Lecture 17: Finish NP-Completeness, coNP and Friends
**Definition:** A language B is **NP-complete** if:

1. \( B \in \text{NP} \)

2. Every A in NP is poly-time reducible to B
   That is, \( A \leq_p B \)
   When this is true, we say “B is NP-hard”

---

**Last time:** We showed

\( 3\text{SAT} \leq_p \text{CLIQUE} \leq_p \text{IS} \leq_p \text{VC} \leq_p \text{SUBSET-SUM} \leq_p \text{KNAPSACK} \)

All of them are in NP, and 3SAT is NP-complete, so all of these problems are NP-complete!
The Knapsack Problem

Input: \( S = \{(v_1, c_1), \ldots, (v_n, c_n)\} \) of pairs of positive integers (items)

- a capacity budget \( C \)
- a value target \( V \)

Decide: Is there an \( S' \subseteq \{1, \ldots, n\} \) such that

\[
\sum_{i \in S'} v_i \geq V \quad \text{and} \quad \sum_{i \in S'} c_i \leq C?
\]

Define: \( \text{KNAPSACK} = \{(S, C, V) \mid \text{the answer is yes}\} \)

A classic economics/logistics/OR problem!

Theorem: \( \text{KNAPSACK} \) is NP-complete
KNAPSACK is NP-complete

KNAPSACK is in NP?

Theorem: SUBSET-SUM \( \leq_p \) KNAPSACK

Proof: Given an instance \((S = \{a_1, \ldots, a_n\}, t)\) of SUBSET-SUM, create a KNAPSACK instance:

1. For all \(i\), set \((v_i, c_i) := (a_i, a_i)\)
2. Define \(T = \{(v_1, c_1), \ldots, (v_n, c_n)\}\)
3. Define \(C := V := t\)

Then, \((S, t) \in \text{SUBSET-SUM} \iff (T, C, V) \in \text{KNAPSACK}\)

Subset of \(S\) that sums to \(t = \) Solution to the Knapsack instance!
The Partition Problem

Input: Set $S = \{a_1, \ldots, a_n\}$ of positive integers

Decide: Is there an $S' \subseteq S$ where $(\sum_{i \in S'} a_i) = (\sum_{i \in S-S'} a_i)$?

(Formally: PARTITION is the set of all encodings of sets $S$ such that the answer to the question is yes.)

In other words, is there a way to partition $S$ into two parts, so that both parts have equal sum?

A problem in Fair Division:
Think of $a_i$ as “value” of item $i$. Want to divide a set of items into two parts $S'$ and $S-S'$, of the same total value. Give $S'$ to one party, and $S-S'$ to the other.

Theorem: PARTITION is NP-complete
PARTITION is NP-complete

(1) PARTITION is in NP

(2) SUBSET-SUM \leq_p PARTITION

Input: Set \( S = \{a_1, \ldots, a_n\} \) of positive integers
positive integer \( t \)

Reduction: If \( t > \sum_i a_i \) then output \( \{1,2\} \)
Else output \( T := \{a_1, \ldots, a_n, 2A-t, A+t\} \), where \( A := \sum_i a_i \)

Claim: \( (S,t) \in \text{SUBSET-SUM} \iff T \in \text{PARTITION} \)
That is, \( S \) has a subset that sums to \( t \)
\( \iff T \) can be partitioned into two sets with equal sums

Easy case: \( t > \sum_i a_i \)
Input: Set $S = \{a_1, \ldots, a_n\}$ of positive integers, positive $t$

Output: $T := \{a_1, \ldots, a_n, 2A-t, A+t\}$, where $A := \sum_i a_i$

**Claim:** $(S,t) \in \text{SUBSET-SUM} \iff T \in \text{PARTITION}$

What’s the sum of all numbers in $T$?  $4A$

**Therefore:** $T \in \text{PARTITION}$

$\iff$ There is a $T' \subseteq T$ that sums to $2A$.

**Proof of** $(S,t) \in \text{SUBSET-SUM} \Rightarrow T \in \text{PARTITION}$:

If $(S,t) \in \text{SUBSET-SUM}$, then let $S' \subseteq S$ sum to $t$.

