Problem Complexity

Algorithm Complexity

- Good algorithms:

  logarithmic: **O(log n)**
  linear: **O(n)**
  Subquadratic: **O(n log n)**
  quadratic: **$O(n^2)$**
  Cubic: **$O(n^3)$**

  Polynomial Algorithms

- Bad algorithms:

  exponential: **$O(a^n)$, a > 1**
  Factorial: **O(n!)**

  Exponential Algorithms

Algorithm Complexity (cont.)

n	log n	n	n^2	2^n
2	1	2	4	4
4	2	4	16	16
8	3	8	64	256
16	4	16	256	65,536
32	5	32	1,024	4,294,967,296

P, NP and NP-Complete Problems

Problems with Polynomial Complexity

- Most of the problems studied so far have Polynomial upper bounds with input size, *n*.
  - **O(1)** **Constant**
  - **O(log *n*)** **Sub-linear**
  - **O(*n*)** **Linear**
  - **O(*n* log *n*)** **Nearly linear**
  - **O(n^2)** **Quadratic**
- And have polynomial lower bound

Complexity class P

P is a class of problems which can be solved by a polynomial time algorithm.

Tractable vs. Intractable

P-class Problems are tractable

Problems are intractable if the upper and lower bounds have an exponential factor.

- O(n!)
- $O(n^n)$
- $O(2^n)$

Problems that Cross the Line

- What if a problem has:
  - An exponential upper bound
  - A polynomial lower bound
- We have only found exponential algorithms, so it appears to be intractable.
- But... we can’t prove that an exponential solution is needed, we can’t prove that a polynomial algorithm cannot be developed, so we can’t say the problem is intractable...

Complexity class NP

Non-deterministic **P**olynomial

NP is a class of problems with the following property:

- If someone tells us a solution to the problem,
- we can verify it in polynomial time.

Complexity class NP

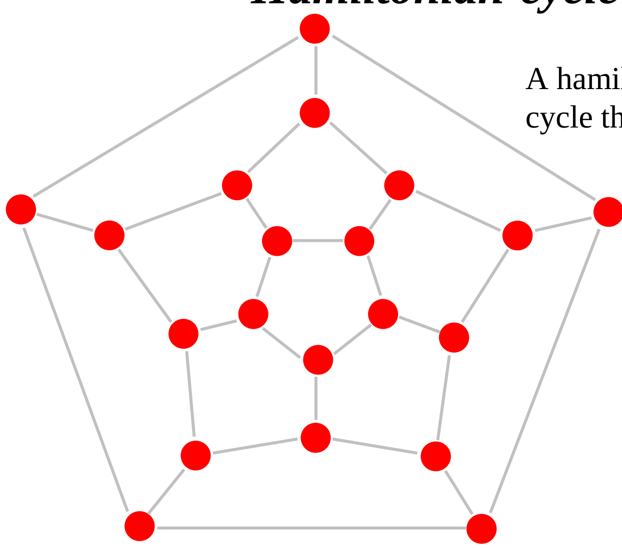
The name **NP** comes from a 2-step process to solve them:

- There exists a non-deterministic method that generates a possible solution to the problem in polynomial time.
- There exists a deterministic method that verifies the solution and determines in polynomial time if it is a true solution.

