COT 6405 Introduction to Theory of Algorithms

Topic 17. NP-complete problems

What we have covered

- **The role of computing (Ch.1)**
- **Analysis of Algorithms: Insertion Sort, Merge Sort (Ch.2)**
- **Growth functions, Asymptotic Notations (Ch.3)**
- **Divide and Conquer, Recurrences (Ch.4)**
- **Heapsort (Ch.6), Quicksort (Ch.7)**
- **Dynamic Programming (Ch. 15)**
- **Greedy Algorithms (Ch.16)**
- **Linear-time Sorting, Lower Bounds, Counting Sort, etc. (Ch.8)**
- **Elementary data structure (Ch.10) (Self study)**
- **Hash Tables (Ch.11) , Binary Search Trees (Ch.12)**
- **Elementary graph algorithms, representation. DFS. BFS (Ch.22)**
- **Minimum Spanning Tree: Prim/ Kruskal algorithm. (Ch.23)**
- **Single source shortest path: Dijkstra's, Bellman-Ford (Ch.24)**
- **NP-completeness (Ch.34)**

Classification of Problems

Which problems will we be able to solve in practice?

Yes
Shortest path
Minimum spanning tree
BFS
DFS
Sorting
Order statistics

Probably no
Longest path
Subset-sum
0/1 Knapsack
3CNF-SAT
Hamiltonian-cycle
Vertex cover

Classification of Problems

1. Tractable problems
2. Intractable problems
3. Impossible problems

Tractable Problems

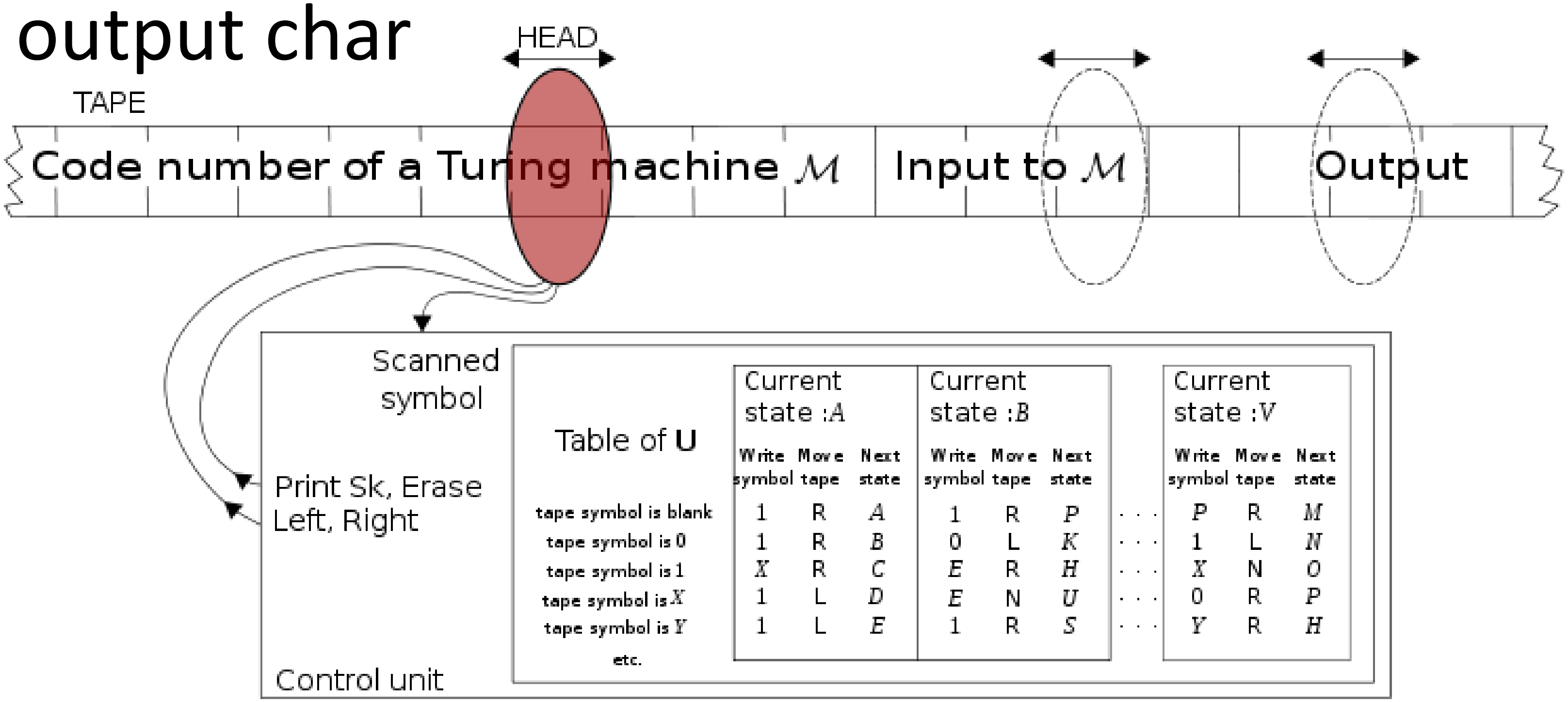
- We have generally studied tractable problems (solvable in polynomial time).
- Algorithm design patterns: Examples.
  - Divide-and-conquer: O(n log n) merge sort.
  - May long, but we can solve them in known time

