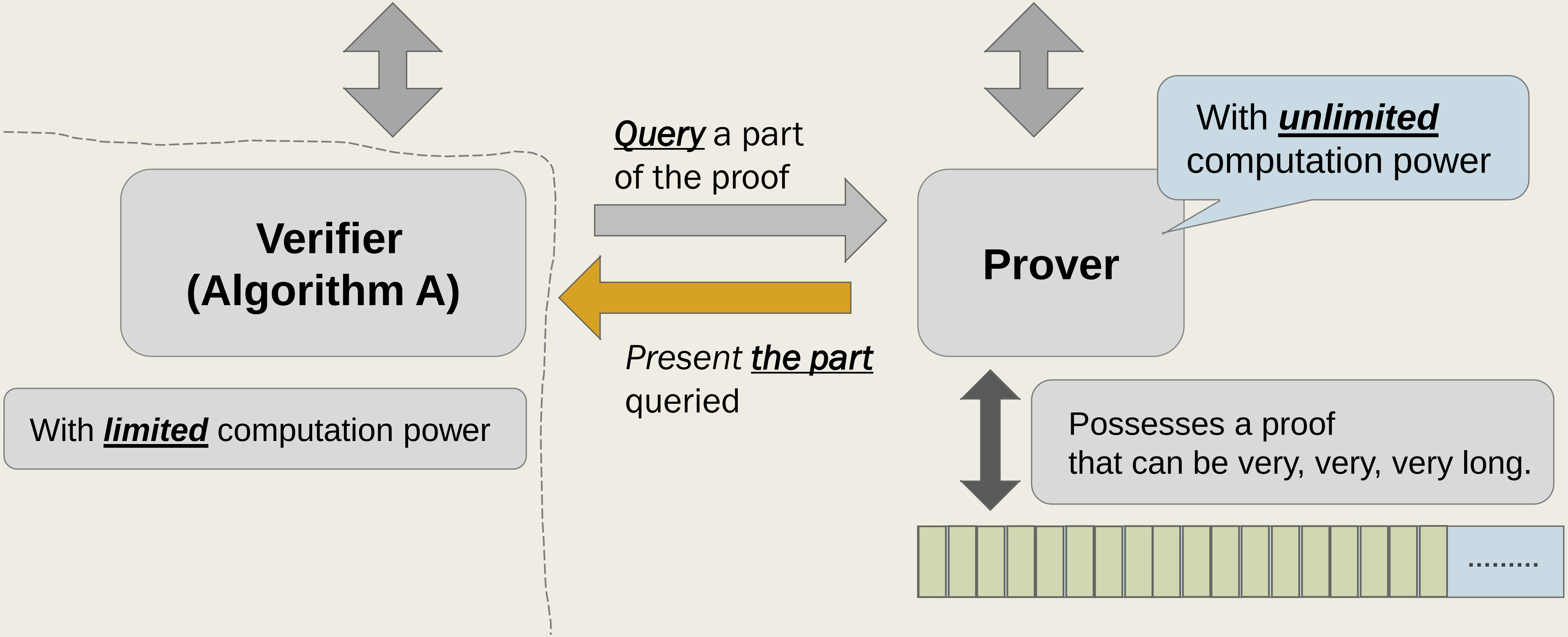
    V uses $O\big(r(n)\big)$ random bits,

    makes $O\big(q(n)\big)$ ***queries*** to a given proof $\boldsymbol{\pi} \in \{\mathbf{0},\mathbf{1}\}^*$, and

    accepts / rejects.

Assume a ***valid proof system*** for Π

that can be *efficiently verified* by a randomized ***algorithm*** $\boldsymbol{A}$.



The Complexity Class PCP(r(n),q(n))

- A language L is in PCP(r(n),q(n))

  if there is a polynomial-time randomized algorithm V such that

  on any input string $x \in \{0,1\}^n$,

  – (Completeness)

    If $x \in L$, then there exists a proof $\pi \in \{0,1\}^*$ such that

    $$\Pr[\, V^{\pi}(x) \text{ accepts} \,] = 1.$$

  – (Soundness)

    If $x \notin L$, then for any $\pi \in \{0,1\}^*$, $\Pr[\, V^{\pi}(x) \text{ accepts} \,] \leq 1/2.$

The PCP Theorem

- The PCP theorem says that,

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

- Every language in NP ***<u>has a proof system</u>*** that can be verified probabilistically using $O(\log n)$ random bits and $O(1)$ queries to the proof.

The PCP Theorem

- The PCP theorem has several equivalent formulations.
  - Probabilistically checkable proofs,
    Graph version,
    Error-correcting code version, etc.

