CS 473: Algorithms

Chandra Chekuri
chekuri@cs.illinois.edu
3228 Siebel Center

University of Illinois, Urbana-Champaign

Fall 2010
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:

- 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 $x \notin L$, no choice of moves will make $A$ accept $x$

- $L$ 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 $x \notin L$, no string $y$ will make $C(x, y)$ accept
Complementation

**Definition**

Given a decision problem \( X \), its complement \( \bar{X} \) is the collection of all instances \( s \) such that \( s \notin X \).

Equivalently, in terms of languages:

**Definition**

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

- $\overline{\text{PRIME}} = \{n \mid n \text{ is an integer and } n \text{ is prime}\}$
  $\overline{\text{PRIME}} = \{n \mid n \text{ is an integer and } n \text{ is not a prime}\}$
  $\overline{\text{PRIME}} = \text{COMPOSITE}$

- $\overline{\text{SAT}} = \{\varphi \mid \varphi \text{ is a SAT formula and } \varphi \text{ is satisfiable}\}$
  $\overline{\text{SAT}} = \{\varphi \mid \varphi \text{ is a SAT formula and } \varphi \text{ is not satisfiable}\}$
  $\overline{\text{SAT}} = \text{UnSAT}$
Examples

- $PRIME = \{ n \mid n \text{ is an integer and } n \text{ is prime} \}$
  - $PRIME = \{ n \mid n \text{ is an integer and } n \text{ is not a prime} \}$
  - $PRIME = COMPOSITE$

- $SAT = \{ \varphi \mid \varphi \text{ is a SAT formula and } \varphi \text{ is satisfiable} \}$
  - $\overline{SAT} = \{ \varphi \mid \varphi \text{ is a SAT formula and } \varphi \text{ is not satisfiable} \}$
  - $\overline{SAT} = UnSAT$

**Technicality:** $\overline{SAT}$ also includes strings that do not encode any valid SAT 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 $\overline{X}$ is in $P$. 
Proposition

Decision problem \( X \) is in \( P \) if and only if \( \overline{X} \) is in \( P \).

Proof.

- If \( X \) is in \( P \) let \( A \) be a polynomial time algorithm for \( X \).
- Construct polynomial time algorithm \( A' \) for \( \overline{X} \) 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 \( x \).
- Only if direction is essentially the same argument.
Asymmetry of $NP$

**Definition**

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

**Observation**

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

**Definition**

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

**Observation**

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

Given CNF formulat $\varphi$, is $\varphi$ unsatisfiable?
Asymmetry of \( NP \)

### Definition

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

### Observation

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

given CNF formulat \( \varphi \), 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

Some languages

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

Some languages

- UnSAT: CNF formulas $\varphi$ that are not satisfiable
- No-Hamilton-Cycle: graphs $G$ that do not have a Hamilton cycle
- No-3-Color: graphs $G$ that are not 3-colorable

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

NP

Decision problems with a polynomial certifier. Examples, SAT, Hamiltonian Cycle, 3-Colorability
**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 $\overline{X} \in NP$. Examples, UnSAT, No-Hamiltonian-Cycle, No-3-Colorable.
$L$ is a language in co-NP implies that there is a polynomial time certifier/verifier $C(\cdot, \cdot)$ such that

- for $s \notin L$ there is a proof $t$ of size polynomial in $|s|$ such that $C(s, t)$ correctly says NO
- for $s \in L$ there is no proof $t$ for which $C(s, t)$ will say NO
co-NP

$L$ is a language in co-NP implies that there is a polynomial time certifier/verifier $C(\cdot, \cdot)$ such that

- for $s \notin L$ there is a proof $t$ of size polynomial in $|s|$ such that $C(s, t)$ correctly says NO
- 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.
Natural Problems in co-NP

- **Tautology**: given a Boolean formula (not necessarily in CNF form), is it true for *all* possible assignments to the variables?

- **Graph expansion**: given a graph $G$, is it an *expander*? A graph $G = (V, E)$ is an expander iff 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.
co-P: complement of P. Language $X$ is in co-P iff $\overline{X} \in P$
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 $\bar{X} \in P$

**Proposition**

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

**Proposition**

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

**Proposition**

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

**Proposition**

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

Saw that $P \subseteq NP$. Same proof shows $P \subseteq \text{co-NP}$
Complementation
Self Reduction
Recap
Motivation
co-NP Definition
Relationship between $P$, $NP$ and co-$NP$

$P$, $NP$, and co-$NP$

Open Problems:
- Does $NP = co-NP$?
**P, NP, and co-NP**

Open Problems:
- Does $NP = \text{co-NP}$? **Consensus opinion: No**
- Is $P = NP \cap \text{co-NP}$?
Open Problems:

- Does $NP = \text{co-NP}$? **Consensus opinion:** No
- Is $P = NP \cap \text{co-NP}$? 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* $co-NP = co-P = P$. 

---

**Corollary**

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

**Importance of corollary:** Try to prove $P \neq NP$ by proving that $NP \neq co-NP$. 

---

**Chekuri**

CS473
Proposition

If \( P = NP \) then \( NP = co-NP \).

Proof.

\( P = co-P \)

If \( P = NP \) then \( co-NP = co-P = P \).

Corollary

If \( NP \neq co-NP \) then \( P \neq NP \).
**Proposition**

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

**Proof.**

$P = co-P$

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

**Corollary**

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

Importance of corollary: try to prove $P \neq NP$ by proving that $NP \neq co-NP$. 
**Complexity Class** $NP \cap co-NP$

Problems in this class have

- Efficient certifiers for yes-instances
- 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 $\in NP \cap \text{co-NP}$
**Example**

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

Bipartite Matching $\in NP \cap co-NP$

- If $G$ is a yes-instance, then proof is just the perfect matching.
**NP \cap \text{co-NP}: Example**

**Example**

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

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

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


Good Characterization \(\equiv\) Efficient Solution

- Bipartite Matching has a polynomial time algorithm
- Do all problems in \(NP \cap co-NP\) have polynomial time algorithms? That is, is \(P = NP \cap co-NP\)?
Bipartite Matching has a polynomial time algorithm.