Intractable Problems

- There are other problems that probably require exponential time
- Examples:
  - Given a Turing machine, does it halt in at most k steps on any finite input?
  - In 1936, Alan Turing presented a model for the first digital computer, which is today known as the Turing Machine.

Turing Machine

- A state machine, a tape with symbols, a head
- Based on (Current state, input char)
- Transit into a New state, head move left/right, output char



Impossible Problems

- There are other problems that cannot be solved by any algorithms
  - E.g., Fermat's Last Theorem: no three positive integers *a*, *b*, and *c* can satisfy the equation $a^n + b^n = c^n$ for any integer value of $n > 2$
  - it is considered an impossible problem until 1995, proved by Andrew Wiles, > 350 years
  - The proof itself is over 100 pages long and consumed seven years of Wiles's research time.
- Why do we need to know this as an engineer?
  - We need to know that some problems cannot be solved efficiently
  - To develop approximation algorithms, instead of trying to find an efficient solution

The Halting Problem

- The halting problem is a particular decision problem
  - Given a description of a program and a finite input, decide whether the program will halt, or run forever on that input
  - Can we solve this?
- A general algorithm to solve the halting problem for all possible “program-input” pairs cannot exist: The halting problem is undecidable.

The Halting Problem: two examples

- Suppose your program consists of the instruction "take every number between 1 and 10, add 2 to it and then output the result".
  - It's obvious that this program halts after 10 steps
- If the instructions are "take a number x, which is not negative, and keep multiplying it by 2 until the result is bigger than 1",
  - then the program will stop as long as the input x is not 0. If it is 0, it will keep going forever.

The Halting Problem

- In these two examples, it is easy to see whether the program stops or not.
- But what if the program is much more complicated?
- Of course, we can simply run it and see if it stops
  - but how long should you wait before you decide that it doesn't? A week, a month, a year?
- The basic question is whether there is a test which in a finite amount of time decides whether or not *any* given program ever halts.
  - as Turing proved, the answer is no.

Polynomial time algorithms

- Most algorithms we have studied so far are polynomial time in the size of their inputs n
  - We call them “in P”
- Worst case running time is $O(n^k)$ for some constant k.
- Problems that are solvable by polynomial time algorithms are said to be tractable.

What's so great about polynomial time?

1. Although we would consider a problem that is $\Omega(n^{100})$, this type of problem is rarely encountered.
   - The exponents on the polynomial are usually small.
2. For most reasonable models of computation, a problem that can be solved in polynomial time in one model can be solved in polynomial time in another mode

Do all problems have polynomial time solutions?

NO!

- Some problems are not solvable
  - Turing's Halting problem is an example
  - Cannot be solved by any computer no matter how much time is given
- There is a large class of important problems for which we do not know the answer.
- These are the NP complete problems.