The set $S' \cup \{2A-t\}$ sums to $2A$, so $T \in \text{PARTITION}$
Input: Set $S = \{a_1, \ldots, a_n\}$ of positive integers, positive $t$

Output: $T := \{a_1, \ldots, a_n, 2A-t, A+t\}$, where $A := \sum_i a_i$

Remember: sum of all numbers in $T$ is $4A$.

Claim: $(S, t) \in \text{SUBSET-SUM} \iff T \in \text{PARTITION}$

$T \in \text{PARTITION} \iff$ There is a $T' \subseteq T$ that sums to $2A$.

Proof of: $T \in \text{PARTITION} \Rightarrow (S, t) \in \text{SUBSET-SUM}$

If $T \in \text{PARTITION}$, let $T' \subseteq T$ be a subset that sums to $2A$. 

Observation: Exactly one of $\{2A-t, A+t\}$ is in $T'$.

If $(2A-t) \in T'$, then $T' - \{2A-t\}$ sums to $t$. By Observation, the set $T' - \{2A-t\}$ is a subset of $S$. So $(S, t) \in \text{SUBSET-SUM}$.

If $(A+t) \in T'$, then $(T - T') - \{2A-t\}$ sums to $(2A - (2A-t)) = t$

By Observation, $(T - T') - \{2A-t\}$ is a subset of $S$. Therefore $(S, t) \in \text{SUBSET-SUM}$ in this case as well.
The Bin Packing Problem

Input: Set \( S = \{a_1, \ldots, a_n\} \) of positive integers, a bin capacity \( B \), and a number of bins \( K \).

Decide: Can \( S \) be partitioned into disjoint subsets \( S_1, \ldots, S_k \) such that each \( S_i \) sums to at most \( B \)?

Think of \( a_i \) as the capacity of item \( i \).

Is there a way to pack the items of \( S \) into \( K \) bins, where each bin has capacity \( B \)?

Ubiquitous problem in shipping and optimization!

Theorem: BIN PACKING is NP-complete
BIN PACKING is NP-complete

(1) BIN PACKING is in NP (Why?)

(2) PARTITION $\leq_p$ BIN PACKING

Proof: Given an instance $S = \{a_1, \ldots, a_n\}$ of PARTITION, output an instance of BIN PACKING with:

$S = \{a_1, \ldots, a_n\}$
$B = (\sum_i a_i)/2$
$k = 2$

Then, $S \in \text{PARTITION} \iff (S,B,k) \in \text{BIN PACKING}$:
There is a partition of $S$ into two equal sums iff there is a solution to this Bin Packing instance!
Two Problems

Let $G$ denote a graph, and $s$ and $t$ denote nodes.

**SHORTEST PATH**

$= \{ (G, s, t, k) \mid G \text{ has a simple path of } < k \text{ edges from } s \text{ to } t \}$

**LONGEST PATH**

$= \{ (G, s, t, k) \mid G \text{ has a simple path of } \geq k \text{ edges from } s \text{ to } t \}$

Are either of these in $P$? Are both of them?
HAMPATH = \{ (G,s,t) \mid G \text{ is a directed graph with a Hamiltonian path from } s \text{ to } t \} 

**Theorem:** HAMPATH is NP-Complete

(1) HAMPATH ∈ NP

(2) 3SAT \leq_p HAMPATH

Sipser (p.314-318) and recitation!
HAMPATH $\leq_p$ LONGEST-PATH

LONGEST-PATH
= \{(G, s, t, k) \mid G \text{ has a simple path of } \geq k \text{ edges from } s \text{ to } t \} \\
Can reduce HAMPATH to LONGEST-PATH by observing:

(G, s, t) \in HAMPATH  \\
\iff (G, s, t, |V| - 1) \in LONGEST-PATH

Therefore LONGEST-PATH is NP-hard.
**My Hobby:**
Embedding NP-Complete Problems in Restaurant Orders

<table>
<thead>
<tr>
<th>Chotchkies Restaurant</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Appetizers</strong></td>
<td></td>
</tr>
<tr>
<td>Mixed Fruit</td>
<td>2.15</td>
</tr>
<tr>
<td>French Fries</td>
<td>2.75</td>
</tr>
<tr>
<td>Side Salad</td>
<td>3.35</td>
</tr>
<tr>
<td>Hot Wings</td>
<td>3.55</td>
</tr>
<tr>
<td>Mozzarella Sticks</td>
<td>4.20</td>
</tr>
<tr>
<td>Sampler Plate</td>
<td>5.80</td>
</tr>
<tr>
<td><strong>Sandwiches</strong></td>
<td></td>
</tr>
<tr>
<td>Barbecue</td>
<td>6.55</td>
</tr>
</tbody>
</table>

We'd like exactly $15.05 worth of appetizers, please.

