P, NP, NP-Complete, and NP-Hard

by
Dr. Animesh Chaturvedi
Assistant Professor: LNMIIT Jaipur
Post Doctorate: King's College London & The Alan Turing Institute
PhD: IIT Indore

Computational complexity theory

- Fields in
  - Theoretical Computer Science
  - Analysis of Algorithms
- An algorithm solves a computation problem by mathematical steps.
- A computational problem (such as an algorithm) is a task solved by a computer.
- Focuses on classifying computational problems according to the resource usage
- Resource usage: amount of resources needed to solve computational problem,
- Resources: such as time and storage.

https://en.wikipedia.org/wiki/Computational_complexity_theory

Single Source Shortest-Paths Implementation

algorithm	restriction	typical case	worst case	extra space
topological sort	no directed cycles	$E + V$	$E + V$	V
Dijkstra (binary heap)	no negative weights	$E \log V$	$E \log V$	V
Bellman-Ford	no negative cycles	$E V$	$E V$	V
Bellman-Ford (queue-based)		$E + V$	$E V$	V

Easy

Medium

Hard

Remark 1. Directed cycles make the problem harder.

Remark 2. Negative weights make the problem harder.

Remark 3. Negative cycles makes the problem intractable.

ROBERT SEDGEWICK | KEVIN WAYNE. *Algorithms* 4th Edition, https://algs4.cs.princeton.edu/44sp/

Single Source Shortest-Paths Implementation

algorithm	restriction	typical case	worst case	extra space
topological sort	no directed cycles	$E + V$	$E + V$	V
Dijkstra (binary heap)	no negative weights	$E \log V$	$E \log V$	V
Bellman-Ford	no negative cycles	$E\,V$	$E\,V$	V
Bellman-Ford (queue-based)		$E + V$	$E\,V$	V

Remark 1. Directed cycles make the problem harder.

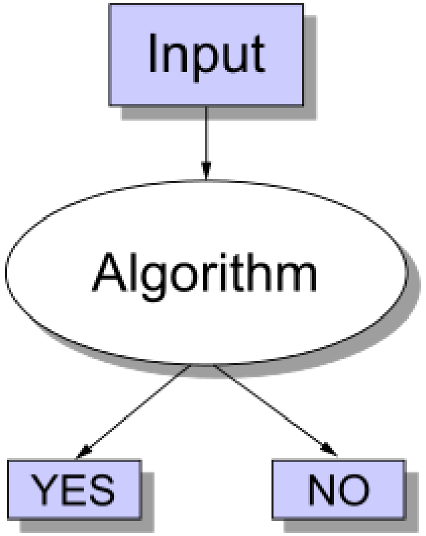
Remark 2. Negative weights make the problem harder.

Remark 3. Negative cycles makes the problem intractable.

ROBERT SEDGEWICK | KEVIN WAYNE. *Algorithms* 4th Edition, https://algs4.cs.princeton.edu/44sp/

Deterministic algorithm

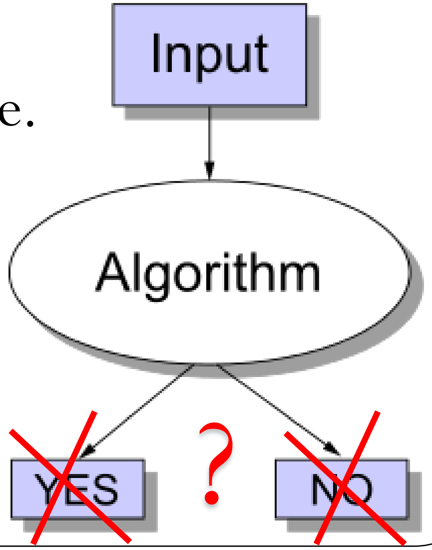
- Given a particular input, will always produce the same output, with the underlying machine always passing through the same sequence of states.
- State machine: a state describes what a machine is doing at a particular instant in time.
- State machines pass in a discrete manner from one state to another.
- Enter the input, initial state or start state.
- Current state determines what will be next state, the set of states is predetermined.



https://en.wikipedia.org/wiki/Deterministic_algorithm

Non-deterministic Algorithms

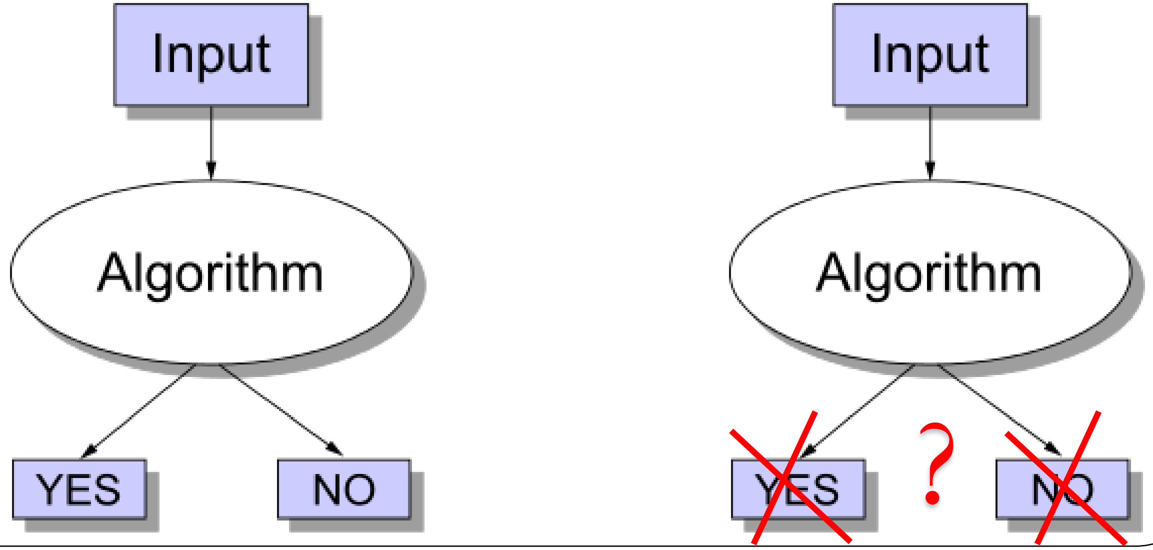
- If it uses external state other than the input, such as
  - user input,
  - a global variable,
  - a hardware timer value,
  - a random value, or
  - stored disk data.
- If it is timing-sensitive,
  - e.g. if it has multiple processors writing to the same data at the same time.
- If a hardware error causes its state to change in an unexpected way.
- The order each processor writes data will affect the result.



https://en.wikipedia.org/wiki/Deterministic_algorithm

Deterministic and Non-deterministic Algorithms

- Disadvantages of Determinism
  - predictable future by players or predictable security by hacker
  - e.g. predictable card shuffling program or security key
- Pseudorandom number generator is often not sufficient,
  - thus cryptographically secure pseudo-random number generator,
  - hardware random number generator.



https://en.wikipedia.org/wiki/Deterministic_algorithm

P (Polynomial) Time Problems

- Contains all decision problems that can be solved deterministically using a polynomial time (polynomial amount of computation time).
- A problem is in P-complete, if it is in P
- P is the class of computational problems that are "efficiently solvable" or "tractable".
- Class P, typically take all the "tractable" problems for a sequential algorithm,
- But, in reality, some problems not known to be solved in polynomial P time.

https://en.wikipedia.org/wiki/P_(complexity)
https://en.wikipedia.org/wiki/P-complete

