Lecture 22: intractability V – More NPC transformations

Readings: see website

Trevor Brown

https://student.cs.uwaterloo.ca/~cs341

trevor.brown@uwaterloo.ca
LAST TIME

• Polynomial transformations
  • Poly transformation from Clique to Vertex Cover
• NP Completeness
  • SAT is NP complete (NPC)
  • Got part way through showing 3SAT is NPC
    • Did poly transformation from SAT to 3SAT
    • Need to also show 3SAT is in NP
LET’S DO A BRIEF REVIEW

of NPC, poly transformations, and showing a problem is in NP
The complexity class **NPC** denotes the set of all decision problems $\Pi$ that satisfy the following two properties:

1. $\Pi \in \text{NP}$
2. For all $\Pi' \in \text{NP}$, $\Pi' \leq_P \Pi$.

**NPC** is an abbreviation for **NP-complete**.

Note that the definition does not imply that NP-complete problems exist!

**Mechanics of proving** $\Pi \in \text{NPC}$

1. Show $\Pi$ is in NP
2. Show a poly transformation from some NPC problem to $\Pi$
MECHANICS OF SHOWING A PROBLEM IS IN NP

• How to show $\Pi \in NP$

1. Define a yes-certificate
2. Design a poly-time $verify(I, C)$ algorithm
3. Correctness proof
   • **Case 1**: Let $I$ be any yes-instance;
     Find $C$ such that $verify(I, C) = true$
   • **Case 2**: Let $I$ be any no-instance, and $C$ be any certificate;
     Prove $verify(I, C) = false$
Let $\Pi_1$ and $\Pi_2$ be decision problems

$\Pi_1 \leq_P \Pi_2$ iff there exists $f : I(\Pi_1) \rightarrow I(\Pi_2)$ such that:

- $f(I)$ is computable in poly-time, for all $I \in I(\Pi_1)$
- If $I \in I_{yes}(\Pi_1)$ then $f(I) \in I_{yes}(\Pi_2)$
- If $f(I) \in I_{yes}(\Pi_2)$ then $I \in I_{yes}(\Pi_1)$

**POLYNOMIAL TRANSFORMATION FOR PROVING $\Pi_2$ IS IN NPC**

Known NPC problem

Problem you want show is NPC

6
LET’S FINISH SHOWING $3\text{SAT} \in \text{NPC}$

- Already poly transformed SAT to 3SAT
- Need to show 3SAT in NP
PROVING 3SAT IS IN NP

1. Define desired YES-certificate
2. Design a poly-time $verify(I, C)$ algorithm
3. Correctness proof
   • **Case 1**: Let $I$ be any yes-instance; Find $C$ such that $verify(I, C) = true$
   • **Case 2**: Let $I$ be any no-instance, and $C$ be any certificate; Prove $verify(I, C) = false$
   • **Contrapositive of case 2**: Suppose $verify(I, C) = true$; Prove $I$ is a yes-instance

3SAT input $I = (Clauses[1..m], n)$:
a list of $m$ clauses, and the number $n$ of variables. Each clause contains literals. Each literal is a pair (var, neg): a variable $\in \{1..n\}$ & a negation bit

YES-certificate $C = array$ with one bit per variable in $\{1..n\}$ representing a satisfying assignment

```
1 verify3SAT(I=(Clauses[1..m], n), C)
2 if C is not an array of n bits return false
3 numSat = 0
4 for each c in Clauses
5   for each literal (var, neg) in c
6     if (C[var] && !neg) or (!C[var] && neg)
7       numSat++
8     break
9 return (numSat == m)
```

This takes $O(|Clauses|)$ time, which is polynomial in $\text{Size}(I)$
MECHANICS OF SHOWING A PROBLEM IS IN NP

1. Define desired YES-certificate

2. Design a poly-time \( verify(I, C) \) algorithm

3. Correctness proof
   - **Case 1:** Let \( I \) be any yes-instance; Find \( C \) such that \( verify(I, C) = true \)
   - **Case 2:** Let \( I \) be any no-instance, and \( C \) be any certificate; Prove \( verify(I, C) = false \)
   - **Contrapositive of case 2:** Suppose \( verify(I, C) = true \); Prove \( I \) is a yes-instance

Let \( I \) be a yes-instance of 3SAT. Then it has a satisfying assignment \( A_s \). And, \( verify(I, A_s) \) will see that each clause contains a literal satisfied by this assignment, so \( verify \) will see \( numSat = |Clauses| \) and return true.