The PCP Theorem (Inapproximability Version)

- **Definition.** (q−CSP)

  An instance of q−CSP consist of a set of alphabet Σ,
  a set of variables $X = \{X_1, \dots, X_n\}$ with $X_i \in \Sigma$, and
  a set of constraints $\phi_1, \dots, \phi_m$, where $\phi_i : X \to \{0,1\}$ depends on
  at most q variables.

The ***value*** of the instance is the *maximum fraction* of the constraints that can be satisfied by any assignment.

For example, vertex cover is a 2-CSP problem.

The PCP Theorem (Inapproximability Version)

- There exist $q \in \mathbb{N}$ and $|\Sigma| > 1$ such that,

  given a q-CSP instance I over alphabet Σ,

  it is NP-hard to distinguish between the two cases:
  - $\mathrm{val}(I) = 1$, or
  - $\mathrm{val}(I) < 1/2$.

- Then, the ratio of the gap corresponds to the hardness of approximating the q-CSP problem.

The PCP Theorem (Inapproximability Version)

- **Definition.** ($\rho-\mathrm{Gap}\ \mathrm{q}-\mathrm{CSP}$)

  Given an instance of $\mathrm{q}-\mathrm{CSP}$ problem, distinguish between the following two cases:

  - $\mathrm{val}(I) = 1$, or
  - $\mathrm{val}(I) < \rho$.

- There exists $q \in \mathbb{N}$ and $\rho \in (0,1)$ such that $\rho-\mathrm{Gap}\ \mathrm{q}-\mathrm{CSP}$ is NP-hard.

The PCP Theorem (Inapproximability Version)

- **Definition.** (Label Cover)

  An instance of label cover consist of $(G = (V_1, V_2, E), \Sigma, \Pi)$, where

  - G is a bipartite graph.
  - For any edge $e \in E$, there is a constraint $\Pi_e: \Sigma \to \Sigma$.

  A labelling of the vertices $\sigma: V \to \Sigma$ is said to satisfy an edge $e = (u, v)$ with $u \in V_1, v \in V_2$ if and only if

$$\Pi_e\big(\sigma(u)\big) = \sigma(v).$$

The ***value*** of the instance is the *maximum fraction* of edges that can be satisfied by any labelling.

The PCP Theorem (Inapproximability Version)

- **Definition.** ($\mathrm{GapLabelCover}_{1,\epsilon}(\Sigma)$)

  Given an instance of I of Label Cover, distinguish between the following two cases:
  - $\mathrm{val}(I) = 1$, or
  - $\mathrm{val}(I) < \epsilon$.

- For any $\epsilon > 0$, there exists a constant $|\Sigma|$ such that $\mathrm{GapLabelCover}_{1,\epsilon}(\Sigma)$ is NP-hard.

Equivalent Views of PCP Theorem

The PCP Theorem

- We have defined the language class $\mathrm{PCP}\big(r(n), q(n)\big)$.

Theorem 1. (PCP Theorem, proof verifying view)

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

The PCP Theorem

Theorem 2. (PCP Theorem, hardness of approximation view)

There exists $\rho < 1$ such that,

for every language $L \in \mathrm{NP}$, there is a polynomial-time mapping

$$f : \{0,1\}^* \mapsto 3CNFs$$

such that

$$x \in L \;\Rightarrow\; \mathrm{val}\big(f(x)\big) = 1$$

$$x \notin L \;\Rightarrow\; \mathrm{val}\big(f(x)\big) < \rho\,.$$

The PCP Theorem

- We have defined the gap version of CSP problems.

Theorem 2. (PCP Theorem, CSP view)

There exists $q \in \mathbb{N}, \rho \in (0,1)$ such that $\rho\mathrm{GAP}q\mathrm{CSP}$ is NP-hard.

Theorem 1 $\Longrightarrow$ Theorem 3

- Suppose that $\mathrm{NP} = \mathrm{PCP}\big(O(\log n), O(1)\big)$.
- It suffices to construct a $\rho\mathrm{GAP}q\mathrm{CSP}$ instance from a PCP verifier V of an NP language, say, 3-SAT.
  - Formulate the execution of V as a CSP constraint.
  - V uses $O(\log n)$ random bits.
    So, at most $poly(n)$ different constraints.
  - V makes $q = O(1)$ random bits.
    Each constraint has arity $q.$

Theorem 1 $\Longrightarrow$ Theorem 3

- It suffices to construct a $\rho\mathrm{GAP}q\mathrm{CSP}$ instance from a PCP verifier V of an NP language, say, 3-SAT.
  - Number of variables = $q \cdot poly(n) = poly(n)$.
  - Hence, the CSP instance has polynomial size.
  - The instance has completeness 1 and soundness $\rho = 1/2$.
- Since 3-SAT is NP-hard,
  the gap instance is NP-hard to decide.

