NP-Complete Reductions 1

Prof. Gregory Provan
Department of Computer Science
University College Cork
NP-Complete Reductions

- Notion of reduction
- Circuit Satisfiability: the first NP-complete problem
- Reduction process
- Sample reductions
  - Circuit-SAT $\rightarrow$ SAT
  - 2SAT Vertex $\rightarrow$ Cover
  - Vertex Cover $\rightarrow$ Clique
Today’s Learning Objectives

● Definitions
  - NP is the set of all problems (languages) that can be
    • accepted non-deterministically (using “choose” operations) in polynomial time.
    • verified in polynomial-time given a certificate y.

● Reduction techniques
  - Local replacement
  - Component design

● Some NP-complete problem reductions
  - Problem reduction
  - SAT (and CNF-SAT and 3SAT)
  - Vertex Cover
  - Clique
  - Hamiltonian Cycle
Some Thoughts about P and NP

- Belief: P is a proper subset of NP.
- Implication: the NP-complete problems are the hardest in NP.
- Why: Because if we could solve an NP-complete problem in polynomial time, we could solve every problem in NP in polynomial time.
- That is, if an NP-complete problem is solvable in polynomial time, then P=NP.
- Since so many people have attempted without success to find polynomial-time solutions to NP-complete problems, showing your problem is NP-complete is equivalent to showing that a lot of smart people have worked on your problem and found no polynomial-time algorithm.
A language $M$ is polynomial-time reducible to a language $L$ if an instance $x$ for $M$ can be transformed in polynomial time to an instance $x'$ for $L$ such that $x$ is in $M$ if and only if $x'$ is in $L$.

Denote this by $M \rightarrow_L L$.

A problem (language) $L$ is NP-hard if every problem in NP is polynomial-time reducible to $L$.

A problem (language) is NP-complete if it is in NP and it is NP-hard.

CIRCUIT-SAT is NP-complete:

- CIRCUIT-SAT is in NP
- For every $M$ in NP, $M \rightarrow$ CIRCUIT-SAT.
Transitivity of Reducibility

- If \( A \rightarrow B \) and \( B \rightarrow C \), then \( A \rightarrow C \).
  - An input \( x \) for \( A \) can be converted to \( x' \) for \( B \), such that \( x \) is in \( A \) if and only if \( x' \) is in \( B \). Likewise, for \( B \) to \( C \).
  - Convert \( x' \) into \( x'' \) for \( C \) such that \( x' \) is in \( B \) iff \( x'' \) is in \( C \).
  - Hence, if \( x \) is in \( A \), \( x' \) is in \( B \), and \( x'' \) is in \( C \).
  - Likewise, if \( x'' \) is in \( C \), \( x' \) is in \( B \), and \( x \) is in \( A \).
  - Thus, \( A \rightarrow C \), since polynomials are closed under composition.

- Types of reductions:
  - Local replacement: Show \( A \rightarrow B \) by dividing an input to \( A \) into components and show how each component can be converted to a component for \( B \).
  - Component design: Show \( A \rightarrow B \) by building special components for an input of \( B \) that enforce properties needed for \( A \), such as “choice” or “evaluate.”
History

- Cook proved the first problem to be NP-Complete (1970)
  - Satisfiability

- Karp (1972) then showed a number of decision problems were also NP-Complete

- Since then, hundreds of problems have been shown to be NP-complete
Satisfiability

- **Literal**: A variable or a negated variable. E.g. $x$, $\neg y$

- **Clause**: Literals connected with $\lor$
  - E.g. $(x \lor \neg y \lor \neg z)$

- A Boolean formula is in **conjunctive normal form**, or a **CNF-formula**, if it consists of clauses connected by $\land$
  - E.g. $\phi(x,y,z,w) = (x \lor \neg y \lor \neg z \lor w) \land (x \lor y \lor z \lor \neg w)$

- A CNF-formula $\phi$ is a **3-CNFS-formula** if each clause of $\phi$ has 3 literals.
  - E.g. $\phi(x,y,z,w) = (x \lor y \lor z) \land (x \lor \neg y \lor w)$

- **CNF-SAT** = $\{\langle \phi \rangle : \phi$ is a satisfiable CNF-formula$\}$

- **3-SAT** = $\{\langle \phi \rangle : \phi$ is a satisfiable 3-CNFS-formula$\}$
The Cook-Levin Theorem

- Theorem (Cook 1970, Levin 1972) SAT is NP-Complete. Further, CNF-SAT is also NP-Complete.

- Corollary 3-SAT is NP-Complete.

  - Proof of Corollary (next lecture)
Cook-Levin Theorem

- SAT is NP-complete.
  - We already showed it is in NP.
- To prove it is NP-hard, we have to show that every language in NP can be reduced to it.
  - Let $M$ be in NP, and let $x$ be an input for $M$.
  - Let $y$ be a certificate that allows us to verify membership in $M$ in polynomial time, $p(n)$, by some algorithm $D$.
  - Let $S$ be a circuit of size at most $O(p(n)^2)$ that simulates a computer (details omitted...)

$\text{CIRCUIT-SAT}$
A First NP-Complete Problem

A Boolean circuit is a circuit of AND, OR, and NOT gates; the CIRCUIT-SAT problem is to determine if there is an assignment of 0’s and 1’s to a circuit’s inputs so that the circuit outputs 1.