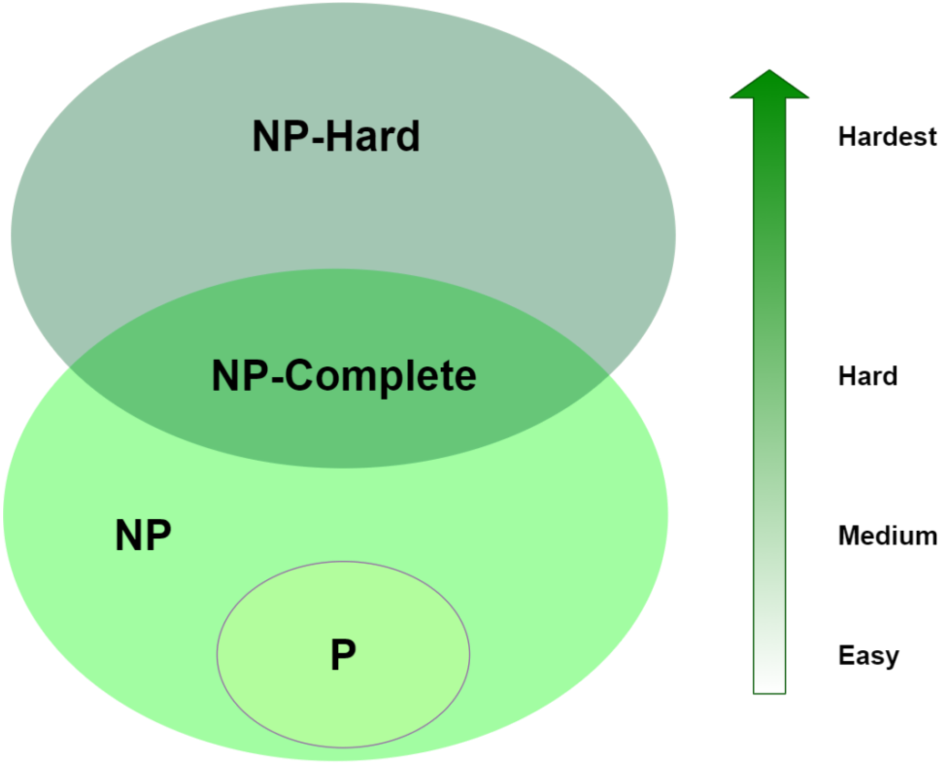
P (Polynomial) Time Problems

- Programmable Function (or method) is polynomial-time
  - if completes in constant-time or polynomial time,
  - then the entire algorithm takes polynomial time.
- Polynomial-time algorithms:
  - Minimum Spanning Tree: Kruskal's $O(E \lg V)$ and Prim's $O(E + V \lg V)$ algorithm
  - Shortest Path Algorithms: Djikstra's $O(E \lg V)$ and Bellman-Ford's $O(EV)$ algorithm

https://en.wikipedia.org/wiki/P_(complexity)
https://en.wikipedia.org/wiki/P-complete

NP - Naming convention

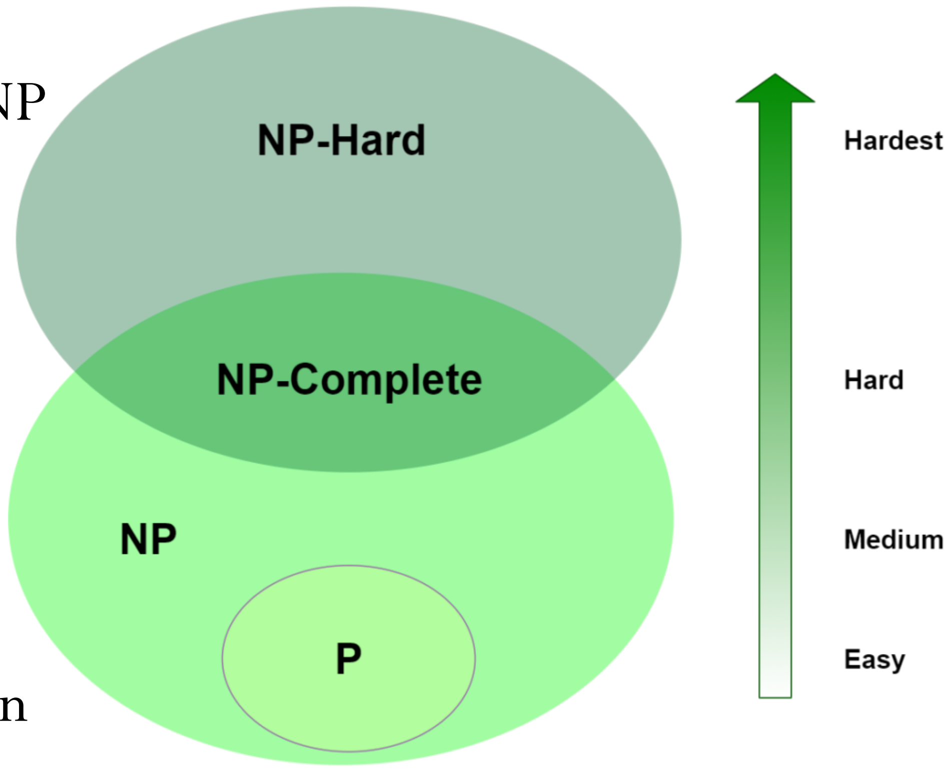
- Classification
  - Hardest → NP-Hard
  - Hard → NP-Complete
  - Medium → NP
  - Easy →P
- Order of N inputs
  - $O(1)$ – constant-time
  - $O(\log_2(n))$ – logarithmic-time
  - $O(n)$ – linear-time
  - $O(n^2)$ – quadratic-time
  - $O(n^k)$ – polynomial-time
  - $O(k^n)$ – exponential-time
  - $O(n!)$ – factorial-time



