co-NP, Self-Reducions

Lecture 24
April 29, 2014
Find this path

G: A directed graph $G$ with weights on edges, vertices $s$ and $t$, and a number $\Delta \geq 0$. Consider the following problems, that all ask if there exists a path $\pi$ between $s$ and $t$ with a desired property.

We have that

1. **ShortestP:** $\text{length}(\pi) \leq \Delta$.
2. **ExactP:** $\text{length}(\pi) = \Delta$.
3. **LongestP:** $\text{length}(\pi) \geq \Delta$.

(A) ShortestP $\in P$, ExactP $\in P$, LongestP $\in P$.
(B) ShortestP $\in P$, ExactP $\in P$, LongestP $\in NPC$.
(C) ShortestP $\in NPC$, ExactP $\in P$, LongestP $\in P$.
(D) ShortestP $\in P$, ExactP $\in NPC$, LongestP $\in NPC$.
(E) ShortestP $\in NPC$, ExactP $\in NPC$, LongestP $\in NPC$. 
Part I

Complementation and Self-Reduction
A language $L$ (equivalently decision problem) is in the class $P$ if there is a polynomial time algorithm $A$ for deciding $L$; that is given a string $x$, $A$ correctly decides if $x \in L$ and running time of $A$ on $x$ is polynomial in $|x|$, the length of $x$. 
The class **NP**

Two equivalent definitions:

1. **Language** $L$ is in **NP** if there is a non-deterministic polynomial time algorithm $A$ (Turing Machine) that decides $L$.
   - For $x \in L$, $A$ has some non-deterministic choice of moves that will make $A$ accept $x$
   - For $x \not\in L$, no choice of moves will make $A$ accept $x$

2. $L$ has an efficient certifier $C(\cdot, \cdot)$.
   - $C$ is a polynomial time deterministic algorithm
   - For $x \in L$ there is a string $y$ (proof) of length polynomial in $|x|$ such that $C(x, y)$ accepts
   - For $x \not\in L$, no string $y$ will make $C(x, y)$ accept
Let $\mathbf{X}$ be an $\textbf{NP-Complete}$ problem, such that for any YES instance $I$ of $\mathbf{X}$ of length $n$, there exists a certificate of length $O(\log n)$ that it is correct.

If such a problem $\mathbf{X}$ exists then

(A) Nothing at all is implied by this.
(B) $P = \text{NP}$.
(C) $P \neq \text{NP}$. 
Complementation

**Definition**

Given a decision problem \( X \), its **complement** \( \overline{X} \) is the collection of all instances \( s \) such that \( s \not\in L(X) \).

Equivalently, in terms of languages:

**Definition**

Given a language \( L \) over alphabet \( \Sigma \), its **complement** \( \overline{L} \) is the language \( \Sigma^* \setminus L \).
Examples

1. \( \text{PRIME} = \{ n \mid \text{n is an integer and } n \text{ is prime} \} \)
   \( \text{PRIME} = \{ n \mid n \text{ is an integer and } n \text{ is not a prime} \} \)
   \( \text{PRIME} = \text{COMPOSITE}. \)

2. \( \text{SAT} = \{ \varphi \mid \varphi \text{ is a CNF formula and } \varphi \text{ is satisfiable} \} \)
   \( \text{SAT} = \{ \varphi \mid \varphi \text{ is a CNF formula and } \varphi \text{ is not satisfiable} \} \).
   \( \text{SAT} = \text{UnSAT}. \)
Examples

1. \( \text{PRIME} = \{ n \mid n \text{ is an integer and } n \text{ is prime} \} \)
   \( \text{PRIME} = \{ n \mid n \text{ is an integer and } n \text{ is not a prime} \} \)
   \( \text{PRIME} = \text{COMPOSITE} \).

2. \( \text{SAT} = \{ \varphi \mid \varphi \text{ is a CNF formula and } \varphi \text{ is satisfiable} \} \)
   \( \text{SAT} = \{ \varphi \mid \varphi \text{ is a CNF formula and } \varphi \text{ is not satisfiable} \} \).
   \( \text{SAT} = \text{UnSAT} \).

