CS 341: ALGORITHMS
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

COMPLEXITY CLASS NP-COMPLETE (NPC)
The complexity class NPC denotes the set of all decision problems \( \Pi \) that satisfy the following two properties:
\( \Pi \in \text{NP} \)
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 SHOWING A PROBLEM IS IN NP
• How to show \( \Pi \in \text{NP} \)
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 \( \text{verify}(I, C) = \text{true} \)
• Case 2: Let \( I \) be any no-instance,
  and \( C \) be any certificate;
  Prove \( \text{verify}(I, C) = \text{false} \)

POLYNOMIAL TRANSFORMATION FOR PROVING \( \Pi_2 \) IS IN NPC
• Let \( \Pi_1 \) and \( \Pi_2 \) be decision problems
• \( \Pi_1 \leq_P \Pi_2 \) iff there exists \( f : \text{sat}(I_1) \to \text{sat}(I_2) \) such that:
  • \( f(I) \) is computable in poly-time, for all \( I \in \text{sat}(I_1) \)
  • If \( I \in \text{sat}(I_1) \) then \( f(I) \in \text{sat}(I_2) \)
  • If \( f(I) \in \text{sat}(I_2) \) then \( I \in \text{sat}(I_1) \)
LET'S FINISH SHOWING 3SAT ∈ 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

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

NP-HARDNESS

Affinately: problems that are at least as hard as NP-complete (but are not necessarily decision problems)
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
- \( \Pi \in \text{NP}\text{Hard} \)
  - 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...

Some Problems in Each

- NP-Complete
- NP-Hard
- Polynomial

Establishing Another NPC Problem

... by transforming 3-SAT to CLIQUE

Showing 3-SAT ≤ \( \text{P} \) 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., \((x_1 \lor x_2 \lor \neg x_3) \land (\neg x_1 \lor x_3 \lor x_4) \land (x_2 \lor \neg x_4 \lor x_5)\) \( \land n = 5, m = 4 \)
- We construct CLIQUE input \( f(I) = (x, k) \):
  - Node \( x \) for each literal \( 1 \leq c \leq 3 \) in each clause \( 1 \leq c \leq m \) \( |V| = 3m \)
  - Edges between all non-contradictory pairs of nodes \( \{x_1 \neq x_2 \} \) in different clauses
  - \( k = m \) (can we find an \( m \)-clique?)
- Must prove that \( f \) is a polynomial transformation

Reasonable 3-SAT representation: array \( \{1, \ldots k\} \) of clauses \( C_{1} \ldots C_{k} \) of literals \( x_{i} \lor \neg x_{i} \lor \neg x_{i} \) where \( x \in \{1 \ldots n\} \)

Node \( \theta(x) = \{x, \neg x, \neg x\} \).
- So runtime \( \Theta(nm^2) \) is \( \Theta(\text{polytime}) \).

SHOWING 3-SAT ≤ \( \text{P} \) CLIQUE

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, let \( \exists \) be a satisfied literal in \( C_i \)
  - E.g., \( s_1 = x_1, s_2 = x_2, s_3 = x_3, s_4 = x_4 \)

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 ≤ₚ CLIQUE

• Let I be an instance of 3-SAT with n variables x₁...xₙ and m clauses C₁...Cₘ.
• E.g., (x₁ ∨ x₂ ∨ x₃) ∧ (x₁ ∨ x₂ ∨ x₄) ∧ (x₁ ∨ x₂ ∨ x₃) ∧ (x₁ ∨ x₂ ∨ x₄)
• 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 xᵢ and ̅xᵢ 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(k²) ∈ O(|V|²) runtime → 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

This additional poly transformation was proved last class (CLIQUE to VC)! We also need to show Vertex Cover is in NP. Exercise. ☺

REDUCING VERTEX-COVER TO SUBSET-SUM

(Proving VertexCover ≤ₚ Subset-Sum)

(If we have time)

SUBSET-SUM (SLIGHTLY DIFFERENT FROM BEFORE)

Problem 7.18

Subset Sum
Instance: A list of sizes S = {s₁,...,sₙ}; and a target sum W. These are all positive integers.
Question: Does there exist a subset J ⊆ {1,...,n} such that \(\sum_{i \in J} s_i = W\)?

• Earlier, we defined SubsetSum with a target sum of 0.
• Here we add a target sum T and take positive integers as input.

God: transform instance J of VC into instance f(J) of SS (in poly time) such that J is a yes-instance of VC if and only if f(J) 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 if 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.
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.

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

Each edge becomes a unique number in the array: edge \( e_j \) becomes \( 10^j \) ints.

E.g.,

Each node becomes a number in the array:

100 + the integers for all edges incident to the node

This target weight asks for \( k \) nodes and for all edges to be included twice.

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

EXAMPLE

Looking for 2 nodes

Correctness of the Transformation

Case 1: Suppose \( I \) is a yes instance of \( \text{Vertex Cover} \).

There is a vertex cover \( V' \) such that \( |V'| = \frac{k}{2} \). For an edge \( e_j \) let \( W_c \) denote the edges having exactly 1 vertex in \( V' \).

Then \( W = E^{1^t} \cup E^{2^t} \) because \( V' \) is a vertex cover.

Let \( W_c = \{ e_j \mid e_j \text{ has one endpoint in } V' \} \).

Both endpoints in \( V' \). Case 2: Suppose \( I \) is a no instance of \( \text{Vertex Cover} \).

There is a vertex cover \( V' \) such that \( |V'| \geq \frac{k}{2} + 1 \).

Then \( W = E^{1^t} \cup E^{2^t} \) because \( V' \) is a vertex cover.

Let \( W_c = \{ e_j \mid e_j \text{ has both endpoints in } V' \} \).

To get \( 2 \times 10^j \) for all \( e_j \) plus \( 10^m \) for each node.
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)

Case 2: Suppose \( f(I) \) is a yes-instance of Subset Sum.

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

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

Compute \( a_i \) by visiting all incident edges. Trivial algorithm yields \( O(m) \) time for each \( a_i \)

Total \( O(nm) \) time. This is polynomial in the input graph size!
SUMMARY OF COMPLEXITY CLASSES

- **P** (Polynomial)
  - Examples: decision problem variants of Dijkstra's, BFS, some DP algorithms
  - Decision problems that can be solved by algorithms with runtime poly(input size)

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

- **NPC** (NP-complete)
  - Decision problems \( \Pi \in NP \) s.t. every \( \Pi' \in NP \) can be transformed to \( \Pi \) in polytime

- **NP-hard** (At least as hard as NPC)
  - All of NPC and e.g., TSP-optimization, TSP-optimal value
  - Problems \( \Pi \) s.t. every \( \Pi' \in NP \) can be reduced to \( \Pi \) in polytime

- **Note:** P, NP and NPC problems are decidable.