Determinism vs. Nondeterminism

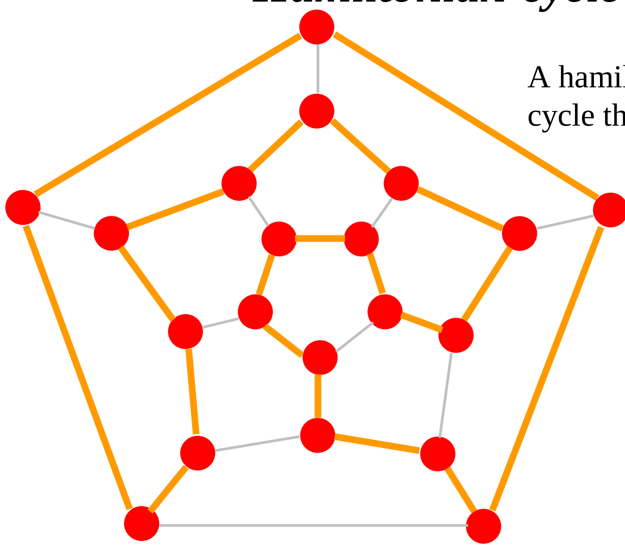
- Nondeterministic algorithms produce an answer by a series of “guesses”
- Deterministic algorithms make decisions based on information.

Complexity class NP
Hamiltonian-cycle problem



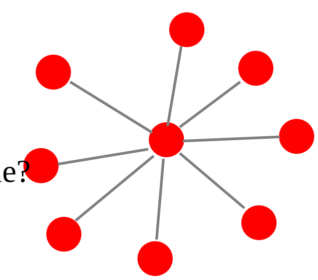
A hamiltonian cycle is a simple cycle that contains each vertex.

Complexity class NP
Hamiltonian-cycle problem



A hamiltonian cycle is a simple cycle that contains each vertex.

Complexity class NP
Hamiltonian cycle problem



Not all graph have hamiltonian cycle.

Hamiltonian cycle problem:
Does a given graph have a hamiltonian cycle?

No polynomial-time algorithm is known to solve HAM-CYCLE.

If someone shows us a hamiltonian cycle in a graph, then we can verify it in polynomial time.

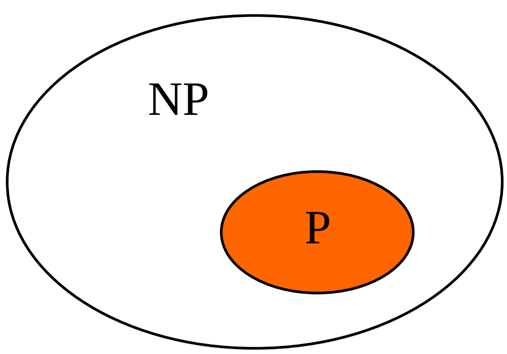
$\Rightarrow$ HAM-CYCLE $\in$ NP

P And **NP** Summary

- **P** = set of problems that can be solved in polynomial time
- **NP** = set of problems for which a solution can be verified in polynomial time

Complexity classes P and NP

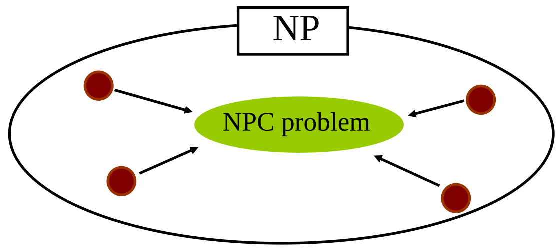
It can be seen easily that P $\subseteq$ NP.
Conjecture: P $\neq$ NP.



Definition of NP-completeness

A problem is **NP-complete** if

1. it is in the class NP and
2. every problem in NP is polynomial time reducible to it.



Polynomial time reducibility

We call the input to a particular problem an instance of that problem.

We say that a problem L is *polynomial time reducible* to a problem M if any instance of L can be rephrased to an instance of M in polynomial time.

$$L \xrightarrow{poly} M$$

•Intuitively: If L reduces in polynomial time to M, L is "no harder to solve" than M.

