EECS 215
Lecture 13

Brian Demsky
bdemsky@uci.edu

The Henry Samueli School of Engineering
Electrical Engineering and Computer Science
University of California, Irvine
Lecture 13: Overview

- NP-Completeness (Chapter 34)
Hamiltonian Cycles

- Given graph $G=(V,E)$
- Find simple cycle in graph that contains each vertex $V$
- Hamiltonian-cycle problem:
  Does graph $G$ have a hamiltonian cycle?
- HAM-CYCLE = \{ <G> : G is a hamiltonian graph \}
How to Decide?

- One solution: check each possible permutation of the vertices to see if it is a hamiltonian path
- Encoding: For adjacency matrix, number of vertices=$\Omega(\sqrt{n})$ where $n=|<G>|$
- $m!$ possible permutations
- Runtime is $\Omega(m!)=\Omega(\sqrt{n}!)=\Omega(2^{\sqrt{n}})$
Verification

- If we are given a sequence of vertices, we can verify whether it is a hamiltonian cycle easily in polynomial time.

- Idea: Algorithm takes in
  - Input string $x$
  - Certification of solution $y$
Complexity Class NP

- Class of languages that can be verified by a polynomial-time algorithm
- \(L = \{x \in \{0, 1\}^* : \text{there exists a certificate } y \text{ with } |y| = O(|x|^c) \text{ such that } A(x, y) = 1\}\)
- We say that \(A\) verifies language \(L\) in polynomial time
Reducibility

- How do we prove a problem NP-complete?
- Intuition: problem Q can be reduced to Q’ if any instance of Q is “easily rephrased” into an instance of Q’
- We say that L₁ is polynomial-time reducible to L₂ if there exists a polynomial time computable function f: \{0,1\}^* → \{0,1\}^* such that for all x in \{0,1\}^*
  x ∈ L₁ if and only if f(x) ∈ L₂
Call f the reduction function and a polynomial-time algorithm F that computes f is called a reduction algorithm
Proving P

• Lemma:
If $L_1, L_2 \subseteq \{0,1\}^*$ are languages such that $L_1$ is polynomial time reducible to $L_2$ then $L_2 \in P$ implies $L_1 \in P$. 
Proving NP-complete

If \( L \in \{0,1\}^* \) is NP-complete if

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

2. \( L' \) is polynomial time reducible to \( L \) for every \( L' \in \text{NP} \) (NP-hard)
Can reduce all of NP to Circuit Satisfiability

• Idea:
  • Represent computation of A as sequence of configurations
  • Each configuration includes PC, machine state, input, certificate, working storage
  • Use combinatorial circuit M that implements computer hardware
Reduction

• Reduction algorithm uses bound to compute number of steps $T(n)$ that algorithm $A$ takes for problem
• Generates $T(n)$ copies of $M$ which feed the configuration to the copy of $M$
• Run circuit satisfiability on this converted problem
• Need to prove that reduction is polynomial
NP-completeness

- Can now reduce circuit satisfiability to another problem to show NP-hard
- 3-CNF is also NP complete
- Steps to reduce
  - Write parse tree
  - Add boolean variables to nodes
  - Convert to boolean formula
  - Write truth table
  - Convert to or’s of three anded terms
Clique Problem

- A clique in an undirected graph is a subset $V' \subseteq V$ of vertices, each pair of which is connected by an edge $E$.
- What is the maximum clique size in a graph?
- Corresponding decision problem: $\text{CLIQUE} = \{<G,k> : G \text{ is a graph with a clique of size } k\}$
- Naive algorithm is to list all $k$ size subsets and see if whether it forms a clique
NP-hard by reducing 3-CNF-SAT

- Let $C_1$ and $C_2$ and $C_3\ldots C_k$ be the 3-CNF formula.
- For each clause $C_i$ put triple of vertices $v_1'$, $v_2'$, and $v_3'$.
- Put an edge between two vertices $v_i'$ and $v_j'$ if both of the following hold:
  - $v_i'$ and $v_j'$ are in different triples.
  - There corresponding literals are consistent (they aren’t negations of each other).
Showing that it is a reduction

- Suppose the formula has a satisfying assignment.
  - Pick one “true” literal out of each term and select the corresponding vertex.
  - Between any two such vertices we have an edge
- We have k such vertices
- So we have a clique of size k
• Suppose we have a clique of size \( k \).
  • No edges of \( G \) connect vertices in the same triple, so \( V' \) contains exactly one vertex per triple
  • Can assign 1 to the corresponding literals without fear of assign 1 to the literal and its complement
  • Each clause is satisfied by this assignment