CS 341: ALGORITHMS

Lecture 21: intractability III – complexity class NP, poly transformations

Readings: see website

Trevor Brown

https://student.cs.uwaterloo.ca/~cs341
trevor.brown@uwaterloo.ca
THIS TIME

• Complexity class **NP**
  • Oracles, certificates, polytime verification algorithms
• Two problems in NP
  • Subset sum
  • Hamiltonian Cycle
• Relationship between P and NP
• Polynomial **transformations**
COMPLEXITY CLASS $NP$

$NP$: Non-deterministic polynomial time
EXAMPLE: SUBSET-SUM PROBLEM

• Suppose we are given some integers, -7, -3, -2, 5, 8

• Does some subset of these sum to zero?

  • In this case, yes: (-3) + (-2) + 5 = 0

Suppose I give you a certificate consisting of an array of numbers, and claim it represents such a subset.

If I’m telling the truth, then we call this a yes-certificate. It is is essentially a proof that “yes” is the correct output.

Of course, I might lie and give you a subset that does not sum to zero…

I could even give you numbers that are not in the input…

Can you use a yes-certificate to solve the problem efficiently?

Finding such a subset can be extremely difficult.

Can you determine whether I am lying in polynomial time?
Suppose there is a **non-deterministic oracle**, which returns a **subset that sums to 0 if one exists** and otherwise can return anything (even garbage).

We call the oracle’s output a **certificate**.

Given a certificate, can you verify in polytime whether it describes a solution to the problem?

```plaintext
SubsetSumWithOracle(I) = Oracle(I)
return verify(I, C)
```

Given such an oracle, this algorithm would **solve** subset-sum.

If there **exists** a subset that sums to 0, then C is one such subset, and we return true.

Otherwise, either C is not a subset of the input (return false), or C sums to a non-zero value (return false).

“Non-deterministic” is the N in NP, and it is so named because of oracles.

Here “**non-deterministic**” just means the oracle is magically guaranteed to return a yes-certificate if one exists.
Suppose there is a non-deterministic oracle, which returns a subset that sums to 0 if one exists and otherwise can return anything (even garbage).

We call the oracle’s output a certificate.

Given a certificate, can you verify in polytime whether it describes a solution to the problem?

Given a certificate from the oracle, would verify solve the problem in poly-time?

Test whether $C$ sums to 0
For loop with $|C||I|$ time...

Test whether $C$ is a subset of $I$
For loop with $|C|$ time...

Input to verify is $(I, C)$.
Runtime is $O(|C||I|)$, which is in $O(\text{Size}(I)^2) = O((|C| + |I|)^2)$

```
1 SubsetSumWithOracle(I)
2     C = Oracle(I)
3     return verify(I, C)
4
5 verify(I, C)
6     if C not subset of I then return false
7     return (sum(C) == 0)
```
DUMB SUBSET-SUM ALGORITHM: PRETEND YOU’RE AN ORACLE AND MAKE CERTS.

1. `SubsetSum(X[1..n])`
2. `for` every possible subset `S` of `X`
3. `if` `sumsToZero(S)` `then` return `true`
4. `return false`
Certificates

Certificate: Informally, a certificate for a yes-instance $I$ is some “extra information” $C$ which makes it easy to verify that $I$ is a yes-instance.

Certificate Verification Algorithm: Suppose that $Ver$ is an algorithm that verifies certificates for yes-instances. Then $Ver(I, C)$ outputs “yes” if $I$ is a yes-instance and $C$ is a valid certificate for $I$. If $Ver(I, C)$ outputs “no”, then either $I$ is a no-instance, or $I$ is a yes-instance and $C$ is an invalid certificate.

Polynomial-time Certificate Verification Algorithm: A certificate verification algorithm $Ver$ is a polynomial-time certificate verification algorithm if the complexity of $Ver$ is $O(n^k)$, where $k$ is a positive integer and $n = \text{Size}(I)$. 
Always keep the following in mind: finding a certificate can be much more difficult than verifying a given certificate.