NP-Complete

"NP-Complete" comes from:

- Nondeterministic Polynomial
- Complete - "Solve one, Solve them all"

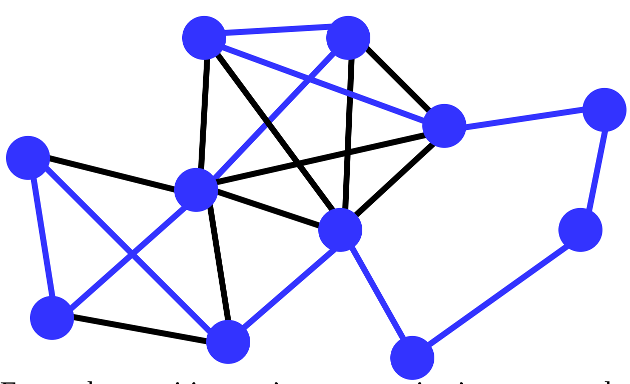
NP-Complete Problems

- The upper bound suggests the problem is intractable
- The lower bound suggests the problem is tractable
- The lower bound is linear: O(N)
- They are all reducible to each other
  - If we find a reasonable algorithm (or prove intractability) for one, then we can do it for all of them!

Example NP-Complete Problems

- Path-Finding (Traveling salesman)
- Map coloring
- Scheduling and Matching (bin packing)
- 2-D arrangement problems
- Planning problems (pert planning)
- Clique

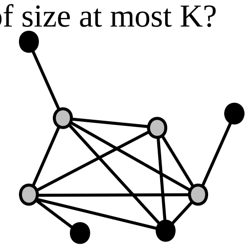
Traveling Salesman



For each two cities, an integer cost is given to travel from one of the two cities to the other one.
The salesman wants to make a minimum cost "circuit" visiting each city exactly once.

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 an graph G and an integer K, does G have a vertex cover of size at most K?

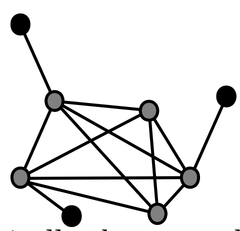


- VERTEX-COVER is in NP: Non-deterministically choose a subset W of size K and check that every edge is covered by W.

Clique

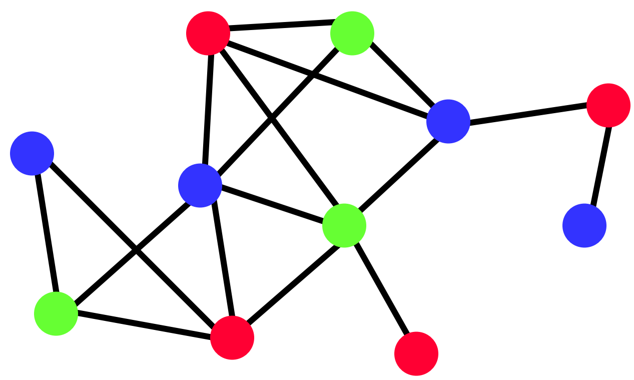
- A **clique** of a graph G=(V,E) is a subgraph C that is fully-connected (every pair in C has an edge).
- CLIQUE: Given a graph G and an integer K, is there a clique in G of size at least K?

This graph has a clique of size 5



- CLIQUE is in NP: non-deterministically choose a subset C of size K and check that every edge pair in C has an edge in G.

Map Coloring



Class Scheduling Problem

- With N teachers with certain hour restrictions M classes to be scheduled, can we:
  - Schedule all the classes
  - Make sure that no two teachers teach the same class at the same time
  - No teacher is scheduled to teach two classes at once

SAT

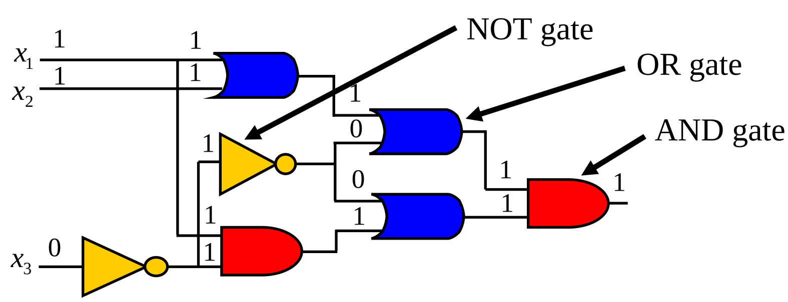
- A Boolean formula is a formula where the variables and operations are Boolean (0/1)
- SAT: Given a Boolean formula S, is S satisfiable, that is, can we assign 0's and 1's to the variables so that S is 1 ("true")?
- CNF-SAT is in NP:
  - Non-deterministically choose an assignment of 0's and 1's to the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