Many P and NP-Complete Problems are Closely Related

- Shortest path problem and longest path problem
- Euler tour and Hamiltonian Cycle
  - Euler tour of connected, directed graph visits **each edge** exactly once
  - Hamiltonian cycle begins and ends at the same vertex and visits **each vertex** exactly once
- 2-SAT and 3-SAT
  - 2-Conjunctive Normal Form (CNF) Satisfiability
    - $(x1 \vee \neg x2) \wedge (\neg x1 \vee x2) \wedge (x1 \vee x2)$
  - 3-Conjunctive Normal Form (CNF) Satisfiability
    - $(x1 \vee \neg x2 \vee x3) \wedge (\neg x1 \vee x2 \vee x3) \wedge (x1 \vee x2 \vee \neg x3)$

The Satisfiability Problem

- Given a Boolean formula containing variables whose value are 0 or 1, connected by the Boolean connectives $\wedge$**,** $\vee$**, and** $\neg$.
- A Boolean formula is satisfiable if there is an assignment of values to variables that cause the formula to evaluate to 1
- k-SAT
  - Informal definition: A formula is in k-conjunctive normal form, if it is the AND of clauses of ORs of exactly k variables (or their negations)
- 3-SAT is NP-complete
  - $(x1 \vee \neg x2 \vee \mathbf{x3}) \wedge (\neg x1 \vee x2 \vee \mathbf{x3}) \wedge (x1 \vee x2 \vee \neg x3)$

Three Classes of Problems

- P (Polynomial)
  - Problems solvable in polynomial time
  - Can be solved in time $O(n^k)$ where k is a constant
- NP (Non-deterministic Polynomial)
  - Problems that are <u>verifiable</u> in polynomial time
  - Given a "<u>certificate</u>" of a solution, we can verify that the solution is correct in polynomial time
    - For Hamiltonian path, the certificate is a sequence of vertices
    - For 3-SAT (3-Conjunctive Normal Form Satisfiability), the certificate is an assignment of values to variables
- NPC (NP Complete)

NP Complete Problems

- The complexity status of these problems is unknown
- There is no established lower bound
- But, it is known that if one of these problems can be solved in polynomial time, all of them can.

What Do We Think?

- Most computer scientists believe that the NP complete problems are intractable.
- Why?
  - People have been trying to find efficient algorithms for them for a long time and no one has succeeded.

NP Complete Problems

- Informal definition:
  - A problem is in the class NPC if it is in NP and is as "hard" as any problem in NPC
  - No polynomial time solutions but "hard"
- If any problem in the class NPC can be solved in polynomial time, then all can be solved in polynomial time

Proofs of NP-Completeness

- Different from other proofs we have done
- Rather than show that a problem has an efficient algorithm, we will be demonstrating that it is hard
  - By mapping it to another 'hard' problem

Some NP-complete Problems

- Summarization
  - CIRCUIT-SAT -- Cook-Levin theorem
  - 3-SAT -- reduced from CIRCUIT-SAT
  - VERTEX-COVER – reduced from 3-SAT
  - CLIQUE – reduced from VERTEX-COVER
- 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

Some Other NP-Complete Problems

- **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
- **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 Salesman 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.

Theory of NP-Completeness

- Limited to one type of problems - decision problems
  - Each element of S (the set of solutions) is an element of {yes, no} (or {0, 1})
  - Usually dealing with <u>optimization problems</u> that are easily restated as a decision problem.
  - We can show that if the decision problem is easy, then the optimization problem is easy.

Optimization Problem vs Decision Problem

- Shortest path problem as an optimization problem

  Given a graph G= (V,E), two vertices u, v $\in$ V, what is the shortest path that exists in G between u and v?

- Shortest path problem as a decision problem

  Given a graph G= (V,E), two vertices u, v $\in$ V and a nonnegative integer k, does a path exist in G between u and v whose length is at most k?

  – k hops away

General Approach

To change an optimization problem to a decision problem, we can usually just impose a bound on the value to be optimized.

- minimization problem example:
  - length of shortest path at most k
- maximization problem example:
  - length of longest path at least k