As a rough analogy, finding a proof for a theorem can be much harder than verifying the correctness of someone else's proof.
GENERALIZING BEYOND SUBSET-SUM

• You can solve any decision problem in non-deterministic poly-time, given:
  1. a poly-time non-deterministic oracle, and
  2. a poly-time verify algorithm

• Such that:
  • If $I$ is a yes-instance, then the oracle returns a yes-certificate $C$ (i.e., a “proof” the answer is “yes”) and $\text{verify}(I, C)$ returns true
  • If $I$ is a no-instance, then $\text{verify}(I, C)$ returns false for all $C$ (i.e., it must be impossible to fool $\text{verify}$ into returning true)

• The algorithm:

```
1 SolveAnyProblemWithOracle(I)
2 C = Oracle(I)
3 return verify(I, C)
```

Our definition of NP will not explicitly involve non-deterministic oracles. But it is based on certificate verification, which makes more sense if you think of such oracles…

Could you “fool” the subset-sum verify function?
Oracle guesses solution in $O(1)$ time

Verifies solution in poly-time

As we are about to see: existence of a poly-time verifier for a problem means problem is in NP
DEFINING NP

• A decision problem $\Pi$ is solved by a poly-time $\text{verify}$ alg. iff:
  • for every $\text{yes}$-instance $I$, there exists a certificate $C$ such that $\text{verify}(I, C)$ returns true, and
  • for every $\text{no}$-instance $I$, $\text{verify}(I, C)$ returns false for every $C$

• The complexity class $\text{NP}$ denotes the set of all decision problems that can be solved by poly-time $\text{verify}$ algorithms

• No oracle needed! Note it is not necessary for an oracle to actually exist for a problem to be in $\text{NP}$. We can simply assume certificates come from an oracle, and show a poly-time $\text{verify}$ algorithm exists.
MECHANICS OF SHOWING A PROBLEM IS IN NP

• How to show $\Pi \in NP$
  1. Define a yes-certificate
  2. Design a poly-time $\text{verify}(I, C)$ algorithm
  3. Correctness proof
    • Case 1: Let $I$ be any yes-instance; Find $C$ such that $\text{verify}(I, C) = true$
    • Case 2: Let $I$ be any no-instance, and $C$ be any certificate; Prove $\text{verify}(I, C) = false$

   **Subset-sum as an example:**
   A yes-certificate is a list of indices in the input array where the elements should sum to 0

   How to verify a certificate $C$ is a subset of input $I$ with sum zero?
   \[
   \forall c \in C, \text{ add } I[c] \text{ to sum, and return true iff sum=0} \\
   \mathcal{O}(|C|) \text{ time}
   \]
   This is certainly polytime...

   Case 1: Let $I$ be a yes-instance.
   There is a subset in $I$ that sums to 0.
   For any such subset $C$, verify($I,C$) will return true.

   Case 2: Let $I$ be a no-instance & $C$ be any certificate.
   No subset of $I$ sums to 0.
   So, $\Sigma_{c \in C} I[c] \neq 0$ and verify returns false.

   So, subset-sum $\in NP$
ANOTHER EXAMPLE: HAMILTONIAN CYCLE PROBLEM

Problem 7.2

Hamiltonian Cycle

Instance: An undirected graph $G = (V, E)$.

Question: Does $G$ contain a hamiltonian cycle?

A hamiltonian cycle is a cycle that passes through every vertex in $V$ exactly once.

Let’s show that this problem is in NP! Have to find a poly-time verify algorithm…

Defining a yes-certificate: array of nodes representing a Hamiltonian cycle

How to verify that a given array of nodes represents a cycle?

How about a Hamiltonian cycle?
**EXAMPLE: SHOWING “HAMILTONIAN CYCLE” IS IN NP**

This is a *verify* algorithm that we imagine being called on the certificate $X$ produced by $oracle(G)$.

A *certificate* $X$ consists of an array of node names (1...n), which might represent a Hamiltonian cycle.

If $G$ is a **yes-instance** of the problem, then must show there **exists** some possible certificate $X$ for which this procedure returns will true.