https://www.baeldung.com/cs/p-np-np-complete-np-hard

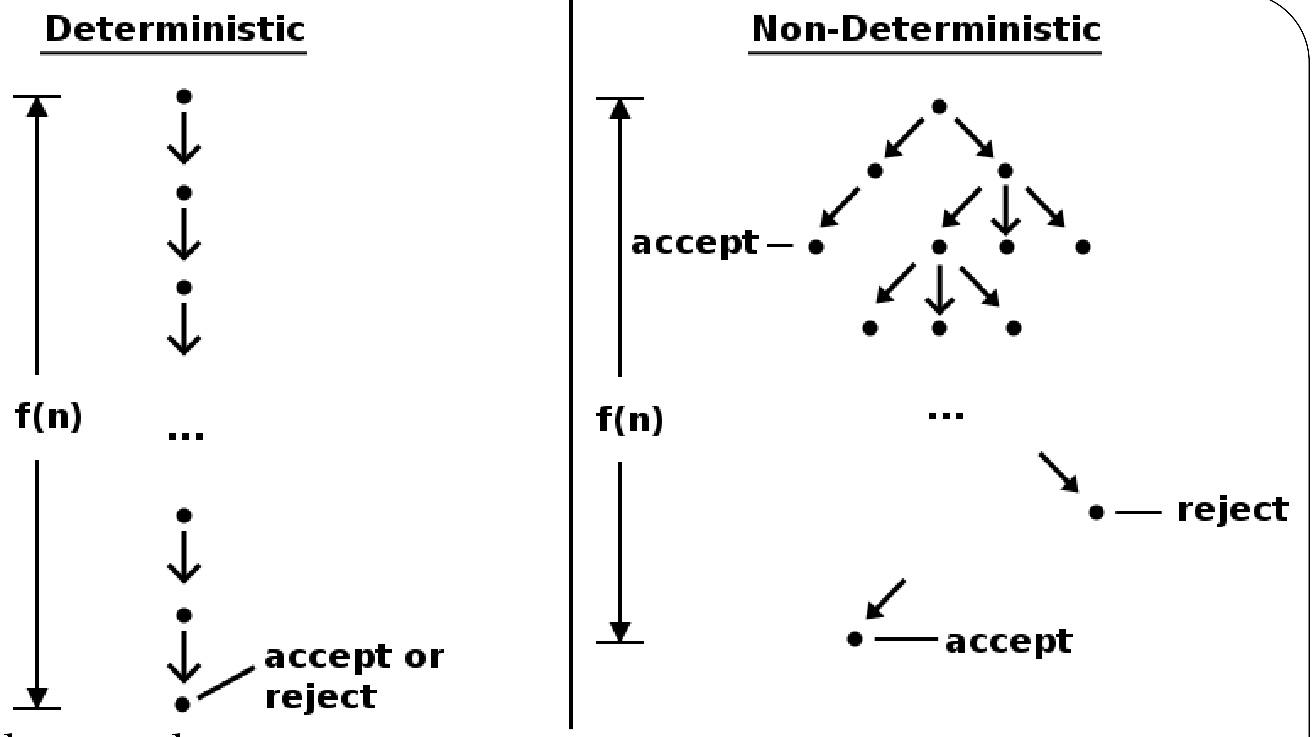
NP - Naming convention

- NP-hard: Class of problems are at least as hard as the hardest problems in NP.
- NP-hard problems do not have to be in NP; means NP hard problem may not even be decidable.
- NP-complete: Class of decision problems which contains the hardest problems in NP. Each NP-complete problem has to be in NP.
- NP: Class of computational decision problems for which any given yes-solution can be verified as a solution in polynomial time
- NP-easy: At most as hard as NP, but not necessarily in NP.



https://en.wikipedia.org/wiki/NP-hardness
https://www.baeldung.com/cs/p-np-np-complete-np-hard

P and NP Problems



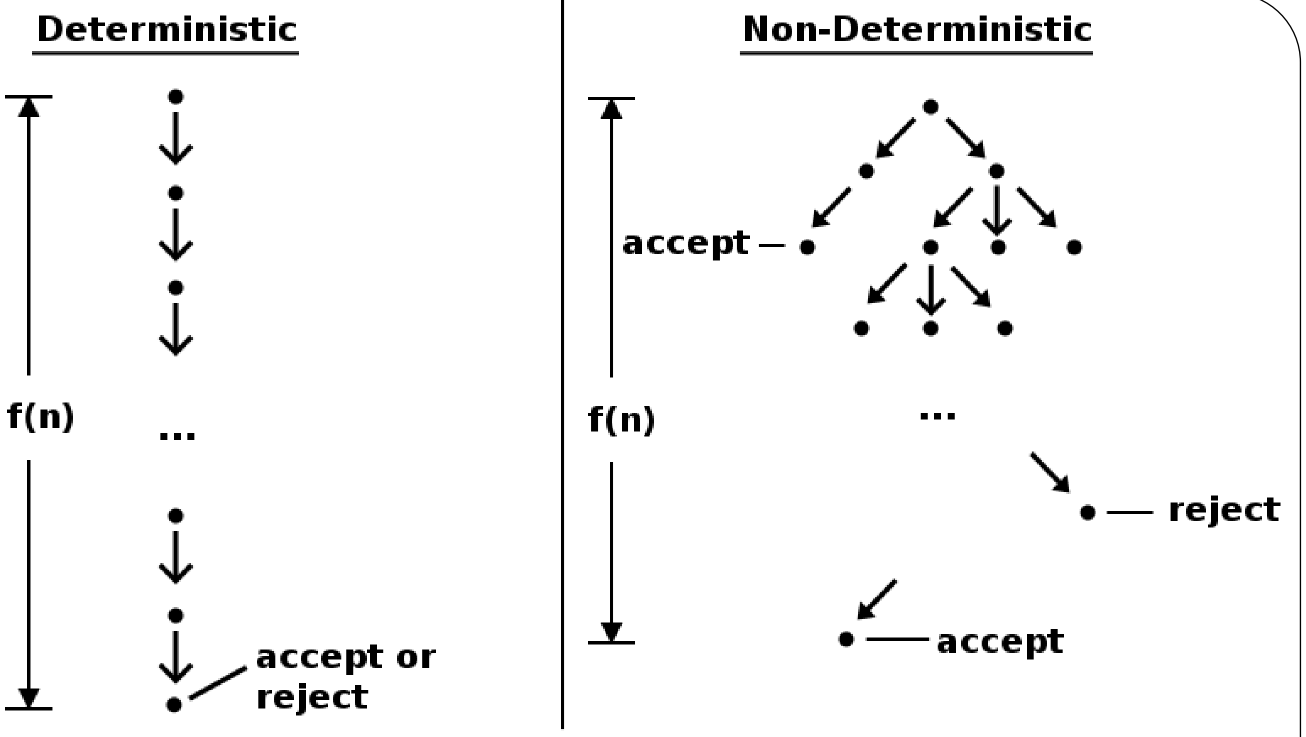
- Nondeterministic Polynomial-time
- "Nondeterministic" refers to
  - "luckiest possible guesser"
- "Complete" refers to
  - "in the same complexity class"
- **P** versus **NP** determine
  - whether a problem can be verified in polynomial time
  - whether the problem can also be solved in polynomial time.
- If it turned out that **P** ≠ **NP**, (widely accepted/believed),
- There are problems in **NP** that are harder to compute than to verify:
- NP problems could not be solved in polynomial time, but the answer could be verified in polynomial time.