Conjuctive Normal Form

– $(a+b+\neg d+e)(\neg a+\neg c)(\neg b+c+d+e)(a+\neg c+\neg e)$

– OR: +, AND: (times), NOT: $\neg$

The circuit-satisfiability problem



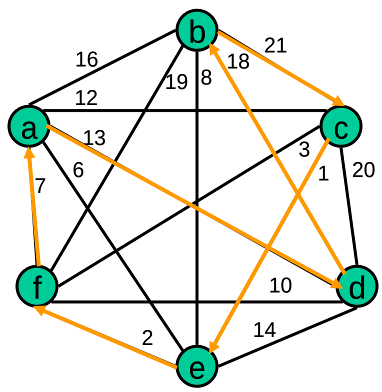
CIRCUIT-SAT: Given a boolean combinational circuit composed of AND, OR and NOT gates, is it satisfiable?
In other words, is there an input that causes the output to be "true".

Approximation Algorithms

Approximation Solution of the TSP Problem

- OPT-TSP: Given a complete, weighted graph, find a cycle of minimum cost that visits each vertex.
  - OPT-TSP is NP-Complete
- Approximation: Greedy Algorithm
  - Choose edges in increasing order
  - Add to the path if
    - The edge does not form a cycle with the existing edges (unless all the nodes are in the cycle)
    - The edge is not a 3rd edge connected to some node.

Approximation Solution of the TSP Problem

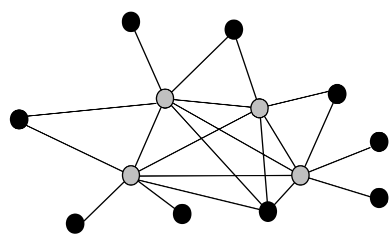


Approximation Ratios

- Optimization Problems
  - We have some problem instance x that has many feasible "solutions".
  - We are trying to minimize (or maximize) some cost function c(S) for a "solution" S to x. For example,
    - Finding a smallest vertex cover of a graph
    - Finding a smallest traveling salesperson tour in a graph
- An approximation produces a solution T
  - T is a **k-approximation** to the optimal solution OPT if
    - $c(T)/c(OPT) \leq k$ for a minimization problem
    - $c(OPT)/c(T) \leq k$ for a maximization problem

Vertex Cover

- A **vertex cover** of graph G=(V,E) is a subset W of V, such that, for every (a,b) in E, a is in W or b is in W.
- OPT-VERTEX-COVER: Given an graph G, find a vertex cover of G with smallest size.
- OPT-VERTEX-COVER is NP-hard.



A 2-Approximation for Vertex Cover

Algorithm ***VertexCoverApprox***(***G***)
    Input graph ***G***
    Output a small cover ***C*** for ***G***
    C ← empty set
    H ← ***G***
    while ***H*** has edges
        e ← ***H.removeEdge***(***H.anEdge***())
        v ← ***H.origin***(***e***)
        w ← ***H.destination***(***e***)
        C.add(***v***)
        C.add(***w***)
        for each ***f*** incident to ***v*** or ***w***
            H.removeEdge(***f***)
    return ***C***

A 2-Approximation for Vertex Cover

- Every chosen edge e has both ends in C
- But e must be covered by an optimal cover; hence, one end of e must be in OPT
- Thus, there is at most twice as many vertices in C as in OPT.
- That is, C is a 2-approx. of OPT
- Running time: O(m)