**Technicality:** \( \text{SAT} \) also includes strings that do not encode any valid \text{CNF} formula. Typically we ignore those strings because they are not interesting. In all problems of interest, we assume that it is "easy" to check whether a given string is a valid instance or not.
Proposition

Decision problem $X$ is in $P$ if and only if $\bar{X}$ is in $P$.

Proof.

1. If $X$ is in $P$ let $A$ be a polynomial time algorithm for $X$.
2. Construct polynomial time algorithm $A'$ for $\bar{X}$ as follows: given input $x$, $A'$ runs $A$ on $x$ and if $A$ accepts $x$, $A'$ rejects $x$ and if $A$ rejects $x$ then $A'$ accepts $x$.
3. Only if direction is essentially the same argument.
Asymmetry of \textbf{NP}

**Definition**

Nondeterministic Polynomial Time (denoted by \textbf{NP}) is the class of all problems that have efficient certifiers.

**Observation**

To show that a problem is in \textbf{NP} we only need short, efficiently checkable certificates for “yes”-instances. What about “no”-instances?
Asymmetry of \textbf{NP}

**Definition**

Non-deterministic Polynomial Time (denoted by \textbf{NP}) is the class of all problems that have efficient certifiers.

**Observation**

To show that a problem is in \textbf{NP} we only need short, efficiently checkable certificates for “yes”-instances. What about “no”-instances?

Given a \textbf{CNF} formula \( \varphi \), is \( \varphi \) unsatisfiable?
Asymmetry of $\textbf{NP}$

**Definition**

Nondeterministic Polynomial Time (denoted by $\textbf{NP}$) is the class of all problems that have efficient certifiers.

**Observation**

To show that a problem is in $\textbf{NP}$ we only need short, efficiently checkable certificates for “yes”-instances. What about “no”-instances?

Given a $\textbf{CNF}$ formula $\varphi$, is $\varphi$ unsatisfiable?

Easy to give a proof that $\varphi$ is satisfiable (an assignment) but no easy (known) proof to show that $\varphi$ is unsatisfiable!
Examples of complement problems

Some languages

1. **UnSAT**: CNF formulas $\varphi$ that are not satisfiable
2. **No-Hamilton-Cycle**: graphs $G$ that do not have a Hamilton cycle
3. **No-3-Color**: graphs $G$ that are not 3-colorable
Examples of complement problems

Some languages

1. **UnSAT**: CNF formulas \( \varphi \) that are not satisfiable
2. **No-Hamilton-Cycle**: graphs \( G \) that do not have a Hamilton cycle
3. **No-3-Color**: graphs \( G \) that are not 3-colorable

Above problems are complements of known \( \text{NP} \) problems (viewed as languages).
NP and co-NP

**NP**

Decision problems with a polynomial certifier.
Examples: SAT, Hamiltonian Cycle, 3-Colorability.

**Definition**

co-NP is the class of all decision problems $X$ such that $\overline{X} \in \text{NP}$.
Examples: UnSAT, No-Hamiltonian-Cycle, No-3-Colorable.
If $L$ is a language in \textbf{co-NP} then there is a polynomial time certifier/verifier $C(\cdot, \cdot)$, such that:

1. for $s \not\in L$ there is a proof $t$ of size polynomial in $|s|$ such that $C(s, t)$ correctly says NO.

2. for $s \in L$ there is no proof $t$ for which $C(s, t)$ will say NO.
If $L$ is a language in co-NP then that there is a polynomial time certifier/verifier $C(\cdot, \cdot)$, such that:

1. for $s \not\in L$ there is a proof $t$ of size polynomial in $|s|$ such that $C(s, t)$ correctly says NO.
2. for $s \in L$ there is no proof $t$ for which $C(s, t)$ will say NO

co-NP has checkable proofs for strings NOT in the language.
1. **Tautology**: given a Boolean formula (not necessarily in CNF form), is it true for *all* possible assignments to the variables?