Yes-instance implies there is a Hamiltonian cycle. Suppose $X$ is a sequence of $n$ consecutive nodes on that cycle. Then we return true!
EXAMPLE: SHOWING “HAMILTONIAN CYCLE” IS IN NP

If $G$ is a no-instance of the problem, then “every possible certificate should cause verify to return false”

Easier to prove the contrapositive: “if verify returns true, then $G$ is a yes-instance.”

This is a verify algorithm that we imagine being called on the certificate $X$ produced by oracle($G$)

A certificate $X$ consists of an array of node names (1…n), which might represent a Hamiltonian cycle

If we return true, then the graph contains a cycle with $n$ distinct nodes… So $G$ is a yes-instance

So, Hamiltonian Cycle is in NP
• $P \subseteq NP$
  
• Consider a problem $\Pi \in P$
  
• We show there exists a poly-time $verify(I, C)$ such that:
    • For every yes-instance $I$ of $\Pi$, $verify(I, C) = true$ for some $C$
    • For every no-instance $I$ of $\Pi$, $verify(I, C) = false$ for all $C$

• By definition, there is a poly-time algorithm $A$ to solve $\Pi$
  
  • Implement $verify(I, C)$ by simply running $A(I)$ [ignoring $C$]
  
  • Regardless of what $C$ is, $verify(I, C)$ satisfies the above

• How about $NP \subseteq P$? 
  
  Million dollar question. We think not.
POLYNOMIAL TRANSFORMATIONS

A subclass of poly-time reductions commonly used for \textit{NP-completeness} and \textit{impossibility} results
POLYNOMIAL TRANSFORMATIONS

For a decision problem \( \Pi \), let \( \mathcal{I}(\Pi) \) denote the set of all instances of \( \Pi \).
Let \( \mathcal{I}_{\text{yes}}(\Pi) \) and \( \mathcal{I}_{\text{no}}(\Pi) \) denote the set of all yes-instances and no-instances (respectively) of \( \Pi \).

Suppose that \( \Pi_1 \) and \( \Pi_2 \) are decision problems. We say that there is a polynomial transformation from \( \Pi_1 \) to \( \Pi_2 \) (denoted \( \Pi_1 \leq_P \Pi_2 \)) if there exists a function \( f : \mathcal{I}(\Pi_1) \rightarrow \mathcal{I}(\Pi_2) \) such that the following properties are satisfied:

- \( f(I) \) is computable in polynomial time (as a function of \( \text{size}(I) \)), where \( I \in \mathcal{I}(\Pi_1) \)
- if \( I \in \mathcal{I}_{\text{yes}}(\Pi_1) \), then \( f(I) \in \mathcal{I}_{\text{yes}}(\Pi_2) \)
- if \( I \in \mathcal{I}_{\text{no}}(\Pi_1) \), then \( f(I) \in \mathcal{I}_{\text{no}}(\Pi_2) \)

[Mechanics] to give a polynomial transformation, you must:
1. **specify** \( f(I) \),
2. **show** it runs in poly-time, and
3. **show** \( I \) is a yes-instance of \( \Pi_1 \) IFF \( f(I) \) is a yes-instance of \( \Pi_2 \).
A polynomial transformation can be thought of as a (simple) special case of a polynomial-time Turing reduction, i.e., if $\Pi_1 \leq_P \Pi_2$, then $\Pi_1 \leq_T^P \Pi_2$.

Given a polynomial transformation $f$ from $\Pi_1$ to $\Pi_2$, the corresponding Turing reduction is as follows:

Given $I \in \mathcal{I}(\Pi_1)$, construct $f(I) \in \mathcal{I}(\Pi_2)$.

Given an oracle for $\Pi_2$, say $A$, run $A(f(I))$.

We transform the instance, and then make a single call to the oracle.

Very important point: We do not know whether $I$ is a yes-instance or a no-instance of $\Pi_1$ when we transform it to an instance $f(I)$ of $\Pi_2$.

