Copyright Warning

COMMONWEALTH OF AUSTRALIA
Copyright Regulation 1969

WARNING

This material has been copied and communicated to you by or on behalf of **Curtin University of Technology** pursuant to Part VB of the *Copyright Act 1968* (**the Act**)

The material in this communication may be subject to copyright under the Act. Any further copying or communication of this material by you may be the subject of copyright protection under the Act.

Do not remove this notice

Curtin University of Technology

Theoretical Foundations of Computer Science 300

Lecture 9

NP-completeness

Curtin University of Technology

Outline

- NP-completeness
- Approximation algorithms.
- Probabilistic algorithms.
- Cryptography.

Curtin University of Technology

Assessment Criteria

- **Define** and relate the P, NP and the NP-complete classes.
- **Classify** a problem as NP-complete
- **Prove** the NP-completeness using mapping reducibility of an algorithm involving graphs or logical expressions.

Curtin University of Technology

NP-Completeness

- Cook and Levin (1970) discovered certain problems in NP whose individual complexity is related to that of the entire class.
- If a polynomial time algorithm exists for any of these problems, all problems in NP would be solvable in polynomial time.
- These problems are called NP-complete.

NP-Completeness

- Theoretical importance
  - ➢ researchers can focus on NP-complete problems to show that P is not equal to NP
  - ➢ to show that P=NP, it enough to find a polynomial time algorithm for a single NP-complete problem

- Practical use
  - ➢ showing that a problem is NP-complete is strong evidence that it will be hard to implement a general solution that performs well for "large" inputs
  - ➢ it is important to know this at the outset, so that alternatives can be considered

Curtin University of Technology

NP-completeness

- **Definition**: A language *B* is NP-complete if it satisfies two conditions:
  1. *B* is in NP, and
  2. every *A* in NP is polynomial time reducible to *B*.

- If *B* satisfies only condition 2, then *B* is NP-hard.

- **Theorem**: If *B* is NP-complete and $B \in P$ then P=NP.

- **Theorem**: If *B* is NP-complete and *B* is polynomial time reducible to *C* for *C* in NP, then *C* is NP-complete.

Curtin University of Technology

Polynomial time reducibility

- Reducing a problem *A* to problem *B* in polynomial time
  - ➢ so that a solution to *B* can be used to solve *A*.

- Language *A* is polynomial time reducible to language *B*, written $A \leq_P B$, if a polynomial time computable function *f* exists, where for every *w*,
  - ➢ $w \in A \Leftrightarrow f(w) \in B$.

- **Theorem**: If $A \leq_P B$ and $B \in P$, then $A \in P$.

Curtin University of Technology

SATISFIABILITY (SAT)

Cook-Levin Theorem

Curtin University of Technology

Cook-Levin theorem

- The Satisfiability problem (SAT)
  - Boolean variables: TRUE, FALSE (represented by 1,0)
  - Boolean operations: AND, OR, NOT ($\wedge$, $\vee$, $\neg$)
  - Boolean formula: expression involving Boolean variables and operations
  - *e.g.*, $\phi = (\neg x \wedge y) \vee (x \wedge z)$
  - A Boolean expression is satisfiable if some assignment of 0s and 1s to the variables makes it evaluate to 1.
  - $SAT = \{\langle \phi \rangle \mid \phi$ is a satisfiable Boolean formula$\}$
- **Theorem**: $SAT \in P$ iff P=NP.

Curtin University of Technology

Proof

1. Show that SAT is in NP.

An NP machine can guess an assignment to a given formula *f* and accepts if the assignment satisfies *f*.

4. Show that any language *A* in NP is polynomial time reducible to SAT.

- See Sipser or the Internet for details

Curtin University of Technology

NP COMPLETENESS BY REDUCTION

Theorem

3SAT

Curtin University of Technology

NP-completeness by Reduction Theorem

- Once we have one NP-complete problem, we may obtain others by polynomial time reduction from it.
- Establishing the first NP-complete problem is difficult.
- **Theorem:** If B is NP-complete and $B \leq_P C$ for C in NP, then C is NP-complete.