https://en.wikipedia.org/wiki/P_versus_NP_problem

NP Complete

- Nondeterministic Polynomial-time

Complete

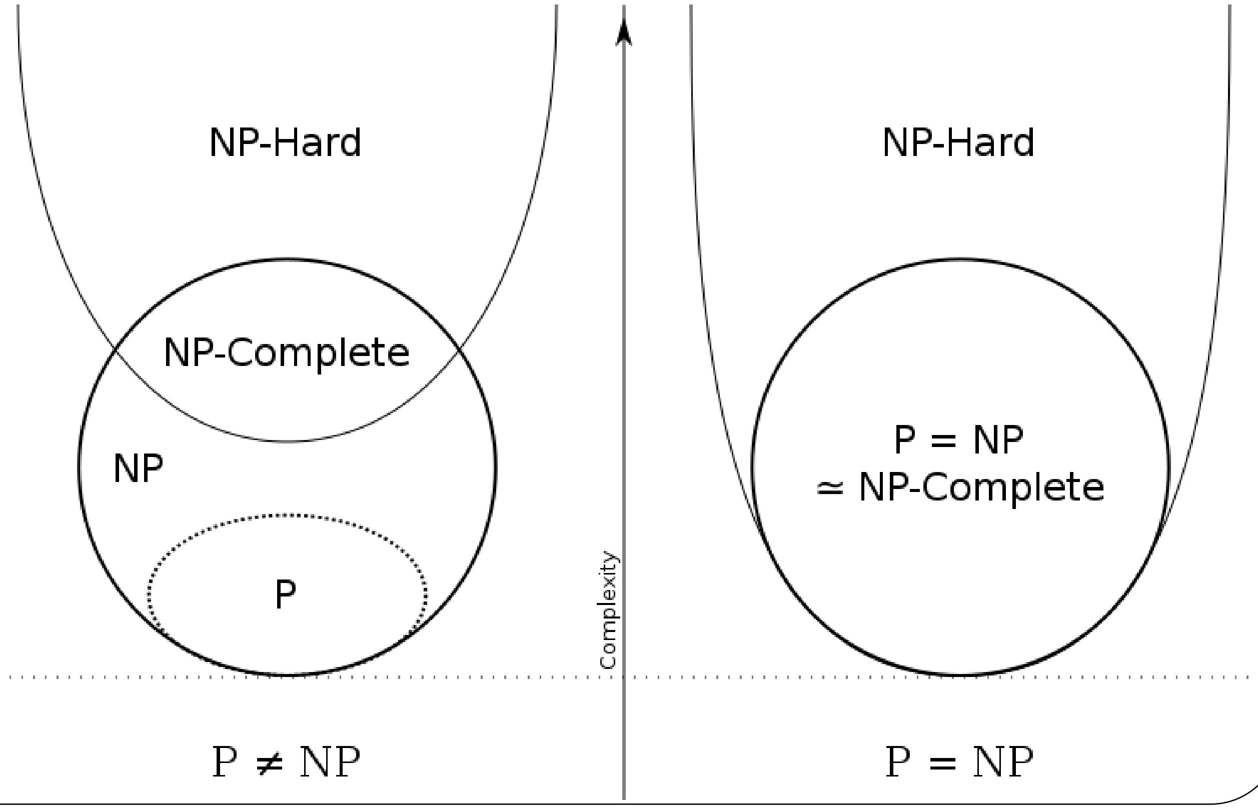


- A problem is NP-complete when:
  - a brute-force search algorithm can solve it, and the correctness of each solution can be verified quickly, and
  - the problem can be used to simulate any other problem with similar solvability.
- NP-complete problem can be *verified* "quickly",
- There is no known way to *find* a solution quickly.

https://en.wikipedia.org/wiki/NP-completeness

NP - Hard Problems

- Non-Deterministic Polynomial-time hardness
- At least as hard as the hardest problems in NP
- The might be some polynomial-time algorithms for NP-hard problems but might not been discovered yet
- NP-hard but not NP-complete
  - halting problem: "given a program and its input, will it run forever?"
  - traveling salesman problem

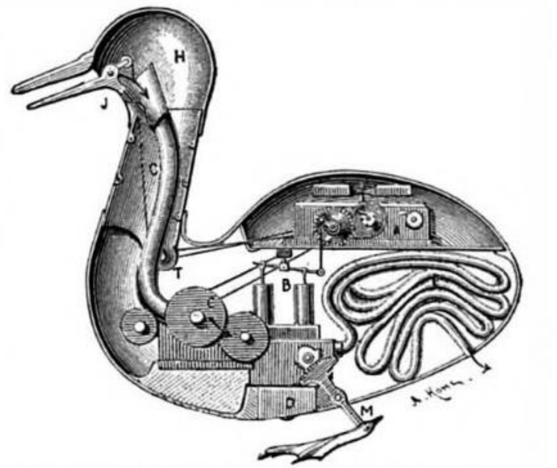


https://en.wikipedia.org/wiki/NP-hardness

Reductionism in Algorithms

Reductionism

- Reductionism is old domain since 16th century
- explains system in terms of parts and their interactions
  - Define a domain of possible parts
  - Generate inputs over the interaction between parts
  - Perform a deterministic computation on the input data
  - Aggregate the results
- interprets a complex system as the sum of its parts



General Problems, Input Size and Time Complexity

- Time complexity of algorithms :
  - polynomial time algorithm ("efficient algorithm") v.s.
  - exponential time algorithm ("inefficient algorithm")
  - Find the shortest path in a graph from X to Y. (easy) polynomial time.
  - Find the longest path in a graph from X to Y. (with no cycles) (hard) exponential time
  - Is there a simple path from X to Y with weight <= M? (easy) polynomial time.
  - Is there a simple path from X to Y with weight >= M? (hard) exponential time

f(n)	10	30	50
n	0.00001 sec	0.00003 sec	0.00005 sec
n^5	0.1 sec	24.3 sec	5.2 mins
2^n	0.001 sec	17.9 mins	35.7 yrs

Decision problems

- The solution to the problem is "yes" or "no". Most optimization problems can be phrased as decision problems (still have the same time complexity).
- Given a decision problem X.
  If there is a polynomial time Non-deterministic Turing machine (or computer) program that solves X, then X belongs to NP

Given a decision problem X.

For every instance I of X,

(a) guess solution S for I, and

(b) check "is S a solution to I?"

If (a) and (b) can be done in polynomial time, then X belongs to NP.

Optimization problems

- We can repeatedly run algorithm X for various profits(P values) to find an optimal solution.
- Example : Use binary search to get the optimal profit, maximum of lg $\sum p_i$ runs, (where M is the capacity of the knapsack optimization problem)

Min Bound Optimal Profit Max Bound

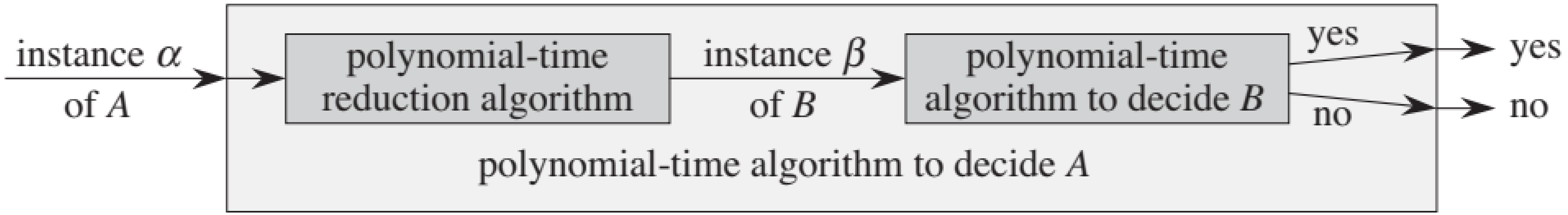
0 $\sum p_i$

|____________________|____________________|

Search for the optimal solution

Polynomial-time Reduction Algorithm

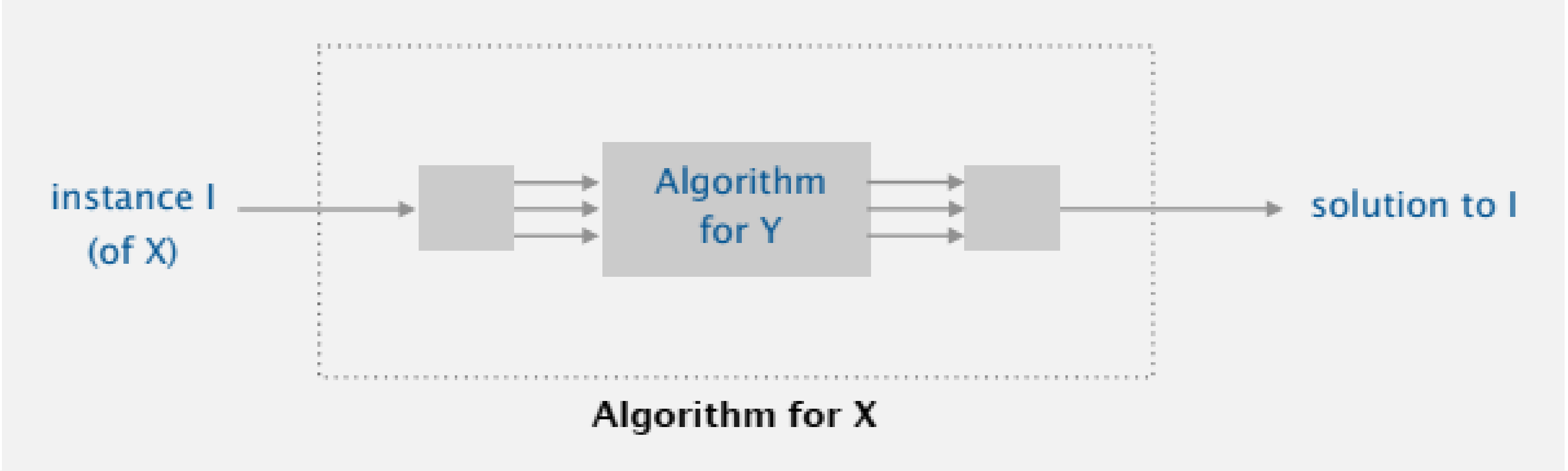
- Input to a particular problem an ***instance*** of that problem
- Suppose that we have a procedure that transforms any instance α of A into some instance β of B
- Given an instance α of problem A, use a polynomial-time reduction algorithm to transform it to an instance β of problem B.
- Run the polynomial-time decision algorithm for B on the instance β
- Use the answer for β as the answer for α.



Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (Vol. 3, pp. 624-642). Cambridge: MIT press.

Algorithm Reduction

- Def. Problem X reduces to problem Y if you can use an algorithm that solves Y to help solve X.



Cost of solving X = total cost of solving Y + cost of reduction.

perhaps many calls to Y on problems of different sizes (though, typically only one call)

preprocessing and postprocessing (typically less than cost of solving Y)

Algorithm Reduction

Ex 1. [finding the median reduces to sorting]

To find the median of N items:

- Sort N items.
- Return item in the middle.

cost of sorting

cost of reduction

Cost of solving finding the median. $\underbrace{N \log N}_{\text{cost of sorting}} + \underbrace{1}_{\text{cost of reduction}}$.

Ex 2. [element distinctness reduces to sorting]

To solve element distinctness on N items:

- Sort N items.
- Check adjacent pairs for equality.

cost of sorting

cost of reduction

Cost of solving element distinctness. $N \log N + N$.

Algorithm Reduction

- Design algorithm. Given algorithm for Y, can also solve X.
  - CPM reduces to topological sort.
  - Arbitrage reduces to negative cycles.
  - Bipartite matching reduces to maxflow.
  - Seam carving reduces to shortest paths in a DAG.
  - Burrows-Wheeler transform reduces to suffix sort

Since I know how to solve Y, can I use that algorithm to solve X?

programmer's version: I have code for Y. Can I use it for X?

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. Turing.

[Received 28 May, 1936.—Read 12 November, 1936.]

ROBERT SEDGEWICK | KEVIN WAYNE. *Algorithms* 4th Edition, https://algs4.cs.princeton.edu/home/

Algorithm Reduction – Language Transformation

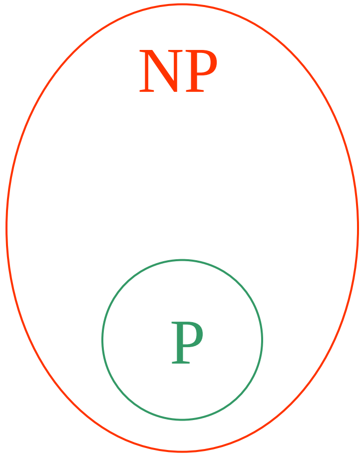
- The class P and Deterministic Turing Machine
  - Given a decision problem X, if there is a polynomial time Deterministic Turing Machine (or computer) program that solves X, then X is belong to P
  - ***Informally, there is a polynomial time algorithm to solve the problem***
  - Polynomial Transformation (" $\propto$ ")
  - **L1 $\propto$ L2** : There is a polynomial time transformation that transforms arbitrary instance of L1 to some instance of L2.
  - **If L1 $\propto$ L2 then L2 is in P implies L1 is in P**
    (or L1 is not in P implies L2 is not in P)
  - **If L1 $\propto$ L2 and L2 $\propto$ L3 then L1 $\propto$ L3**

NP-Completeness and Cooks Theorem

P and NP

- Obvious : P $\subseteq$ NP, i.e. A (decision) problem in P does not need "guess solution". The correct solution can be computed in polynomial time.
- One of the most important open problem in theoretical compute science :

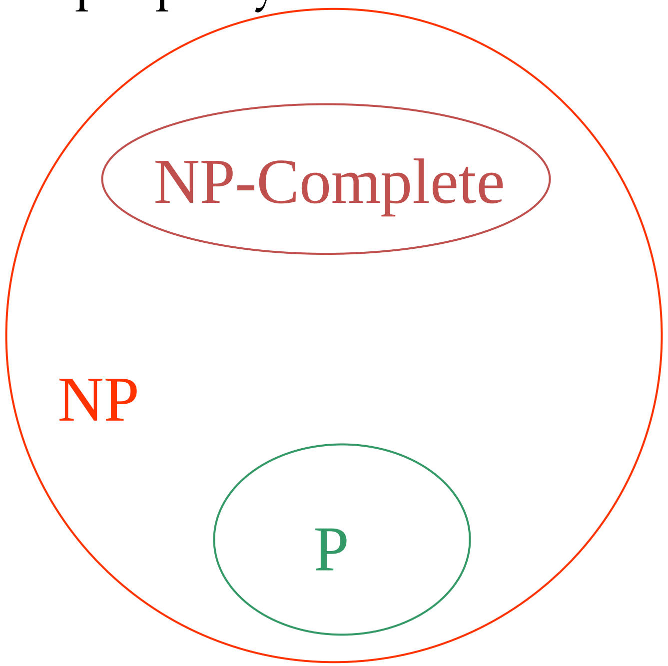
  Is P=NP ?
- Most likely "No".
- Currently, there are many known (decision) problems in NP, and there is no solution to show anyone of them in P.



P, NP, and NP-Complete

- P: (Decision) problems solvable by deterministic algorithms in polynomial time
- NP: (Decision) problems solved by non-deterministic algorithms in polynomial time
- A group of (decision) problems, including all of the ones we have discussed (Satisfiability, 0/1 Knapsack, Longest Path, Partition) have an additional important property:

  If any of them can be solved in polynomial time, then they all can!
- These problems are called NP-complete problems.



Cooks Theorem

- Stephen Cook introduced the notion of NP-Complete Problems.
  - This makes the problem "P = NP ?" much more interesting to study.
  - L is **NP-Complete** *if* $L \in NP$ *then for all other* $L' \in NP, L' \propto L$
  - L is **NP-Hard** if *for all other* $L' \in NP, L' \propto L$
  - If an NP-complete problem can be solved in polynomial time then all problems in NP can be solved in polynomial time.
  - If a problem in NP cannot be solved in polynomial time then all problems in NP-complete cannot be solved in polynomial time.
  - Note that an NP-complete problem is one of those hardest problems in NP.
- Lemma :
  If L1 and L2 belong to NP,
  L1 is NP-complete, and L1 $\propto$ L2
  then L2 is NP-complete.

  i.e. $L1, L2 \in NP$ *and for all other* $L' \in NP, L' \propto L1$ *and* $L1 \propto L2 \rightarrow L' \propto L2$

Cooks Theorem and Satisfiability

- SATISFIABILITY is NP-Complete. (The first NP-Complete problem)
  - Instance : Given a set of variables, U, and a collection of clauses, C, over U.
  - Question : Is there a truth assignment for U that satisfies all clauses in C?
- CNF-Satisfiability (CNF – Conjunctive Normal Form)
  - because the expression is in (the product of sums).
- Example:

"$\neg x_i$" = "not x_i"   "OR" = "logical or"   "AND" = "logical and" $U = \{x_1, x_2\}$

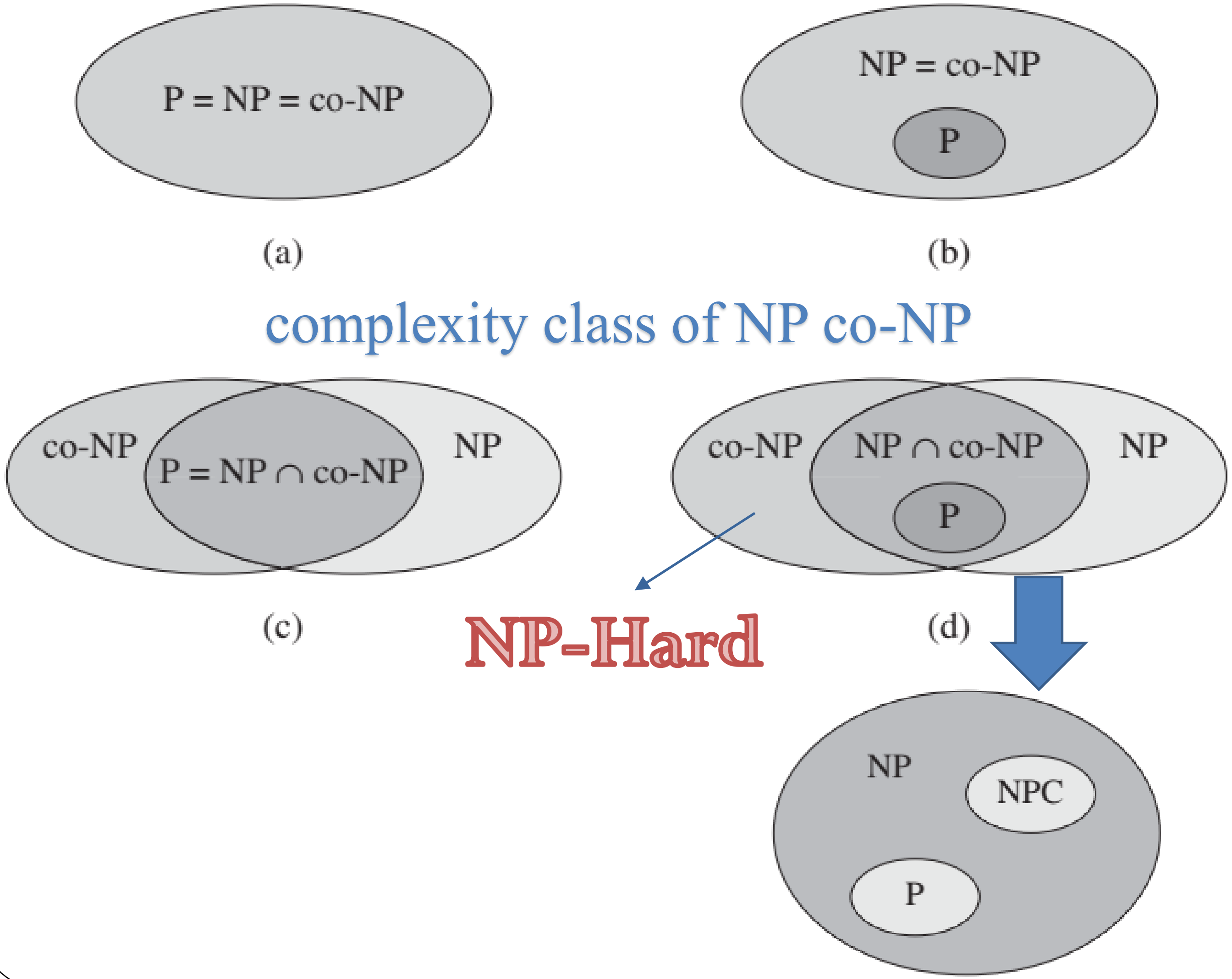
$$C_1 = \{(x_1, \neg x_2), (\neg x_1, x_2)\}$$
$$= (x_1 \text{ OR } \neg x_2) \text{ AND } (\neg x_1 \text{ OR } x_2)$$
$$\text{if } x_1 = x_2 = \text{True} \rightarrow C_1 = \text{True}$$

$$C_2 = (x_1, x_2)\ (x_1, \neg x_2)\ (\neg x_1) \rightarrow \text{not satisfiable}$$

Polynomial-Time Verifier

- NP = decision problems for which there exists a polynomial-time verifier
- Algorithm A with two inputs
  - input to the problem: x
  - certificate: y
- A is polynomial-time verifier: for any x there exists certificate y such that
- A(x, y) outputs
  - "yes" iff x is "yes"- instance, and A runs in polynomial time for such instances.
  - "No" - instances do not have to be verifiable in polynomial time.

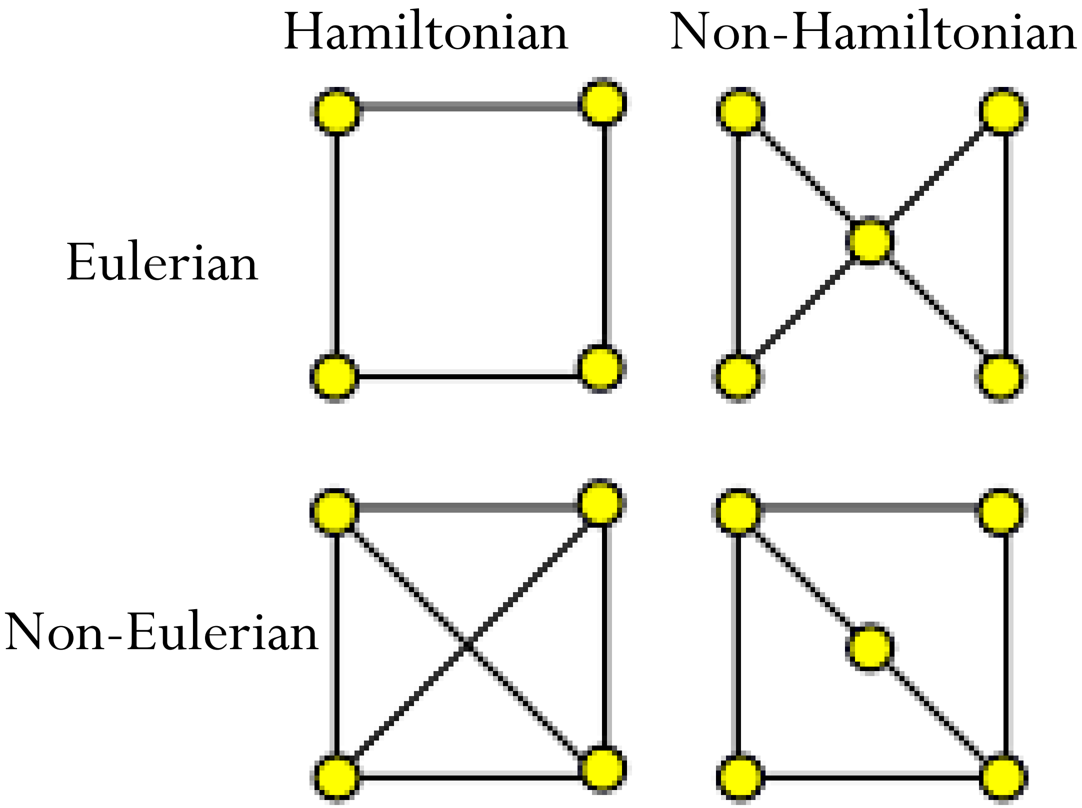
Verification algorithm



(a) Most researchers regard this possibility as the most unlikely.
(b) but it need not be the case that P = NP.
(c) but NP is not closed under complement.
(d) Most researchers regard this possibility as the most likely.