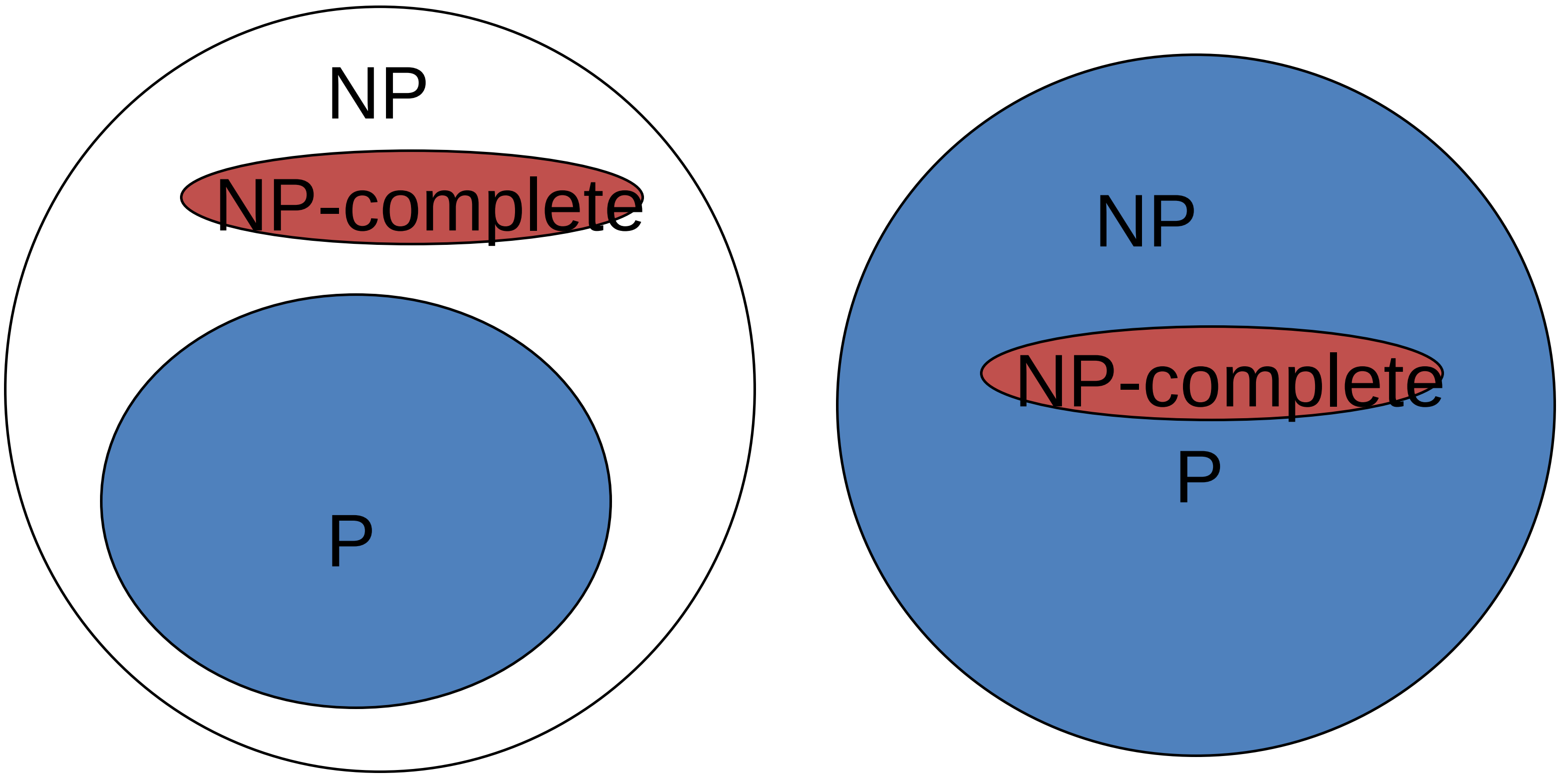
General statements

- Usually, if we can solve the optimization problem quickly, we can solve the decision problem quickly.
- Also, if we can provide evidence that the decision problem is hard, we also provide evidence that the optimization problem is hard.

Nondeterministic Polynomial (NP)

- Let an oracle guess an answer to the problem (certificate).
- If we can find an algorithm that can verify that the answer is correct in polynomial time, then this problem is in the class NP
- P is clearly a subset of NP
  - The complexity class P is the set of concrete decision problems that are solvable in polynomial time.

P = NP?