Curtin University of Technology

3SAT

- A literal is a Boolean variable or a negated Boolean variable (x or $\neg\, x$)
- A clause is several literals connected with $\vee$'s
- A Boolean formula is in ***conjunctive normal form*** (*cnf*) if it comprises several clauses connected with $\wedge$'s.
- 3*cnf* if all the clauses have three literals
- Example:

$$(x_1 \vee \neg\, x_2 \vee \neg\, x_3) \wedge (x_3 \vee \neg\, x_5 \vee x_6)\, (x_3 \vee \neg\, x_6 \vee x_4)$$

- $3SAT = \{<\phi>|\ \phi$ is a satisfiable 3*cnf* formula$\}$

Curtin University of Technology

3SAT

- *3SAT* is NP-complete?
  1. Show that *3SAT* is in NP.
  2. Show that *SAT* is polynomial time reducible to *3SAT*.

- Step 1 – Obvious(?)
- Step 2 proved by reduction of *SAT* to *3SAT*
  - ➢ Booleans can be converted to *cnf*
  - ➢ BUT Construction in *SAT* is to a *cnf*

$\bar{x}$

Curtin University of Technology

Conversion

- Let w $\in$ *SAT*
  - So w is a cnf – a *and* of terms $(x_1 \vee x_2 \vee x_3 \vee \ldots \vee x_n)$
  - $t=x_1$ converts to $f(t) = x_1 \vee x_1 \vee x_1$
  - $t=x_1 \vee x_2$ converts to $f(t) = x_1 \vee x_2 \vee x_2$
  - $t=x_1 \vee x_2 \vee x_3$ left as is
  - $t=(x_1 \vee x_2 \vee x_3 \vee \ldots \vee x_n)$ converts to $f(t) = (x_1 \vee x_2 \vee z_1) \wedge (\neg z_1 \vee x_3 \vee z_2) \wedge (\neg z_2 \vee x_4 \vee z_3) \wedge \ldots \wedge (\neg z_{n-2} \vee x_{n-1} \vee x_n)$
  - This conversion is polynomial

Curtin University of Technology

Reduction

- Note if $w \in SAT$ then $f(w) \in 3SAT$
  - ➢ As f(w) is conjunction of terms of 3 variables
  - ➢ Boolean algebra $w = f(w)$
    - An assignment so $w=1$, means $f(w)=1$
- If $f(w) \in 3SAT$ it is a Boolean expression
  - ➢ As $w = f(w)$ and there is an assignment that $f(w)=1$
  - ➢ So $w=1$ and so w satisfiable
  - ➢ So $w \in SAT$
- Thus *SAT* is polynomial time reducible to *3SAT*
- So *3SAT* is NP-complete

Curtin University of Technology

Clique

CLIQUE is NP-complete.

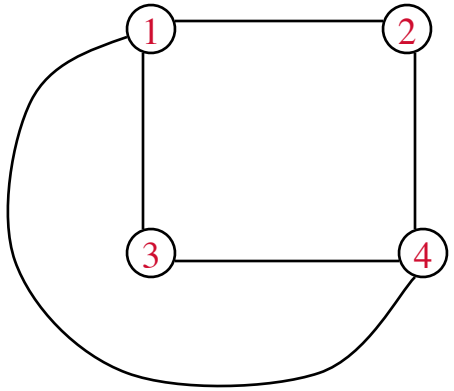
1. Show that CLIQUE is in NP.
2. Show that 3SAT is polynomial time reducible to CLIQUE (discussed before).

Curtin University of Technology

Independent Set

- Independent Set (IS)

  A subset of nodes, *I*, in an undirected graph *G* is an IS if no two nodes of *I* are connected by an edge of *G*.
- IS = {<G, *k*> | G is an undirected graph that has a *k*-independent set}



IS = {2,3}

Curtin University of Technology

Independent Set

- IS is NP-complete
  1. Show that IS in in NP.
  2. Show that 3SAT is polynomial time reducible to IS.