To prove the implication “if $I \in \mathcal{I}_{\text{no}}(\Pi_1)$, then $f(I) \in \mathcal{I}_{\text{no}}(\Pi_2)$”, we usually prove the contrapositive statement “if $f(I) \in \mathcal{I}_{\text{yes}}(\Pi_2)$, then $I \in \mathcal{I}_{\text{yes}}(\Pi_1)$.”

The contrapositive can help when it is hard to precisely characterize certificates for no-instances (or when such certificates don’t prove much).

Also known as Karp reductions and many-one reductions

We saw one instance where a contrapositive was easier to prove when we discussed Hamiltonian cycles.
SUMMARIZING
THE MORE CONVENIENT DEFINITION

- Let $\Pi_1$ and $\Pi_2$ be decision problems
- $\Pi_1 \leq_P \Pi_2$ iff there exists $f : I(\Pi_1) \to I(\Pi_2)$ such that:
  - $f(I)$ is computable in poly-time, for all $I \in I(\Pi_1)$
  - If $I \in I_{\text{yes}}(\Pi_1)$ then $f(I) \in I_{\text{yes}}(\Pi_2)$
  - If $f(I) \in I_{\text{yes}}(\Pi_2)$ then $I \in I_{\text{yes}}(\Pi_1)$

This is the contrapositive. Was previously (2 slides ago):
If $I \in I_{\text{no}}(\Pi_1)$ then $f(I) \in I_{\text{no}}(\Pi_2)$

Note: this is the same as saying $(I \in I_{\text{yes}}(\Pi_1)) \iff (f(I) \in I_{\text{yes}}(\Pi_2))$

This property justifies correctness for the following generic poly-time Karp reduction:

```
P1toP2KarpReduction(I)
    fI = f(I)
    return OracleForP2(fI)
```
### Problem 7.8
**Clique**
**Instance:** An undirected graph $G = (V, E)$ and an integer $k$, where $1 \leq k \leq |V|$.  
**Question:** Does $G$ contain a clique of size $\geq k$? (A **clique** is a subset of vertices $W \subseteq V$ such that $uv \in E$ for all $u, v \in W$, $u \neq v$.)

### Problem 7.9
**Vertex Cover**
**Instance:** An undirected graph $G = (V, E)$ and an integer $k$, where $1 \leq k \leq |V|$.  
**Question:** Does $G$ contain a vertex cover of size $\leq k$? (A **vertex cover** is a subset of vertices $W \subseteq V$ such that $\{u, v\} \cap W \neq \emptyset$ for all edges $uv \in E$.)
CLIQUE ≤ₚ VERTEX-COVER

• Suppose \( I = (G, k) \) is an instance of Clique where \( G = (V, E) \), \( V = \{v_1, ..., v_n\} \) and \( 1 \leq k \leq n \)

- **Want to solve** \( \text{Clique}(G, k) \)

• **Construct** instance \( f(I) = (\overline{G}, n - k) \) of Vertex-Cover, where \( H = (V, \overline{E}) \) and \( v_i v_j \in \overline{E} \iff v_i v_j \notin E \)

- **Claim:** there is a \( k \)-clique in \( G \) iff there is an \( (n - k) \) Vertex-Cover in \( \overline{G} \)

- **Consider the complement graph** \( \overline{G} \) of \( G \)

  - Every edge of \( G \) is a non-edge of \( \overline{G} \).
  - Every non-edge of \( G \) is an edge of \( \overline{G} \).

- **Given an adjacency matrix for** \( G \), get \( \overline{G} \) by flipping 0’s and 1’s.
We denote Clique by $CL$ and Vertex-Cover by $VC$.