Suppose \( verify(I, C) \) returns true. Then \( numSat = |Clauses| \), so \( numSat \) was incremented in each iteration of the loop over clauses, so each clause contains a satisfied literal, so the 3SAT formula in \( I \) is satisfied by \( C \), so \( I \) is a yes-instance.

It follows that 3SAT is in NP.

Since we have already shown SAT \( \leq_p \) 3SAT, we now know that 3SAT is NP-COMPLETE.
Every problem in NP can be poly transformed to

Since SAT is NP-complete, so is 3-SAT!

Today and next time let's start filling out a hierarchy of reductions that prove several problems are NP complete

But first, since you need to know NP hardness for your assignment…
NP-HARDNESS

Intuitively: problems that are \textit{at least as hard} as NP-complete (but are not necessarily decision problems)
NP-hard Problems

A problem $\Pi$ is **NP-hard** if there exists a problem $\Pi' \in NPC$ such that $\Pi' \leq_T^P \Pi$.

Every NP-complete problem is automatically NP-hard, but there exist NP-hard problems that are not NP-complete.

Typical examples of NP-hard problems are optimization problems corresponding to NP-complete decision problems.

For example, **TSP-Decision** $\leq_T^P$ **TSP-Optimization** and **TSP-Decision** $\in$ NPC, so **TSP-Optimization** is NP-hard.

**TSP-Optimal Value** is also NP-hard (and not in NP)

This version returns the total weight of an optimal Hamiltonian cycle

Reduction from lecture 19/20

Returns an optimal Hamiltonian cycle
COMPARING NPC AND NP HARD

• $\Pi \in \text{NPC}$
  • Must be a decision problem
  • Must poly transform some NPC problem to $\Pi$
  • Must show $\Pi$ in NP

• $\Pi \in \text{NPHard}$
  • Does not need to be a decision problem
  • Can use either poly transform or poly Turing reduction
  • Does not need to be in NP (and can't be if not decision)
TWO POSSIBLE REALITIES...

P ≠ NP

P = NP

P = NP = NP-Complete

NP

NP-Complete

NP-Hard
SOME PROBLEMS IN EACH

We think some stuff may exist here, called NP intermediate problems... but we're not sure.

TSP optimization
SAT, TSP-Decision
BFS

P \neq \text{NP}

P = \text{NP} = \text{NP-Complete}

\text{NP-Hard}

\text{NP-Complete}

\text{NP}

\text{P}
ESTABLISHING ANOTHER NPC PROBLEM

... BY TRANSFORMING 3-SAT TO CLIQUE

(Proving 3-SAT $\leq_P$ Clique)
SHOWING \( 3\text{-SAT} \leq_p \text{ CLIQUE} \)

- Let \( I \) be an instance of 3-SAT with \( n \) variables \( x_1 \ldots x_n \) and \( m \) clauses \( C_1 \ldots C_m \)
  - E.g., \( (x_1 \lor x_2 \lor \overline{x}_3) \land (x_1 \lor x_2 \lor x_3) \land (\overline{x}_2 \lor x_3 \lor x_5) \land (\overline{x}_3 \lor x_4 \lor \overline{x}_5) \) \[ n = 5, m = 4 \]
  - We construct \textbf{Clique} input \( f(I) = (G, k) \):
    - Node \( v_{\ell}^c \) for each literal \( 1 \leq \ell \leq 3 \) in each clause \( 1 \leq c \leq m \) (so \( |V| = 3m \))
    - Edges between all \textbf{non-contradictory} pairs of nodes (no \( x_i \land \overline{x}_i \)) in \textbf{different clauses}
    - \( k = m \) (can we find an \( m \)-clique?)
    - Must prove this is a \textbf{polynomial transformation}

Reasonable 3-SAT representation: array[1..m] of clauses \( <l_1, l_2, l_3> \) of literals \( <v, neg> \) where \( v \in \{1..n\} \).