Euler tour is in P

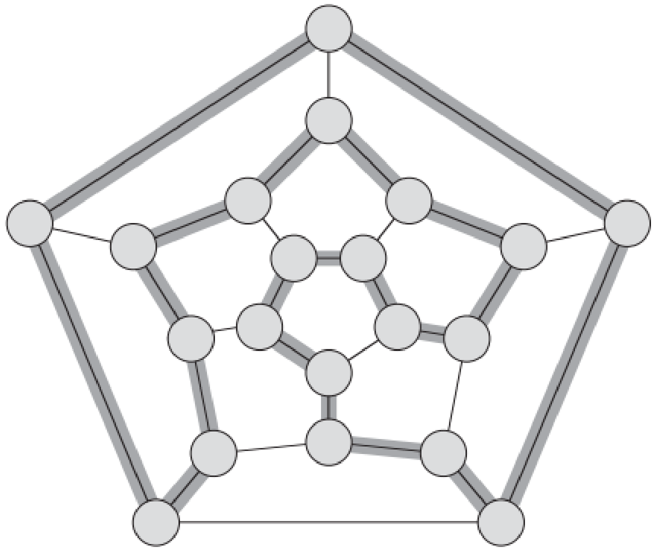
- **Euler tour:** An ***Euler tour*** of a connected, directed gaph is a cycle that traverses each *edge* of G exactly once, although it is allowed to visit each vertex more than once.
- We can determine whether a graph has an Euler tour in only O(E) time and, in fact, we can find the edges of the Euler tour in O(E) time.

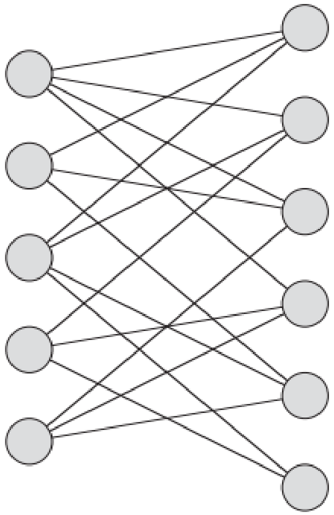


Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (Vol. 3, pp. 624-642). Cambridge: MIT press.

Hamiltonian-cycle Problem is NP-Complete

- "Does a graph G have a Hamiltonian cycle?" **NP-Complete**
- A graph representing the vertices, edges, and faces of a dodecahedron, with a Hamiltonian cycle shown by shaded edges.
- A bipartite graph with an odd number of vertices. Any such graph is non-Hamiltonian.





Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (Vol. 3, pp. 624-642). Cambridge: MIT press.

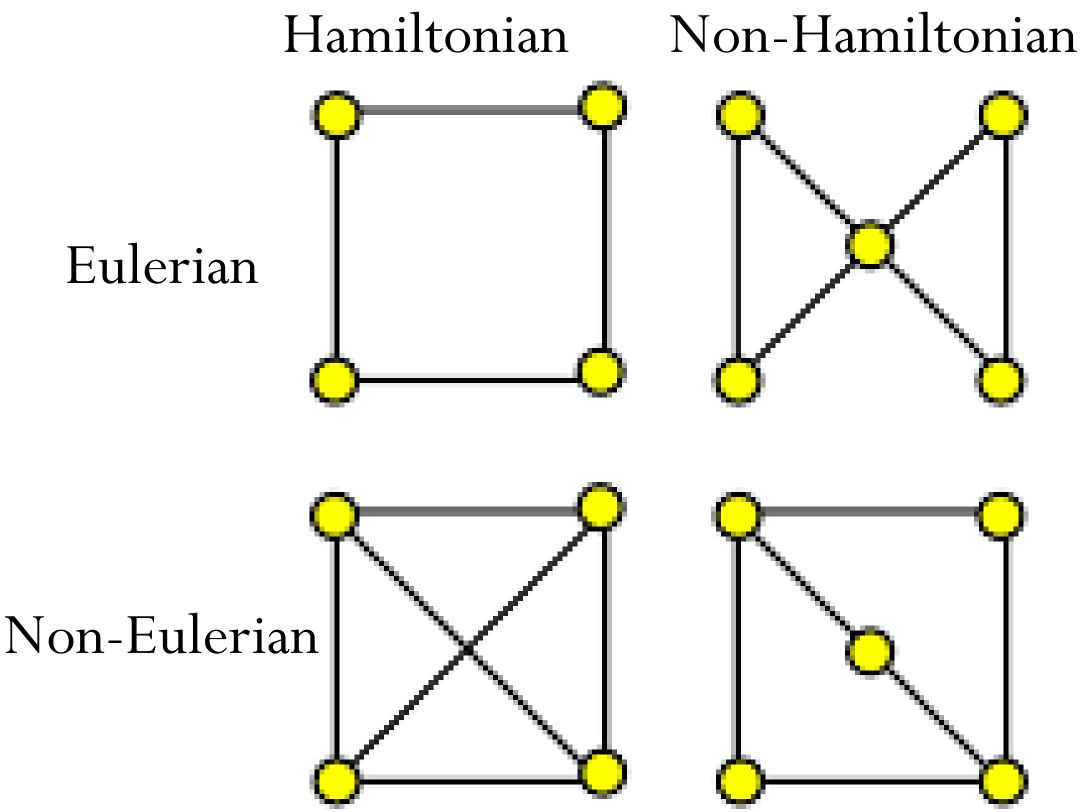
Hamiltonian-cycle Problem is NP-Complete

- "Does a graph G have a Hamiltonian cycle?"

HAM-CYCLE

= { <G> : G is a Hamiltonian graph}

- A Hamiltonian cycle of a directed graph is a simple cycle that contains each vertex in V
- Determining whether a directed graph has a Hamiltonian cycle is NP-complete.
- Determining whether an undirected graph has a Hamiltonian cycle is NP-complete.



Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (Vol. 3, pp. 624-642). Cambridge: MIT press.

NP-Complete and NP-Hard Problems

P, NP, and NP-Hard

- P = class of all problems for which we can compute a solution in polynomial time
- P = problems solvable in polynomial time on Turing machine (or computer)
- NP = class of problems for which we can verify a given solution in polynomial time
- NP = problems solvable in polynomial time on non-deterministic Turing machine (or computer)
- anything that is computable is computable by a Turing machine
  - (*Church-Turing Thesis*)
- "polynomial time" on Turing machine ≅ "polynomial time" on a Computer

NP-Complete and NP-Hard

- $L_1 \leq_P L_2$, then L_1 is not more than a polynomial factor harder than L_2,
- A language $L \subseteq \{0, 1\}^*$ is ***NP-complete*** if