2. **Graph expansion**: given a graph $G$, is it an *expander*? A graph $G = (V, E)$ is an *expander* if and only if for each $S \subseteq V$ with $|S| \leq |V|/2$, $|N(S)| \geq |S|$. Expanders are very important graphs in theoretical computer science and mathematics.
Problem: Primality

Instance: An integer $n$.
Question: Is the number $n$ prime?

Problem: Factoring

Instance: Integers $n$, $k$.
Question: Does the number $n$ have a factor $\leq k$?
Formally, is there $\ell$, such that $2 \leq \ell \leq k$, such that $\ell$ divides $n$?

1. Primality is in $P$.
2. Factoring is in $NP \cap co-NP$. 
Factoring is a very naughty problem

**Problem:** Factoring

**Instance:** Integers $n, k$.

**Question:** Does the number $n$ has a factor $\leq k$? Formally, is there $\ell$, such that $2 \leq \ell \leq k$, such that $\ell$ divides $n$?

If answer is:

1. **NO:** certificate is all prime factors of $n$. Certification: multiply the given numbers.

2. **YES:** Certificate is the factor $\ell$. Verify it divides $n$.

**Belief:** Unlikely Factoring is NP-Complete. Can be solved in polynomial time on a quantum computer.
co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-

co-
co-P: complement of P. Language $X$ is in co-P iff $\overline{X} \in P$

Proposition

$P = \text{co-P}$.
co-P: complement of P. Language $X$ is in co-P iff $\overline{X} \in P$

**Proposition**

$P = \text{co-P}$.

**Proposition**

$P \subseteq \text{NP} \cap \text{co-NP}$. 
**P, NP, co-NP**

**co-P**: complement of P. Language $X$ is in co-P iff $\overline{X} \in P$

**Proposition**

$P = co-P$.

**Proposition**

$P \subseteq NP \cap co-NP$.

Saw that $P \subseteq NP$. Same proof shows $P \subseteq co-NP$. 
P, NP, and co-NP

Open Problems:
1. Does \( NP = co-NP \)?
Open Problems:

1. Does $NP = co-NP$?
   
   Consensus opinion: No.
Open Problems:
1. Does $NP = co-NP$?
   Consensus opinion: No.
2. Is $P = NP \cap co-NP$?
Open Problems:

1. Does \( \text{NP} = \text{co-NP} \)?
   
   Consensus opinion: No.

2. Is \( \text{P} = \text{NP} \cap \text{co-NP} \)?
   
   No real consensus.
Proposition

If $P = NP$ then $NP = co-NP$. 
Proposition

If $P = NP$ then $NP = co-NP$.

Proof.

$P = co-P$

If $P = NP$ then $co-NP = co-P = P$. 
P, NP, and co-NP

We know that:

Proposition

If $P = NP$ then $NP = co-NP$.

As such:

(A) If $P \neq NP$ then $NP \neq co-NP$.

(B) If $NP \neq co-NP$ then $P \neq NP$.

(C) If $NP = co-NP$ then $P \neq NP$.

(D) If $NP \neq co-NP$ then $P = NP$.

(E) None of the above.
P, NP, and co-NP

Which means that...

**Corollary**

If NP $\neq$ co-NP then $P \neq NP$. 
Corollary

If $\text{NP} \neq \text{co-NP}$ then $\text{P} \neq \text{NP}$.

Importance of corollary: try to prove $\text{P} \neq \text{NP}$ by proving that $\text{NP} \neq \text{co-NP}$. 

Which means that...
Complexity Class \( \text{NP} \cap \text{co-NP} \)

Problems in this class have

1. Efficient certifiers for yes-instances
2. Efficient disqualifiers for no-instances

Problems have a good characterization property, since for both yes and no instances we have short efficiently checkable proofs.
Example

**Bipartite Matching**: Given bipartite graph $G = (U \cup V, E)$, does $G$ have a perfect matching?

**Bipartite Matching $\in \mathcal{NP} \cap \mathcal{co-NP}$**
Example

Bipartite Matching: Given bipartite graph $G = (U \cup V, E)$, does \( G \) have a perfect matching?

Bipartite Matching $\in \text{NP} \cap \text{co-NP}$

1. If $G$ is a yes-instance, then proof is just the perfect matching.

Example (More interesting...)