- Let ϕ be a formula with k clauses such as:

$$\phi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \ldots \wedge (a_k \vee b_k \vee c_k)$$

- The reduction f generates the string <G, k>, where G is an undirected graph with $3k$ nodes defined as:

Curtin University of Technology

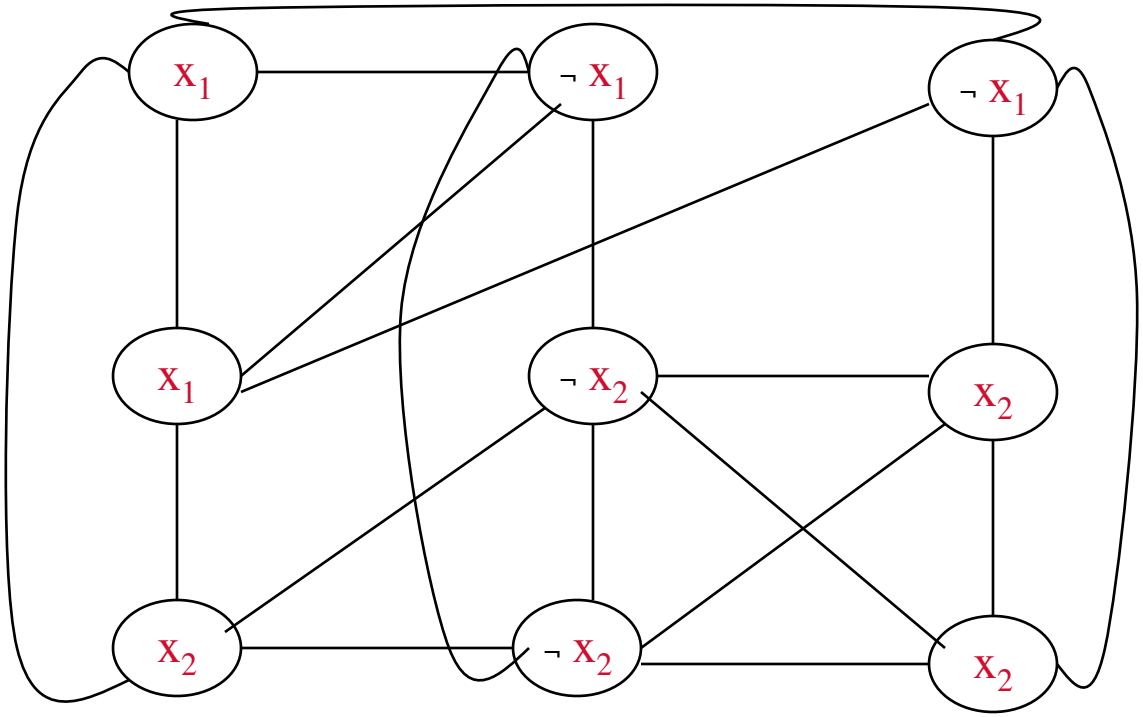
Independent Set

1. The nodes in G are organized into 3 rows and *k* columns.
2. Each column corresponds to one of the clause in ϕ
3. Each node in a column corresponds to a literal in the associated clause.
4. Label each node of G with its corresponding literal in ϕ .
5. The edges of G connect the following pairs of nodes in G:
   a. Between nodes in the same column.
   b. Between two nodes with contradictory labels.

Curtin University of Technology

Example

$$\phi = (x_1 \vee x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee x_2)$$



Curtin University of Technology

Example (Cont.)

- Have to demonstrate that the construction works.
- IS = $\{x_2, \neg x_1, \neg x_1\}$

- $\phi = (x_1 \vee x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee x_2) = 1$ when $x_2 = 1$ and $\neg x_1 = 1$.

Curtin University of Technology

Subset-Sum

- SUBSET-SUM is NP-complete.
  SUBSET-SUM = $\{<S,t> \mid S = \{x_1, \ldots, x_k\}$ and for some $\{y_1, \ldots, y_l\} \subseteq \{x_1, \ldots, x_k\}$ we have $\Sigma\, y_i = t\}$

- Example: $< \{ 4,11,16,21,27\},25> \in$ SUBSET-SUM
- Proof.
  1. Show that SUBSET-SUM is NP
  2. Show that 3SAT $\leq_p$ SUBSET-SUM

Curtin University of Technology

APPROXIMATION ALGORITHMS

Curtin University of Technology

Approximation algorithms

- Finding approximately optimal solutions
  - ➢ when the optimal solution is NP-hard no polynomial time algorithm exists unless P=NP
    - *B* is NP-hard if every *A* in NP is polynomial time reducible to *B* and *B* is not in NP.
  - ➢ a nearly optimal solution may be good enough and much easier to find