...Exactly? Uhh...

Here, these papers on the knapsack problem might help you out.

Listen, I have six other tables to get to—

As fast as possible, of course. Want something on Traveling Salesman?
coNP and Friends

(Note: any resemblance to other characters, living or animated, is purely coincidental)
NP: “Nifty Proofs”

For every $L$ in NP, if $x \in L$ then there is a “short proof” that $x \in L$:
$L = \{x \mid \exists y \text{ of poly}(|x|) \text{ length so that } V(x, y) \text{ accepts}\}$
But if $x \notin L$, there might not be a short proof!

There is an asymmetry between the strings in $L$ and strings not in $L$.

Compare with a recognizable language $L$:
Can always verify $x \in L$ in finite time (a TM accepts $x$), but if $x \notin L$, that could be because the TM goes in an infinite loop on $x$!
Definition: \( \text{coNP} = \{ L | \neg L \in \text{NP} \} \)

What does a coNP problem \( L \) look like?

The instances \( \text{NOT} \) in \( L \) have \textit{nifty proofs}. Recall we can write any NP problem \( L \) in the form: 
\[
L = \{ x | \exists y \text{ of poly}(|x|) \text{ length so that } V(x,y) \text{ accepts} \}
\]
Therefore:
\[
\neg L = \{ x | \neg \exists y \text{ of poly}(|x|) \text{ length so that } V(x,y) \text{ accepts} \}
= \{ x | \forall y \text{ of poly}(|x|) \text{ length, } V(x,y) \text{ rejects} \}
\]

Instead of using an \textit{“existentially guessing”} (nondeterministic) machine, we can define a \textit{“universally verifying”} machine!
Definition: \( \text{coNP} = \{ L \mid \neg L \in \text{NP} \} \)

What does a coNP computation look like?

A \textit{co-nondeterministic} machine has multiple computation paths, and has the following behavior:

- the machine \textbf{accepts} if \textit{all paths reach} accept state
- the machine \textbf{rejects} if \textit{at least one path} reaches reject state
Definition: $\text{coNP} = \{ L \mid \neg L \in \text{NP} \}$

What does a coNP computation look like?

In NP algorithms, we can use a “guess” instruction in pseudocode: 

Guess string $y$ of $k|x|^k$ length...

and the machine accepts if some $y$ leads to an accept state

In coNP algorithms, we can use a “try all” instruction: 

Try all strings $y$ of $k|x|^k$ length...

and the machine accepts if every $y$ leads to an accept state
TAUTOLOGY = \{ \phi \mid \phi \text{ is a Boolean formula and every variable assignment satisfies } \phi \} \\

Theorem: TAUTOLOGY is in coNP \\

How would we write pseudocode for a coNP machine that decides TAUTOLOGY? \\

How would we write TAUTOLOGY as the complement of some NP language?
Is $P \subseteq \text{coNP}$?

Yes!

$L \in P$ implies that $\neg L \in P$ (hence $\neg L \in \text{NP}$)

In general, *deterministic* complexity classes are closed under complement.
Is NP = coNP?

THIS IS AN OPEN QUESTION!

It is believed that NP ≠ coNP
Definition: A language $B$ is coNP-complete if

1. $B \in \text{coNP}$

2. For every $A$ in coNP, there is a polynomial-time reduction from $A$ to $B$ (B is coNP-hard)

Key Trick: Can use $A \leq_p B \iff \neg A \leq_p \neg B$ to turn NP-hardness into co-NP hardness
UNSAT = \{ \phi \mid \phi \text{ is a Boolean formula and no variable assignment satisfies } \phi \} 

Theorem: UNSAT is coNP-complete

Proof: (1) UNSAT ∈ coNP (why?)