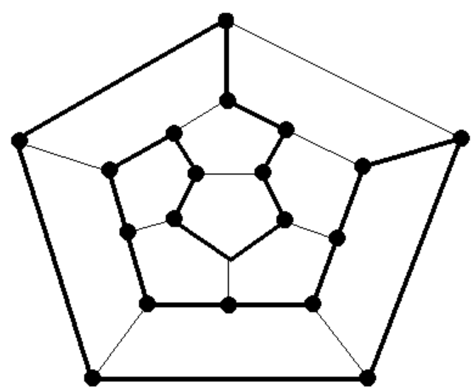


Theorem

- If any NP-complete problem is polynomial-time solvable, then P = NP.
- All problems in NP can be reduced to a NP-complete problem in polynomial time

Example: Hamiltonian Cycle Problem, HAM-CYCLE

- A Hamiltonian cycle of an undirected graph G=(V,E) is a simple cycle that contains each vertex in V.
- The Hamiltonian-cycle problem asks if a given graph contains a Hamiltonian cycle.



A Hamiltonian Cycle
on the Dodecahedron

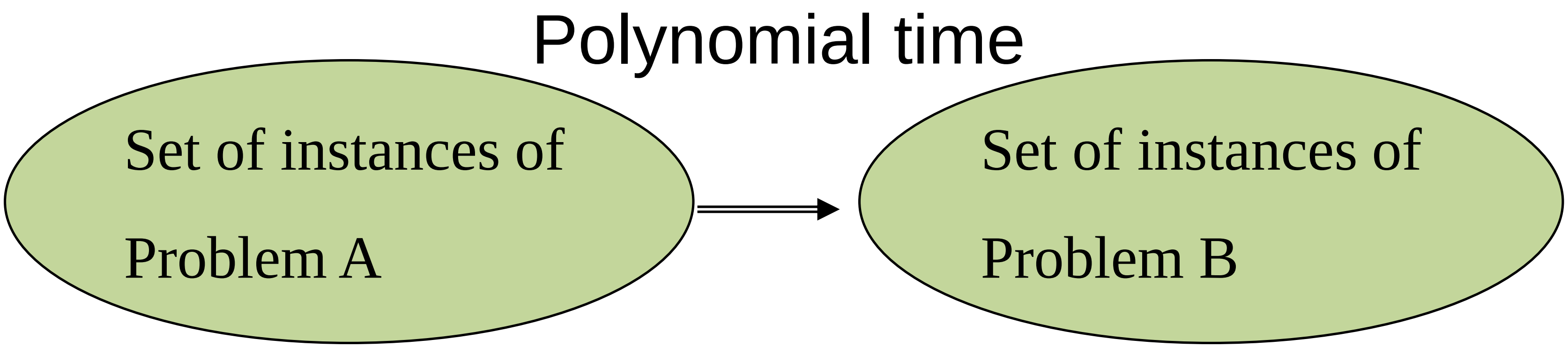
HAM-CYCLE $\in$ NP

- HAM-CYCLE is an element of NP, because we can verify an answer
  - Given a list of vertices, does the list represent a Hamiltonian cycle?
  - We can find a polynomial time algorithm to determine if this list is a Hamiltonian cycle.
    - Is the first vertex the same as the last vertex?
    - Are all vertices visited?
    - For each vertex in the list, is there an edge to the next vertex in the list?
    - Is any vertex other than the first repeated

Showing Problems to be NP- Complete

- Restate a problem as a decision problem
- Demonstrate that the decision problem is in the class NP
- Show that a problem known to be NP-Complete can be reduced to the current problem in polynomial time.

Reducibility



Decision problem A is polynomial-time reducible to decision problem B if a polynomial time algorithm can be developed

- which changes each instance of problem A to an instance of problem B
- such that: if the answer for an instance of B is yes, the answer to the corresponding instance of A is yes.

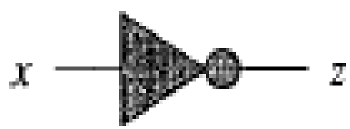
A First NP-Complete Problem

- This technique for showing that a problem is in the class NP-Complete requires that we have one NP-Complete problem to begin with
- <u>Circuit satisifiability </u>was the first problem to be shown to be in NP-Complete.
- Cook's Theorem
  - Stephen Arthur Cook

Sketch of NP-Completeness Proof for CIRCUIT-SAT

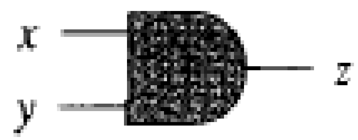
- Very long and difficult in the details.
- Used to provide the basis for most other proofs of NP-Completeness.
- Problem domain is combinatorial circuits.