Runtime: create \( 3m \) nodes, \( O(m^2) \) edges, at \( O(1) \) time each

Note \( O(m) \subseteq O(\text{Size}(I)) \),
So runtime \( O(m^2) \subseteq O(\text{Size}(I)^2) \) \( \Rightarrow \) \textbf{polytime!}
SHOWING $3$-SAT $\leq_p$ CLIQUE

Let $I$ be an instance of $3$-SAT with $n$ variables $x_1 \ldots x_n$ and $m$ clauses $C_1 \ldots C_m$

- E.g., $(x_1 \lor \overline{x_2} \lor \overline{x_3}) \land (\overline{x_1} \lor x_2 \lor x_3) \land (\overline{x_2} \lor x_3 \lor x_5) \land (x_3 \lor x_4 \lor x_5)$

**Case 1:** Suppose $I$ is a yes-instance of $3$-SAT, and show $f(I)$ is a yes-instance of $m$-clique

- Since $I$ is a yes-instance, $\exists$ a satisfying assignment
  - E.g., $x_1 = 1$, $x_2 = 1$, $x_3 = 1$, $x_4 = 0$, $x_5 = 0$

- For each clause $C_i$, let $s_i$ be a satisfied literal in $C_i$
  - E.g., $s_1 = x_1$, $s_2 = x_2$, $s_3 = x_3$, $s_4 = \overline{x_5}$

**Claim:** the corresponding nodes form an $m$-clique

- There are $m$ of these nodes, each in a different clause
- None of them represent contradictory truth assignments
- So, there are edges between all pairs of them $\Rightarrow$ they form an $m$-clique
SHOWING 3-SAT \leq_p CLIQUE

• Let \( I \) be an instance of 3-SAT with \( n \) variables \( x_1 \ldots x_n \) and \( m \) clauses \( C_1 \ldots C_m \)
  
  • E.g., \((x_1 \lor x_2 \lor \overline{x_3}) \land (\overline{x_1} \lor x_2 \lor x_3) \land (x_2 \lor x_3 \lor x_5) \land (\overline{x_3} \lor x_4 \lor \overline{x_5})\)

• **Case 2:** Suppose \( f(I) \) is a yes-instance of \( m \)-clique, and show \( I \) is a yes-instance of 3-SAT

• Since \( f(I) \) is a yes-instance, it contains an \( m \)-clique

• Clique contains edges between all pairs of nodes

• There are no edges between nodes in same clause, so clique contains one node from each clause

• Set the corresponding literals to be satisfied

• Clique contains no edges between contradictory literals (i.e., no edge connects \( x_i \) and \( \overline{x_i} \) for any \( i \))

• So, truth assignment is consistent and satisfies each clause (and the formula)
LAST STEP: SHOW CLIQUE IS IN NP

- **YES-certificate**: array of k nodes forming a clique
- **Verify(I,C)**:
  - Check certificate is array of length k, containing vertex IDs
  - Check all-to-all edges to verify these vertices form a clique
  - \( O(k^2) \subseteq O(|V|^2) \) runtime \( \rightarrow \) polytime
- **Correctness: exercise!** Need to prove:
  - if I is a yes instance, verify returns yes, and
  - if verify returns yes then I is a yes instance
Every problem in NP can be poly transformed to 3-SAT.
Every problem in NP can be poly transformed to

This additional poly transformation was proved last class (CL to VC)!