$CL \leq_p VC$ iff there exists $f: I(CL) \rightarrow I(VC)$ such that:

1. $f(I)$ is computable in poly-time, for all $I \in I(CL)$
2. If $I \in I_{yes}(CL)$ then $f(I) \in I_{yes}(VC)$
3. If $f(I) \in I_{yes}(VC)$ then $I \in I_{yes}(CL)$

First let's show this.
**COMPLEXITY OF THE TRANSFORMATION**

**Suppose** \( I = (G, k) \) is an instance of Clique where \( G = (V, E) \), \( V = \{v_1, ..., v_n\} \) and \( 1 \leq k \leq n \)

**Construct** instance \( f(I) = (\overline{G}, n - k) \) of Vertex-Cover, where \( \overline{G} = (V, \overline{E}) \) and \( v_i v_j \in \overline{E} \iff v_i v_j \notin E \)

Want to solve \( Clique(G, k) \)

**Idea:** reduce to \( VertexCover(\overline{G}, n - k) \)

Assuming adjacency matrix, \( Size(I) = \Theta(n^2 + \log_2 k) \)

**Time to compute** \( f(I) \)?

Constructing \( \overline{G} \) takes \( O(n^2) \) time, and computing \( n - k \) takes \( O(\log n) \) time.

So computing \( f(I) \) takes \( O(n^2) \) time, which is polynomial in \( Size(I) \).
PROVING THIS IS A POLYNOMIAL TRANSFORMATION

- We denote Clique by $CL$ and Vertex-Cover by $VC$.

- $CL \leq_p VC$ iff there exists $f: I(CL) \rightarrow I(VC)$ such that:
  - $f(I)$ is computable in poly-time, for all $I \in I(CL)$
  - If $I \in I_{yes}(CL)$ then $f(I) \in I_{yes}(VC)$
  - If $f(I) \in I_{yes}(VC)$ then $I \in I_{yes}(CL)$

Now let's show this, i.e., if $G$ contains a $k$-clique then $\bar{G}$ contains an $(n - k)$ vertex cover.
PROVING: \( I \in I_{\text{yes}}(CL) \Rightarrow f(I) \in I_{\text{yes}}(VC) \)

- Suppose \( I = (G, k) \) is a \textbf{yes}-instance of Clique

- Then there is a set \( W \) of \( k \) vertices in a clique (with \textit{all-to-all} edges)

- Define \( \overline{W} = V \setminus W \). Clearly \(|\overline{W}| = n - k\).

- We \textbf{claim} \( \overline{W} \) is a vertex cover of \( G \)

- Consider any edge \((u, v) \in G\), then we are done, so assume \( u, v \notin \overline{W} \) to obtain a contradiction

- Then \( u, v \in W \), and \( W \) is a clique in \( G \), so \((u, v) \in G\)

- But \((u, v) \in \overline{G}\) implies \((u, v) \notin G\). Contradiction!

Example: \( \text{Clique}(G, 4) \)
PROVING THIS IS A POLYNOMIAL TRANSFORMATION

• We denote Clique by $CL$ and Vertex-Cover by $VC$
• $CL \leq_p VC \iff$ there exists $f : I(CL) \to I(VC)$ such that:
  • $f(I)$ is computable in poly-time, for all $I \in I(CL)$
  • If $I \in I_{yes}(CL)$ then $f(I) \in I_{yes}(VC)$
  • If $f(I) \in I_{yes}(VC)$ then $I \in I_{yes}(CL)$

Now let's show this, i.e., if $\tilde{G}$ contains an $(n - k)$ vertex cover, then $G$ contains a $k$-clique
PROVING: $f(I) \in J_{\text{yes}}(VC) \Rightarrow I \in J_{\text{yes}}(CL)$

- Suppose $f(I) = (\overline{G}, n - k)$ is a yes-instance of $VC$
- Then there is a set of $n - k$ vertices $\overline{W}$ that is a vertex cover of $\overline{G}$
- Define $W = V \setminus \overline{W}$. Clearly $|W| = k$.
- We claim $W$ is a clique in $G$

Since $\overline{W}$ is a vertex cover of $\overline{G}$, every edge in $\overline{G}$ has at least one endpoint in $\overline{W}$

Therefore, no edge in $\overline{G}$ has two endpoints in $W$

So, in $G$, there are edges between all pairs of nodes in $W$. So, $W$ is a clique in $G$.

So, we have demonstrated a polynomial transformation from CLIQUE to VERTEX-COVER