Combinatorial Circuit Components



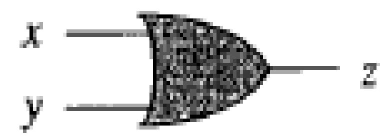
x	$\neg x$
0	1
1	0

(a)



x	y	$x \wedge y$
0	0	0
0	1	0
1	0	0
1	1	1

(b)

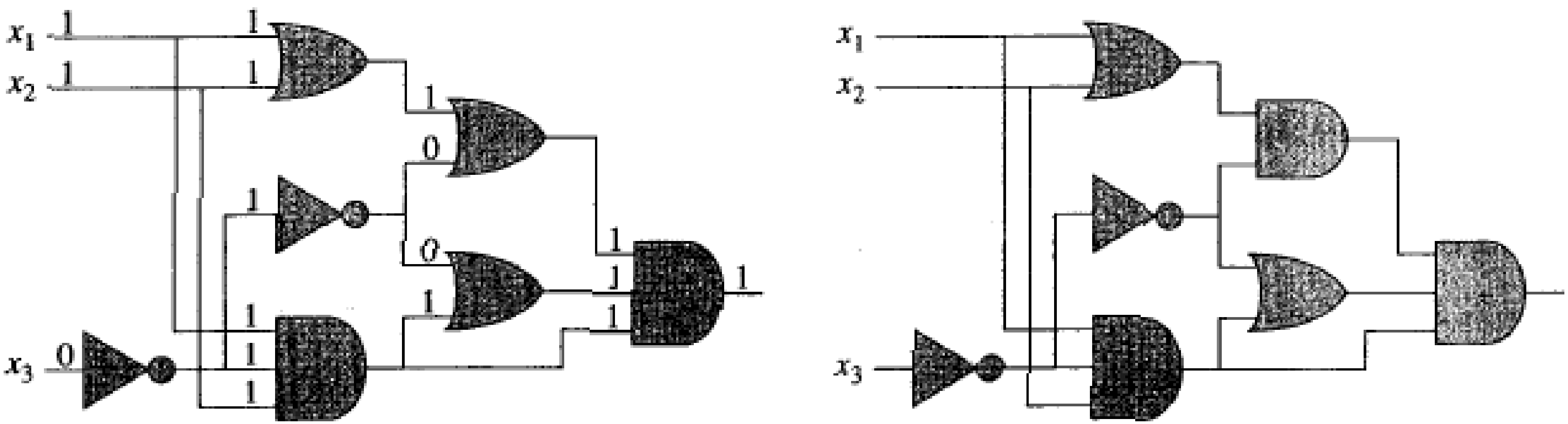


x	y	$x \vee y$
0	0	0
0	1	1
1	0	1
1	1	1

(c)

Circuit-Satisfiability Problem

Given a Boolean combinatorial circuit composed of AND, OR, and NOT gates, is it satisfiable? The output is 1.



(a) (b)

Demonstrating Circuit Satisfiability is NP Complete

- This proof was done from first principles.
- It did not depend on the existence of any other NPC problems.
- A problem Q is NP-Complete if
  1. It is an element of the class NP
  2. $Q' \leq_p Q$ for every Q' in NP