We also need to show Vertex Cover is in NP. Exercise. 😊
REDUCING VERTEX-COVER TO SUBSET-SUM

(Proving Vertex-Cover $\leq_p$ Subset-Sum)

(if we have time)
Earlier, we defined Subset-Sum with a target sum of 0.

Here we add a target sum $T$ and take positive integers as input.

**Goal:** transform instance $I$ of VC into instance $f(I)$ of SS (in poly time) such that $I$ is a yes-instance of VC iff $f(I)$ is a yes-instance of SS.

**Idea:** turn nodes and edges into a list of integers and a target sum $W$. Sum $W$ should be achievable IFF there is a $k$-vertex cover.

Somehow want the array of integers to encode which edges are covered by various nodes, and target sum to encode that every edge is covered if $W$ is achieved.
Vertex Cover $\leq_P$ Subset Sum

Suppose $I = (G, k)$, where $G = (V, E)$, $|V| = n$, $|E| = m$ and $1 \leq k \leq n$. Suppose $V = \{v_1, \ldots, v_n\}$ and $E = \{e_0, \ldots, e_{m-1}\}$. For $1 \leq i \leq n$, $0 \leq j \leq m-1$, let $C = (c_{ij})$, where

$$c_{ij} = \begin{cases} 1 & \text{if } e_j \text{ is incident with } v_i \\ 0 & \text{otherwise.} \end{cases}$$

$c_{ij}$ = is edge $j$ covered by node $i$?

Input to Vertex Cover

Sort of like an adjacency matrix, but instead of storing which node-pairs are adjacent, store which edges are incident to each node.
Each edge becomes a unique number in the array: edge \( e_j \) becomes \( 10^j \)

\[
\begin{align*}
b_0 &= 1 \\
b_1 &= 10 \\
b_2 &= 100 \\
b_3 &= 1000 \\
b_4 &= 10000
\end{align*}
\]
Each edge becomes a unique integer in the array:
\[ e_j \] becomes \[ 10^j \] for all edges incident to the node.

Each node becomes a number in the array:
\[ 10^m + \text{the integers for all edges incident to the node} \]

Define \( n + m \) ints and a target sum \( W \) as follows:
\[
\begin{align*}
a_i &= 10^m + \sum_{j=0}^{m-1} c_{ij}10^j \quad (1 \leq i \leq n) \\
b_j &= 10^j \quad (0 \leq j \leq m-1)
\end{align*}
\]

E.g.,
\[
\begin{align*}
a_1 &= 100011 \\
a_2 &= 110110 \\
b_0 &= 1 \\
b_1 &= 10 \\
b_2 &= 100 \\
b_3 &= 1000 \\
b_4 &= 10000
\end{align*}
\]
Vertex Cover $\leq_P$ Subset Sum

Suppose $I = (G, k)$, where $G = (V, E)$, $|V| = n$, $|E| = m$ and $1 \leq k \leq n$.
Suppose $V = \{v_1, \ldots, v_n\}$ and $E = \{e_0, \ldots, e_{m-1}\}$. For $1 \leq i \leq n$, $0 \leq j \leq m - 1$, let $C = (c_{ij})$, where

$$c_{ij} = \begin{cases} 
1 & \text{if } e_j \text{ is incident with } v_i \\
0 & \text{otherwise.}
\end{cases}$$

Define points and a target sum $W$ as follows:

$$a_i = 10^m + \sum_{j=0}^{m-1} c_{ij}10^j \quad (1 \leq i \leq n)$$

$$b_j = 10^j \quad (0 \leq j \leq m - 1)$$

$$W = k \cdot 10^m + \sum_{j=0}^{m-1} 2 \cdot 10^j$$

Then define $f(I) = (a_1, \ldots, a_n, b_0, \ldots, b_{m-1}, W)$.

Why twice? If both endpoints of $e_j$ are in the vertex cover, it is counted twice. Otherwise once, and can add $b_j$.

This target weight asks for $k$ nodes and for all edges to be included twice.

Each node becomes a number in the array: $10^m +$ the integers for all edges incident to the node.