Factoring $\in \text{NP} \cap \text{co-NP}$, and we do not know if it is in $P$!
Example

**Bipartite Matching**: Given bipartite graph $G = (U \cup V, E)$, does $G$ have a perfect matching?

**Bipartite Matching $\in \text{NP} \cap \text{co-NP}$**

1. If $G$ is a yes-instance, then proof is just the perfect matching.
2. If $G$ is a no-instance, then by Hall’s Theorem, there is a subset of vertices $A \subseteq U$ such that $|N(A)| < |A|$.

Example (More interesting...)

**Factoring $\in \text{NP} \cap \text{co-NP}$**, and we do not know if it is in $\text{P}$!
Bipartite Matching has a polynomial time algorithm

Do all problems in $\text{NP} \cap \text{co-NP}$ have polynomial time algorithms? That is, is $\text{P} = \text{NP} \cap \text{co-NP}$?
Bipartite Matching has a polynomial time algorithm

Do all problems in $\text{NP} \cap \text{co-NP}$ have polynomial time algorithms? That is, is $\text{P} = \text{NP} \cap \text{co-NP}$?

Problems in $\text{NP} \cap \text{co-NP}$ have been proved to be in $\text{P}$ many years later

- Linear programming (Khachiyan 1979)
  - Duality easily shows that it is in $\text{NP} \cap \text{co-NP}$

- Primality Testing (Agarwal-Kayal-Saxena 2002)
  - Easy to see that PRIME is in co-NP (why?)
  - PRIME is in NP - not easy to show! (Vaughan Pratt 1975)
Some problems in $\text{NP} \cap \text{co-NP}$ still cannot be proved to have polynomial time algorithms

1. Parity Games.
2. Other more specialized problems.
A problem $X$ is said to be co-NP-Complete (co-NPC) if
1. $X \in \text{co-NP}$
2. (Hardness) For any $Y \in \text{co-NP}$, $Y \leq_P X$
co-NP Completeness

Definition

A problem $X$ is said to be co-NP-Complete (co-NPC) if

1. $X \in \text{co-NP}$
2. (Hardness) For any $Y \in \text{co-NP}$, $Y \leq_P X$

co-NP-Complete problems are the hardest problems in co-NP.
co-NP Completeness

Definition
A problem $X$ is said to be co-NP-Complete (co-NPC) if
1. $X \in \text{co-NP}$
2. (Hardness) For any $Y \in \text{co-NP}$, $Y \leq_P X$

co-NP-Complete problems are the hardest problems in co-NP.

Lemma
$X$ is co-NPC if and only if $\overline{X}$ is NP-Complete.

Proof left as an exercise.
coNP-Complete?

Every co-NP-Complete problem has an exponential time algorithm; that is, an algorithm that runs in $2^{p(n)}$ time on an input of size $n$ for some fixed polynomial $p(\cdot)$.

(A) True.
(B) False.
(C) True only if $P = NP$.
(D) True only if $P \neq NP$. 
P, NP and co-NP

Possible scenarios:

1. P = NP. Then P = NP = co-NP.
2. NP = co-NP and P \neq NP (and hence also P \neq co-NP).
3. NP \neq co-NP. Then P \neq NP and also P \neq co-NP.

Most people believe that the last scenario is the likely one.
P, NP and co-NP

Possible scenarios:

1. \( P = \text{NP} \). Then \( P = \text{NP} = \text{co-NP} \).
2. \( \text{NP} = \text{co-NP} \) and \( P \neq \text{NP} \) (and hence also \( P \neq \text{co-NP} \)).
3. \( \text{NP} \neq \text{co-NP} \). Then \( P \neq \text{NP} \) and also \( P \neq \text{co-NP} \).

Most people believe that the last scenario is the likely one.

**Question:** Suppose \( P \neq \text{NP} \). Is every problem that is in \( \text{NP} \setminus P \) is also \text{NP-Complete}?
Possible scenarios:

1. $P = NP$. Then $P = NP = \text{co-NP}$.
2. $NP = \text{co-NP}$ and $P \neq NP$ (and hence also $P \neq \text{co-NP}$).
3. $NP \neq \text{co-NP}$. Then $P \neq NP$ and also $P \neq \text{co-NP}$.