Step 1: Prove CIRCUIT-SAT $\in$ NP

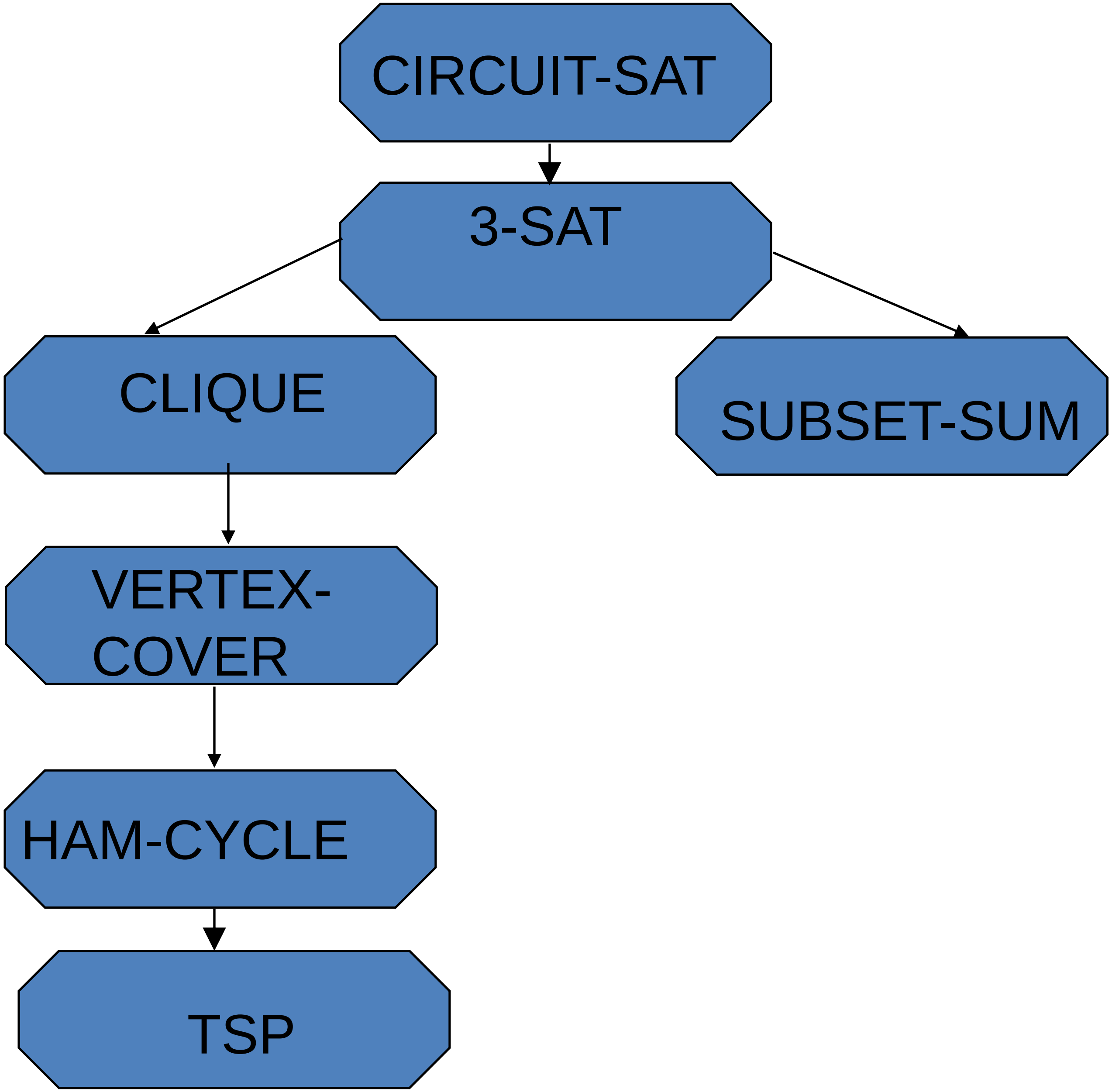
- Develop an algorithm A that can verify a solution in polynomial time.
- Inputs for A
  - A standard representation of a combinatorial circuit
  - An assignment of input values to the circuit
- Algorithm A determines the output of the combinatorial circuit. Output is 0 or 1.
- This can be done in polynomial time (linear for a clever algorithm.)
- Therefore, CIRCUIT-SAT $\in$ NP

Step 2: Reduction

- We need to show that every other problem in NP can be reduced to CIRCUIT-SAT.
- The basic argument is:
  - A combinatorial circuit can be used to implement a computer: Program, Memory, Program counter, Working storage, etc.
  - We can show that every problem in NP can be mapped onto operations that can be represented as a sequences of states of combinatorial circuits.
  - We show that this can be done in polynomial time.
  - It is not simple to show all of this, but in the end we can show that any problem in NP can be reduced to CIRCUIT-SAT in polynomial time

Implications of CIRCUIT-SAT$\in$NPC

- CIRCUIT-SAT is the “seed” problem in NPC
- Once we know that one problem is in NPC, we can use it to demonstrate that other problems are in NPC using a simpler procedure.


CIRCUIT-SAT
3-SAT
CLIQUE
SUBSET-SUM
VERTEX-
COVER
HAM-CYCLE
TSP

Summary

- Classification of problems
- P and NP
- Well-known NP complete problems
- Optimization problems v.s. decision problems
- General ways to prove whether or not a problem is in NP and NPC
- Common mistakes: NPC stands for non-polynomial. This is actually unknown.