Each edge becomes a unique integer in the array: edge $e_j$ becomes $10^j$. 
Is there a 2-VC? Use subset sum to search for $W = 222222$

Subset sum looks for a subset of \{a_1, a_2, a_3, a_4, a_5, b_0, b_1, b_2, b_3, b_4\} that sums to $W$

It finds $W = a_2 + a_3 + b_0 + b_1 + b_3 + b_4$

$W = 222222 = a_2 + a_3 + b_0 + b_1 + b_3 + b_4$

Edge $e_2$ counted twice, other edges once. Sum uses $b_0, b_1, b_3, b_4$ to get all to be counted twice.

Note: no “carrying” can occur even if we sum everything

Most significant digit(s) of $W$ accurately capture # of nodes

Other digits are in $[0,3]$. An edge is definitely covered by a node if its digit is 2.
Correctness of the Transformation

**Case 1:** Suppose $I$ is a **yes**-instance of Vertex-Cover. There is a vertex cover $V' \subseteq V$ such that $|V'| = k$. For $i = 1, 2$, let $E^i$ denote the edges having exactly $i$ vertices in $V'$. Then $E = E^1 \cup E^2$ because $V'$ is a vertex cover.

Let $A' = \{a_i : v_i \in V'\}$ and $B' = \{b_j : e_j \in E^1\}$.

The sum of the nodes in $A'$ is $k \cdot 10^m + \sum_{\{j : e_j \in E^1\}} 10^j + \sum_{\{j : e_j \in E^2\}} 2 \times 10^j$.

The sum of the edges in $B'$ is $\sum_{\{j : e_j \in E^1\}} 10^j$.

Therefore the sum of all the chosen ints is $k \cdot 10^m + \sum_{\{j : e_j \in E\}} 2 \cdot 10^j = k \cdot 10^m + \sum_{j=1}^{m} 2 \cdot 10^j = W$.
We show $I$ is a **yes**-instance of Vertex-Cover

Since $f(I)$ is a yes-instance, there exists $A' \cup B'$ that sums to $W$
- where $A'$ contains node ints and $B'$ contains edge ints

Define $V' = \{v_i : a_i \in A'\}$. We claim $V'$ is a vertex cover of size $k$.
- We must have $|V'| = k$ for the coefficient of $10^m$ to be $k$ (no carrying)
- Suppose (for contra.) $V'$ does **not** cover some edge $e_j = (u, v)$
  - Then the coefficient of $10^j$ is **zero** for every $a_i \in A'$
  - But the coefficient of $10^j$ is 2, so a subset of $B'$ must sum to $2 \times 10^j$
  - But this is impossible (so $e_j$ is covered, so all edges are covered)
**Vertex Cover \( \leq_P \) Subset Sum**

Suppose \( I = (G, k) \), where \( G = (V, E) \), \( |V| = n \), \( |E| = m \) and \( 1 \leq k \leq n \).

Suppose \( V = \{v_1, ..., v_n\} \) and \( E = \{e_0, ..., e_{m-1}\} \). For \( 1 \leq i \leq n \), \( 0 \leq j \leq m - 1 \), let \( C = (c_{ij}) \), where

\[
c_{ij} = \begin{cases} 1 & \text{if } e_j \text{ is incident with } v_i \\ 0 & \text{otherwise.} \end{cases}
\]

Define \( n + m \) sizes and a target sum \( W \) as follows:

\[
a_i = 10^m + \sum_{j=0}^{m-1} c_{ij}10^j \quad (1 \leq i \leq n)
\]

\[
b_j = 10^j \quad (0 \leq j \leq m - 1)
\]

\[
W = k \cdot 10^m + \sum_{j=0}^{m-1} 2 \cdot 10^j
\]

Then define \( f(I) = (a_1, ..., a_n, b_0, ..., b_{m-1}, W) \).

---

**Complexity of the transformation:** Easy! Included for your notes.

Assume adjacency matrix and unit cost model for simplicity.