Logic Gates:
- NOT
- OR
- AND

Inputs:
- 0 → NOT → 1
- 1 → OR → 1
- 0 → AND → 0
- 1 → AND → 0

Output:
1
CIRCUIT-SAT is in NP

- Non-deterministically choose a set of inputs and the outcome of every gate, then test each gate’s I/O.
Lemma 34.5. The circuit-satisfiability problem belongs to the class NP.

Lemma 34.6. The circuit-satisfiability problem is NP-hard.

Proof. $L \leq_P CIRCUIT\_SAT \quad \forall L \in NP.$

Theorem 34.7. The circuit-satisfiability problem is NP-Complete.
Reducibility Chart

- CIRCUIT-SAT
  - SAT
    - 3CNF-SAT
      - Clique
        - VERTEX COVER
      - HAM-CYCLE
        - TSP
    - SUBSET-SUM
SAT

- A Boolean formula is a formula where the variables and operations are Boolean (0/1):
  - \((a+b+\neg d+e)(\neg a+\neg c)(\neg b+c+d+e)(a+\neg c+\neg e)\)
  - OR: +, AND: (times), NOT: \(\neg\)

- SAT: Given a Boolean formula \(S\), is \(S\) satisfiable, that is, can we assign 0’s and 1’s to the variables so that \(S\) is 1 ("true")?
  - Easy to see that CNF-SAT is in NP:
    - Non-deterministically choose an assignment of 0’s and 1’s to the variables and then evaluate each clause. If they are all 1 ("true"), then the formula is satisfiable.
SAT is NP-complete

- Reduce CIRCUIT-SAT to SAT.
  
  - Given a Boolean circuit, make a variable for every input and gate.
  
  - Create a sub-formula for each gate, characterizing its effect. Form the formula as the output variable AND-ed with all these sub-formulas:
    
    • Example: \( m((a+b)\leftrightarrow e)(c\leftrightarrow \neg f)(d\leftrightarrow \neg g)(e\leftrightarrow \neg h)(ef\leftrightarrow i) \ldots \)

The formula is satisfiable if and only if the Boolean circuit is satisfiable.
Vertex Cover

- A vertex cover of graph $G=(V,E)$ is a subset $W$ of $V$, such that, for every edge $(a,b)$ in $E$, $a$ is in $W$ or $b$ is in $W$.

- **VERTEX-COVER**: Given a graph $G$ and an integer $K$, is does $G$ have a vertex cover of size at most $K$?

- **VERTEX-COVER** is in NP: Non-deterministically choose a subset $W$ of size $K$ and check that every edge is covered by $W$. 
Vertex-Cover is NP-complete

- Reduce 3SAT to VERTEX-COVER.
  - Let $S$ be a Boolean formula in CNF with each clause having 3 literals.
  - For each variable $x$, create a node for $x$ and $\neg x$, and connect these two:

```
X    ¬X
```

- For each clause $(a+b+c)$, create a triangle and connect these three nodes.

```
a
  /\  /
 /   \
 b---c
```
Completing the construction

Connect each literal in a clause triangle to its copy in a variable pair.

E.g., a clause (¬x+y+z)

Let n= # of variables
Let m= # of clauses
Set K = n + 2m
Vertex-Cover is NP-complete

Example: \((a+b+c)(\neg a+b+\neg c)(\neg b+\neg c+\neg d)\)

Graph has vertex cover of size \(K=4+6=10\) iff formula is satisfiable.
Clique

- A clique of a graph $G=(V,E)$ is a subgraph $C$ that is fully-connected (every pair in $C$ has an edge).
- CLIQUE: Given a graph $G$ and an integer $K$, is there a clique in $G$ of size at least $K$?
- CLIQUE is in NP: non-deterministically choose a subset $C$ of size $K$ and check that every pair in $C$ has an edge in $G$.

This graph has a clique of size 5

CS 4407, Algorithms
University College Cork,
Gregory M. Provan
CLIQUE is NP-Complete

- Reduction from VERTEX-COVER.
- A graph $G$ has a vertex cover of size $K$ if and only if its complement has a clique of size $n-K$. 

![Graph G](image)

![Graph G'](image)
Lecture Summary

- **NP-Completeness proof steps for problem Q**
  - Q is in NP
  - Some NP-complete problem reduces to Q in polynomial time

- **Study of Reduction techniques**
  - Local replacement
  - Component design

- **Some NP-complete problem reductions**
  - Circuit-SAT → SAT
  - 3SAT → Vertex Cover
Some Other NP-Complete Problems

- **SET-COVER**: Given a collection of $m$ sets, are there $K$ of these sets whose union is the same as the whole collection of $m$ sets?
  - NP-complete by reduction from VERTEX-COVER

- **SUBSET-SUM**: Given a set of integers and a distinguished integer $K$, is there a subset of the integers that sums to $K$?
  - NP-complete by reduction from VERTEX-COVER
Some Other NP-Complete Problems

- **0/1 Knapsack**: Given a collection of items with weights and benefits, is there a subset of weight at most W and benefit at least K?
  - NP-complete by reduction from SUBSET-SUM

- **Hamiltonian-Cycle**: Given an graph G, is there a cycle in G that visits each vertex exactly once?
  - NP-complete by reduction from VERTEX-COVER

- **Traveling Salesperson Tour**: Given a complete weighted graph G, is there a cycle that visits each vertex and has total cost at most K?
  - NP-complete by reduction from Hamiltonian-Cycle.