Do all problems in \( NP \cap co-NP \) have polynomial time algorithms? That is, is \( P = NP \cap co-NP \)?

Problems in \( NP \cap co-NP \) have been proved to be in \( P \) many years later:

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

- Some problems in $NP \cap \text{co-NP}$ still cannot be proved to have polynomial time algorithms
$P \subseteq NP \cap \text{co-NP}$ (contd)

- Some problems in $NP \cap \text{co-NP}$ still cannot be proved to have polynomial time algorithms
  - Parity Games
Some problems in $NP \cap co-NP$ still cannot be proved to have polynomial time algorithms
- Parity Games
- Other more specialized problems
Definition

A problem $X$ is said to be *co-NP-complete* if

- $X \in \text{co-NP}$
- *(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* 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-NP-Complete if and only if $\overline{X}$ is NP-Complete.

Proof left as an exercise.
co-NP Completeness

Definition
A problem $X$ is said to be \textit{co-NP-complete} if
- $X \in \text{co-NP}$
- (Hardness) For any $Y \in \text{co-NP}$, $Y \leq_P X$

\textit{co-NP-Complete} problems are the hardest problems in \textit{co-NP}.

Lemma

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

Proof left as an exercise.
$P$, $NP$ and $co-NP$

Possible scenarios:

- $P = NP$. Then $P = NP = co-NP$.
- $NP = co-NP$ and $P \neq NP$ (and hence also $P \neq co-NP$).
- $NP \neq co-NP$. 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:

- $P = NP$. Then $P = NP = co-NP$.
- $NP = co-NP$ and $P \neq NP$ (and hence also $P \neq co-NP$).
- $NP \neq co-NP$. Then $P \neq NP$ and also $P \neq co-NP$.

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

**Question:** Suppose $P \neq NP$. Is every problem in $NP - P$ $NP$-Complete?
Possible scenarios:

- $P = NP$. Then $P = NP = co-NP$.
- $NP = co-NP$ and $P \neq NP$ (and hence also $P \neq co-NP$).
- $NP \neq co-NP$. Then $P \neq NP$ and also $P \neq co-NP$.

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

**Question:** Suppose $P \neq NP$. Is every problem in $NP - P$ NP-Complete?

**Theorem (Ladner)**

If $P \neq NP$ then there is a problem/language $X \in NP - P$ such that $X$ is not NP-Complete.
Back to Decision versus Search

- Recall, decision problems are those with yes/no answers, while search problems require an explicit solution for a yes instance.
Recall, decision problems are those with yes/no answers, while search problems require an explicit solution for a yes instance.

Example
Recall, decision problems are those with yes/no answers, while search problems require an explicit solution for a yes instance.

Example

- Satisfiability
Recall, decision problems are those with yes/no answers, while search problems require an explicit solution for a yes instance.

**Example**

- **Satisfiability**
  - **Decision:** Is the formula $\varphi$ satisfiable?
Recall, decision problems are those with yes/no answers, while search problems require an explicit solution for a yes instance.

**Example**

- **Satisfiability**
  - **Decision:** Is the formula $\varphi$ satisfiable?
  - **Search:** Find assignment that satisfies $\varphi$
Recall, decision problems are those with yes/no answers, while search problems require an explicit solution for a yes instance.

**Example**

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

- **Graph coloring**
Recall, decision problems are those with yes/no answers, while search problems require an explicit solution for a yes instance.

**Example**

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

- **Graph coloring**
  - **Decision:** Is graph $G$ 3-colorable?
Recall, decision problems are those with yes/no answers, while search problems require an explicit solution for a yes instance.

**Example**

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

- **Graph coloring**
  - **Decision**: Is graph $G$ 3-colorable?
  - **Search**: Find a 3-coloring of the vertices of $G$
Decision “reduces to” Search

- Efficient algorithm for search implies efficient algorithm for decision
Decision “reduces to” Search

- Efficient algorithm for search implies efficient algorithm for decision
- If decision problem is difficult then search problem is also difficult
Decision “reduces to” Search

- Efficient algorithm for search implies efficient algorithm for decision
- If decision problem is difficult then search problem is also difficult
- Can an efficient algorithm for decision imply an efficient algorithm for search?
Decision “reduces to” Search

- Efficient algorithm for search implies efficient algorithm for decision
- If decision problem is difficult then search problem is also difficult
- 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.
\[ \varphi = (x_1 \lor x_2 \lor \overline{x}_3) \land (\overline{x}_1 \lor \overline{x}_2 \lor x_4) \land (\overline{x}_1 \lor \overline{x}_2 \lor x_5) \]

\[ \varphi'(x_1=1) = (\overline{x}_2 \lor x_4) \land (\overline{x}_2 \lor x_5) \]
Self Reduction

Definition

A problem is said to be self reducible if the search problem reduces (by Cook 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

- A conventional computational step, or
- A call to subroutine solving the decision problem
Proposition

*SAT is 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 from a Decision Algorithm for SAT

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

- 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 \( x_2, \ldots, x_n \).
- 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 \( x_2, \ldots, x_n \).
- 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 \( 2n \) calls to decision algorithm.
Self-Reduction for NP-Complete Problems

Theorem

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

Proof out of scope.

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 HBS problems and finals prep.