(2) UNSAT is coNP-hard:

Let A ∈ coNP. We show A ≤_p UNSAT

Since \neg A ∈ NP, we have \neg A ≤_p 3SAT by the Cook-Levin theorem. This reduction already works!

\[
\begin{align*}
w &\in \neg A \Rightarrow \phi_w \in 3SAT \\
w &\notin \neg A \Rightarrow \phi_w \notin 3SAT
\end{align*}
\]

\[
\begin{align*}
w &\notin A \Rightarrow \phi_w \notin UNSAT \\
w &\in A \Rightarrow \phi_w \in UNSAT
\end{align*}
\]
TAUTOLOGY = \{ \phi \mid \phi \text{ is a Boolean formula and } \text{no variable assignment satisfies } \phi \} \}

Theorem: UNSAT is coNP-complete

TAUTOLOGY = \{ \phi \mid \phi \text{ is a Boolean formula and every variable assignment satisfies } \phi \} = \{\phi \mid \neg \phi \in \text{UNSAT}\}

Theorem: TAUTOLOGY is coNP-complete

(1) TAUTOLOGY \in \text{coNP} (already shown)

(2) TAUTOLOGY is coNP-hard:

UNSAT \leq_p TAUTOLOGY:
Given Boolean formula \( \phi \), output \( \neg \phi \)
Is $P = \text{NP } \cap \text{coNP}$?

**THIS IS AN OPEN QUESTION!**

\[ \text{NP } \cap \text{coNP} = \{ L \mid L \text{ and } \neg L \in \text{NP} \} \]

$L \in \text{NP } \cap \text{coNP}$ means that both $x \in L$ and $x \notin L$ have “nifty proofs”
Is $P = NP \cap \text{coNP}$?

Why might this be true?
- Analogy with computability

Why might this be false?
- If it’s true, most crypto fails!
An Interesting Problem in NP $\cap$ coNP

FACTORING

$= \{ (n, k) \mid n > k > 1 \text{ are integers written in binary, and there is a prime factor } p \text{ of } n \text{ where } k \leq p < n \}$

If FACTORING $\in$ P, we could potentially use the algorithm to factor every integer, and break RSA! Can binary search on $k$ to find a prime factor of $n$. More details in slides posted online.

**Theorem:** FACTORING $\in$ NP $\cap$ coNP
PRIMES = \{n \mid n \text{ is a prime number written in binary}\}

Theorem (Pratt ‘70s): PRIMES ∈ NP ∩ coNP

PRIMES is in P
Manindra Agrawal, Neeraj Kayal and Nitin Saxena

Abstract
We present an unconditional deterministic polynomial-time algorithm that determines whether an input number is prime or composite.
FACTORING
= \{ (n, k) \mid n > k > 1 \text{ are integers written in binary, there is a prime factor } p \text{ of } n \text{ where } k \leq p < n \} \\

**Theorem:** FACTORING $\in$ NP $\cap$ coNP

**Proof:**

(1) FACTORING $\in$ NP

A prime factor $p$ of $n$ such that $p \geq k$ is a proof that $(n, k)$ is in FACTORING

*(can check primality in P, can check $p$ divides $n$ in P)*

(2) FACTORING $\in$ coNP

The prime factorization $p_1^{e_1} \ldots p_m^{e_m}$ of $n$ is a proof that $(n, k)$ is not in FACTORING:

- **Verify** each $p_i$ is prime in P, and that $p_1^{e_1} \ldots p_m^{e_m} = n$
- **Verify** that for all $i=1,\ldots,m$ that $p_i < k$
Theorem: If FACTORING \( \in P \), then there is a polynomial-time algorithm which, given an integer \( n \), outputs either “\( n \) is PRIME” or a prime factor of \( n \).

Idea: Binary search for the prime factor!

Given binary integer \( n \), initialize an interval \([2,n]\).

If \((n, 2)\) is not in FACTORING then output “PRIME”

If \((n,\lceil n/2 \rceil)\) is in FACTORING then

- shrink interval to \([\lceil n/2 \rceil,n]\) (set \( k := \lceil 3n/4 \rceil \))
- else, shrink interval to \([2,\lceil n/2 \rceil]\) (set \( k := \lceil n/4 \rceil \))

Keep picking \( k \) to halve the interval after each \((n,k)\) call to FACTORING. Takes \( O(\log n) \) calls to FACTORING!