Curtin University of Technology

Example approximation algorithm

- Example:
  - Vertex cover of an undirected graph *G* is a subset of nodes where every edge of *G* touches one of these nodes.
  - Vertex cover problem is to find the size of the smallest vertex cover.
- Vertex-cover problem is NP-complete
  - the following polynomial time algorithm approximately solves it
  - finds a vertex cover that is not more than twice the min cover

Example approximation algorithm

- An approximation algorithm:
  - ➢ *A*= "On input <G>, where G is an undirected graph:
    1. Repeat the following until all edges in G touch a marked edge.
    2. Find an edge in G untouched by any marked edge.
    3. Mark that edge.
    4. Output all nodes that are endpoints of marked edges."

Curtin University of Technology

Approximation algorithm: proof

- *A* runs in polynomial time.
- Let *X* be the set of nodes it outputs, and *H* the set of edges it marks.
  - ➢ *X* is a vertex cover because *H* touches or contains every edge in *G*.
  - ➢ *X* is twice as large as *H*.
  - ➢ *H* is not larger than the smallest vertex cover *Y*. Every node in *H* is touched by some node in *Y*. No node in *Y* touches two edges in *H* because edges in *H* do not touch each other. So *Y* is at least as large as *H*.

Curtin University of Technology

PROBABILISTIC ALGORITHMS

Concept

Probabilistic TM

Examples

Curtin University of Technology

Probabilistic algorithms

- Designed to use the outcome of a random process
  - ➢ typically the algorithm may contain instruction to “flip a coin”
  - ➢ result of coin flip influences the subsequent execution and output
  - ➢ Certain types of problems more easily solvable than by deterministic algorithms

Probabilistic algorithms

- Can deciding by coin flips be better than actually calculating or estimating the best choice ?
  - ➢ calculating the best choice may need excessive time
  - ➢ estimation may introduce bias
  - ➢ *e.g.*, random sampling to determine information about individuals in a large population such as tastes or political preferences
    - querying all individuals may take too long
    - querying a nonrandom subset may give wrong results

Curtin University of Technology

Probabilistic TM

- A type of nondeterministic TM M where each nondeterministic step is called a *coin-flip step*
  - ➢ each such step has two legal next moves
  - ➢ a probability is assigned to each branch b of M's computation on input w
  - ➢ probability of branch b is $\Pr[b] = 2^{-k}$, where k is the number of coin-flip steps on branch b.
  - ➢ probability that M accepts w is $\Pr[M \text{ accepts } w] = \Sigma \Pr[b]$, where b is an accepting branch.

Curtin University of Technology

Probabilistic TM

- *M* recognizes language *A* with error probability ε for $0 \leq \varepsilon \leq 0.5$ if
  - 1. $w \in A$ implies Pr[*M* accepts *w*] ≥ 1- ε
  - 2. $w \notin A$ implies Pr[*M* rejects *w*] ≥ 1- ε
- Also consider error probability bounds that depend upon the input length *n*
  - e.g., $\varepsilon = 2^{-n}$ indicates an exponentially small probability of error

Curtin University of Technology

Bonus problem - Minesweeper

- Let G be an undirected graph, where each node either contains a single, hidden mine or is empty. The player chooses nodes, one by one. If the player chooses a node containing a mine, the player loses. If the player chooses an empty node, the player learns the number of neighbouring nodes containing mines. (A neighbouring node is one connected to the chosen node by an edge.). The player wins if and when all empty nodes have been so chosen.
- In the mine consistency problem you are given a graph G, along with numbers labelling some of G's nodes. You must determine whether a placement of mines on the remaining nodes is possible, so that any node v that is labelled *m* has exactly *m* neighbouring nodes containing mines. Formulate this problem as a language and show that it is NP-complete.

Curtin University of Technology

Summary

- Problems that are shown to be NP-complete are difficult to find a good, general solution for.

- Tackle these problems by relaxing the requirements (approximation, probabilistic), solving only a part of the problem or limiting the size of the inputs.