Most people believe that the last scenario is the likely one.

Question: Suppose $P \neq NP$. Is every problem that is in $NP \setminus P$ also $NP$-Complete?

**Theorem (Ladner)**

*If $P \neq NP$ then there is a problem/language $X \in NP \setminus P$ such that $X$ is not $NP$-Complete.*
Question: Why restrict to Karp reductions for NP-Completeness?

Lemma

If $X \in \text{co-NP}$ and $Y$ is NP-Complete then $X \leq_P Y$ under Turing reduction.

Thus, Turing reductions cannot distinguish NP and co-NP.
Recall, decision problems are those with yes/no answers, while search problems require an explicit solution for a yes instance.

Example

1. **Satisfiability**
   - **Decision:** Is the formula $\varphi$ satisfiable?
   - **Search:** Find assignment that satisfies $\varphi$

2. **Graph coloring**
   - **Decision:** Is graph $G$ 3-colorable?
   - **Search:** Find a 3-coloring of the vertices of $G$
1. Efficient algorithm for search implies efficient algorithm for decision.

2. If decision problem is difficult then search problem is also difficult.

3. Can an efficient algorithm for decision imply an efficient algorithm for search?
   Yes, for all the problems we have seen. In fact for all NP-Complete Problems.
A problem is said to be **self reducible** if the search problem reduces (by Turing reduction) in polynomial time to decision problem. In other words, there is an algorithm to solve the search problem that has polynomially many steps, where each step is either

1. A conventional computational step, or
2. a call to subroutine solving the decision problem.
Max 3SAT...

**Problem: MAX 3SAT**

**Instance:** \( F: 3\text{CNF} \) Formula with \( n \) variables and \( m \) clauses.

**Question:** Find assignment satisfying as many clauses of \( F \) as possible.

(A) MAX 3SAT is NP-Complete.
(B) MAX 3SAT is NP-Hard.
(C) MAX 3SAT is P.
(D) MAX 3SAT is not a decision problem.
Proposition

\textbf{SAT} is \textit{self reducible}.

In other words, there is a polynomial time algorithm to find the satisfying assignment if one can periodically check if some formula is satisfiable.
Search Algorithm for SAT given a Decision Algorithm for SAT

Input: SAT formula \( \varphi \) with \( n \) variables \( x_1, x_2, \ldots, x_n \).

1. set \( x_1 = 0 \) in \( \varphi \) and get new formula \( \varphi_1 \). check if \( \varphi_1 \) is satisfiable using decision algorithm. if \( \varphi_1 \) is satisfiable, recursively find assignment to \( x_2, x_3, \ldots, x_n \) that satisfy \( \varphi_1 \) and output \( x_1 = 0 \) along with the assignment to \( x_2, \ldots, x_n \).

2. if \( \varphi_1 \) is not satisfiable then set \( x_1 = 1 \) in \( \varphi \) to get formula \( \varphi_2 \). if \( \varphi_2 \) is satisfiable, recursively find assignment to \( x_2, x_3, \ldots, x_n \) that satisfy \( \varphi_2 \) and output \( x_1 = 1 \) along with the assignment to \( x_2, \ldots, x_n \).

3. if \( \varphi_1 \) and \( \varphi_2 \) are both not satisfiable then \( \varphi \) is not satisfiable.

Algorithm runs in polynomial time if the decision algorithm for SAT runs in polynomial time. At most \( 2^n \) calls to decision algorithm.
Theorem

Every NP-Complete problem/language $L$ is self-reducible.

Proof is not hard but requires understanding of proof of Cook-Levin theorem.

Note that proof is only for complete languages, not for all languages in NP. Otherwise Factoring would be in polynomial time and we would not rely on it for our current security protocols.

Easy and instructive to prove self-reducibility for specific NP-Complete problems such as Independent Set, Vertex Cover, Hamiltonian Cycle, etc. See discussion section problems.