Theorem 3 $\Longrightarrow$ Theorem 1

- It suffices to construct a PCP verifier for $\rho\mathrm{GAP}q\mathrm{CSP}$.
  - The verifier expects the proof to be the assignment of the variables.
  - Pick a constant $c \geq 1$ such that $\rho^c \leq 1/2$.
  - Pick c random constraints and test them.
  - Number of random bits = $c \cdot \log m$.
    Number of queries = $cq = O(1)$.
  - The verifier has completeness 1 and soundness $1/2$.

Mapping of Concepts between Different Views

Proof verifying view		CSP view (hardness of approx.)
PCP verifier V	⟺	CSP instance ϕ
Execution of Verifier	⟺	CSP constraint
Probability that V accepts	⟺	Value of ϕ
Number of random bits r	⟺	Logarithm of **number of constraints** $\log m$

Proof verifying view	CSP view (hardness of approx.)
Length of proof (to be accessed)	Number of variables
PCP proof π	Assignment to variables
Number of queries q	Arity of constraints q
Soundness parameter (usually $1/2$)	Maximum value of any No instance

Proof verifying view

CSP view
(hardness of approx.)

Theorem 1.
$\mathrm{NP} = \mathrm{PCP}\big(O(\log n), O(1)\big)$

Theorem 3.
$\rho\mathrm{GAP}q\mathrm{CSP}$ is NP-hard

Theorem 2.
$\rho\mathrm{GAP}$-3SAT is NP-hard

Corollary.
$(\rho - \epsilon)$-approximation
for Max-3SAT is NP-hard

Theorem 3 $\Longrightarrow$ Theorem 2

- Suppose that ρGAP-3SAT is NP-hard.
- 3SAT is a $q\text{CSP}$ problem with $q = 3$.
  - An algorithm that decides $\rho\text{GAP}q\text{CSP}$ can be used to decide ρGAP-3SAT.
- Hence, $\rho\text{GAP}q\text{CSP}$ must also be NP-hard to decide.

Theorem 2 $\Longrightarrow$ Theorem 3

- Now suppose that $\rho\text{GAP}q\text{CSP}$ is NP-hard.
- Given an instance of $\rho\text{GAP}q\text{CSP}$,
  we construct an instance of $\rho'\text{GAP}$-3SAT with $\rho' = \rho/(q2^q)$.
  - Then, $\rho'\text{GAP}$-3SAT must be NP-hard to decide.

Theorem 2 $\Longrightarrow$ Theorem 3

- First, each CSP constraint, say, $\phi_i = \phi_i(y_1, y_2, \dots, y_q)$, can be transformed to an equivalent q-CNF with at most 2^q clauses.
  - Collect all configurations of $y_1, y_2, \dots, y_q$ that make ϕ_i false.
  - This corresponds to a q-DNF with at most 2^q clauses.
  - Taking negation, we get a q-CNF as claimed.

Theorem 2 $\Longrightarrow$ Theorem 3

- Next, we can apply the Cook-Levin technique to transform the q-CNF into an equivalent 3-CNF.

- Repeat the following two steps until we have a 3-CNF.
  - Pick a clause with size at least 4, say, $y_1 \vee y_2 \vee \phi'$, where $|\phi'| \geq 2$.
  - Add a new variable z and replace the clause with
    $$(y_1 \vee y_2 \vee z) \wedge (\bar{z} \vee \phi') .$$

The number of literals is decreased by 1.

The number of variables and clauses are increased by 1.

- Repeat the following two steps until we have a 3-CNF.
  - Pick a clause with size at least 4,
    say, $\phi' = y_1 \lor y_2 \lor \phi''$, where $|\phi'| \geq 2$.
  - Introduce a new variable z and replace ϕ' with

$$(y_1 \lor y_2 \lor z) \land (\bar{z} \lor \phi'') .$$

The number of literals is decreased by 1.

The number of variables and clauses are increased by 1.

- If ϕ' is satisfied, then there exists $z \in \{0,1\}$ such that $(y_1 \lor y_2 \lor z) \land (\bar{z} \lor \phi'')$ is satisfied.
- If ϕ' is not satisfied, then no $z \in \{0,1\}$ can simultaneously satisfy $(y_1 \lor y_2 \lor z)$ and $(\bar{z} \lor \phi'')$.

Theorem 2 $\Longrightarrow$ Theorem 3

- Next, we can apply the Cook-Levin technique to transform the q-CNF into an equivalent 3-CNF.

- From the q-CNF with n variables and $2^q m$ clauses,
  we obtain a 3-CNF with $n + qm$ variables and $q2^q m$ clauses.
  - The completeness is 1.
  - Each unsatisfied clause in q-CNF results in at least one unsatisfied clause in 3-CNF.
  - The soundness is $\rho' = \rho/(q2^q)$.