Compute \( C \) with trivial algorithm in \( O(nm) \) time.

Compute \( a_i \) by visiting all incident edges. Trivial algorithm yields \( O(m) \) time for each \( a_i \), totaling \( O(nm) \) over all \( i \).

Trivial to compute all \( b_j \) in \( O(m) \) time.

Trivial to compute \( W \) in \( O(m) \) time.

Total \( O(nm) \) time. This is polynomial in the input graph size!
Every problem in NP can be poly transformed to

Technically need to also show SubsetSum with target T is in NP (exercise) to know it is in NPC
IS 2-SAT ALSO HARD?
(IF WE HAVE TIME – VERY UNLIKELY)
2-SAT EXAMPLES

• \((p \lor q) \land (\neg p \lor r) \land (\neg r \lor \neg p)\)

  • Satisfiable: \(p = 0, q = 1, r \in \{0,1\}\)
  
• \((p \lor q) \land (\neg p \lor r) \land (\neg r \lor \neg p) \land (p \lor \neg q)\)

Logical refresher:

\(p \Rightarrow q\) is equivalent to \(\neg p \lor q\).

Therefore, \(p \lor q\) is equivalent to \(\neg p \Rightarrow q\) and equivalent to \(\neg q \Rightarrow p\)

Edges (implications of clauses)...

\[
\begin{align*}
\neg p &\Rightarrow q & p &\Rightarrow r & r &\Rightarrow \neg p & \neg p &\Rightarrow \neg q \\
\neg q &\Rightarrow p & \neg r &\Rightarrow \neg p & p &\Rightarrow \neg r & q &\Rightarrow p \\
\end{align*}
\]

\(q \Rightarrow p \Rightarrow \neg r \Rightarrow \neg p \Rightarrow \neg q\) ... so \(q\) cannot be true

\(\neg q \Rightarrow p \Rightarrow \neg r \Rightarrow \neg p \Rightarrow q\) ... so \(q\) cannot be false

Therefore the formula cannot be satisfied!
2-SAT can be solved in polynomial time. Suppose we are given an instance $I$ of 2-SAT on a set of boolean variables $X = \{1..|X|\}$.

1. For every clause $x \lor y$ (where $x$ and $y$ are literals), construct two directed edges $\overline{x}y$ and $\overline{y}x$. We get a directed graph on vertex set $X \cup \overline{X}$.

2. Determine the strongly connected components of this directed graph.

3. $I$ is a yes-instance if and only if there is no strongly connected component containing $x$ and $\overline{x}$, for any $x \in X$.

Suppose no variable $x$ is in the same SCC as $\overline{x}$, then to get a satisfying assignment do the following:

For each $x$, if there exists a path from $x$ to $\overline{x}$, then set $x = false$ else set $x = true$. 

(variable names are integers in 1..|X|)
BONUS SLIDES
SUMMARY OF COMPLEXITY CLASSES

- **P** (Poly-time)
  - E.g., (decision problem variants of:) BFS, Dijkstra’s, some DP algorithms
  - **Decision** problems that can be solved by algorithms with runtime $\text{poly}(\text{input size})$

- **NP** (Non-deterministic poly-time)
  - All of P, and e.g., vertex cover, clique, SAT, subset sum
  - **Decision** problems for which certificates can be verified in time $\text{poly}(\text{input size})$
  - Equivalently: decision problems that can be solved in poly-time if you have access to a non-deterministic oracle that returns a yes-certificate if one exists

- **NPC** (NP-complete)
  - E.g., vertex cover, clique, SAT, subset sum, TSP-decision
  - **Decision** problems $\Pi \in NP$ s.t. every $\Pi' \in NP$ can be transformed to $\Pi$ in poly-time
  - **NP-hard** (at least as hard as NPC)
    - problems $\Pi$ s.t. every $\Pi' \in NP$ can be reduced to $\Pi$ in poly-time

- Note: P, NP and NPC problems are **decidable**
Found this neat image online