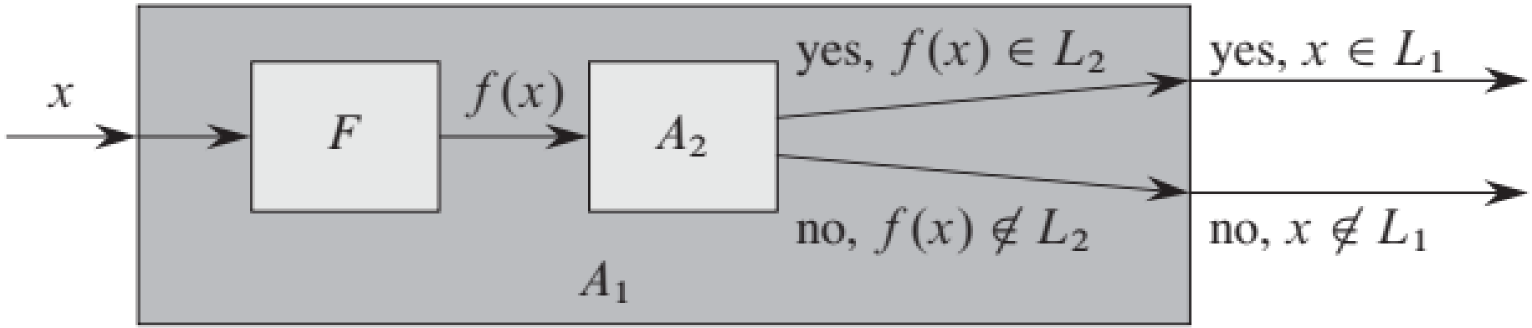
  1. $L \in NP$, and

  2. $L' \leq_P L$ for every $L' \in NP$.
- If a language L satisfies property 2, but not necessarily property 1, we say that L is ***NP-hard***.
- If L is a language such that $L' \leq_P L$ for some $L' \in NPC$, then L is NP-hard. If, in addition, $L \in NP$, then $L \in NPC$.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (Vol. 3, pp. 624-642). Cambridge: MIT press.

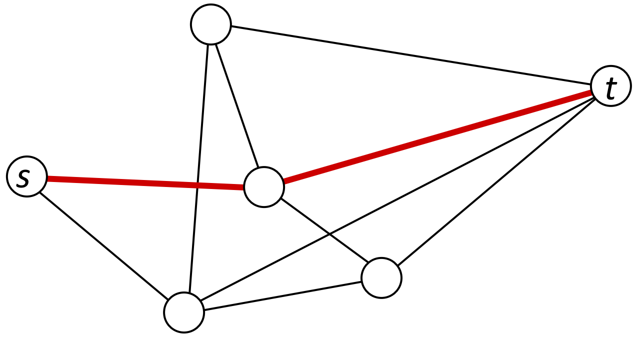
NP-Complete and NP-Hard

- Algorithm F is a reduction algorithm that computes the reduction function f from L_1 to L_2 in polynomial time, and
- A_2 is a polynomial-time algorithm that decides L_2.
- Algorithm A_1 decides whether $x \in L_1$ by using F to transform any input x into f(x) and then using A_2 to decide whether $f(x) \in L_2$.



Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (Vol. 3, pp. 624-642). Cambridge: MIT press.

Two similar graph problems

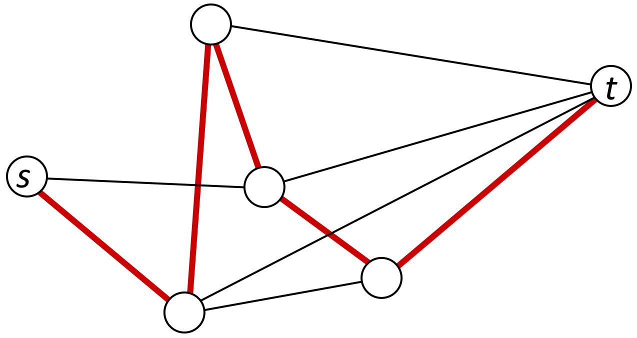


Shortest Path

Input: graph, nodes s and t

Output: simple s-to-t path with minimum number of edges

BFS: $O(\,|V| + |E|\,)$



Longest Path

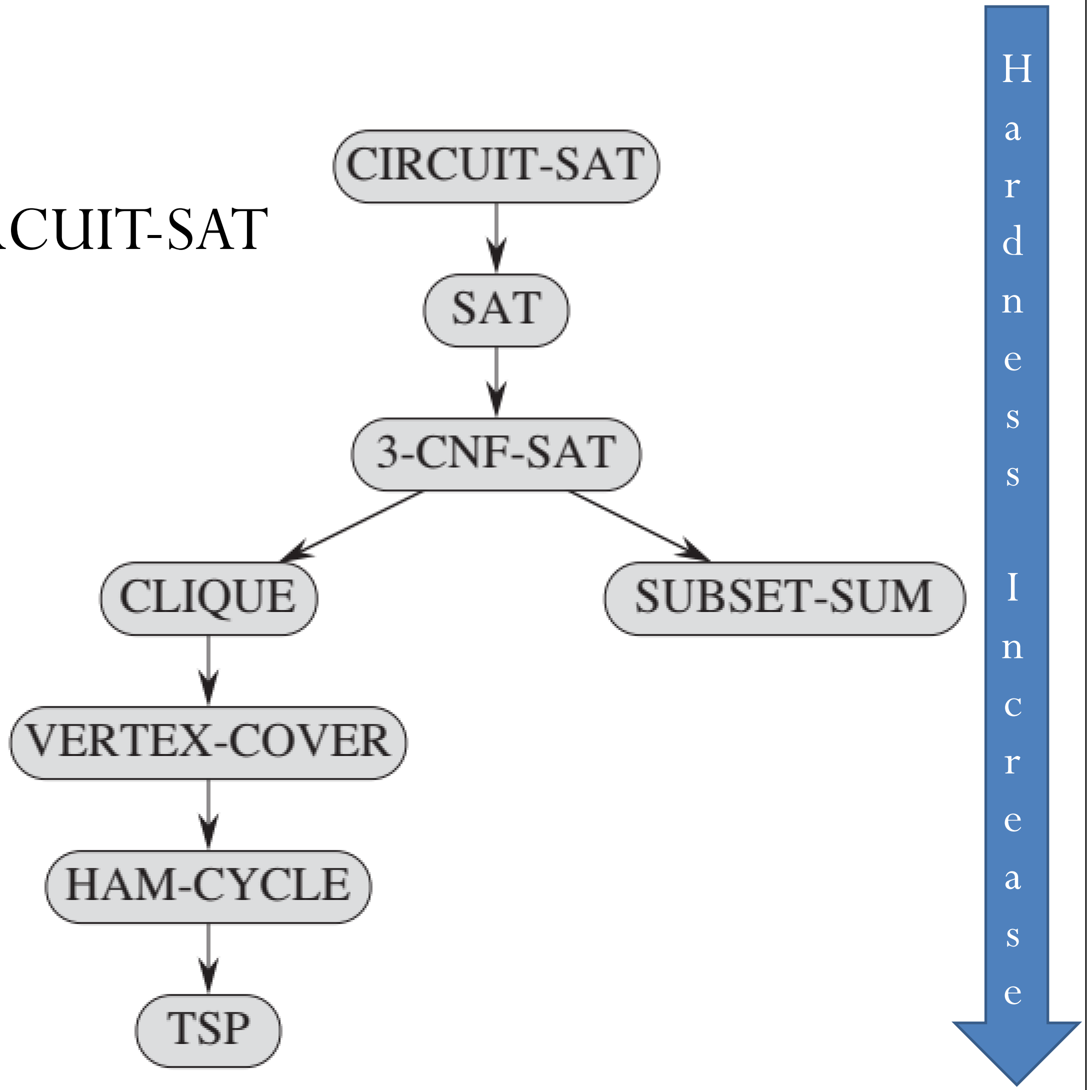
Input: graph, nodes s and t

Output: simple s-to-t path with maximum number of edges

no $\underbrace{\text{polynomial-time}}_{O(n^c) \text{ for some constant } c}$ algorithm known

NP-Complete problems

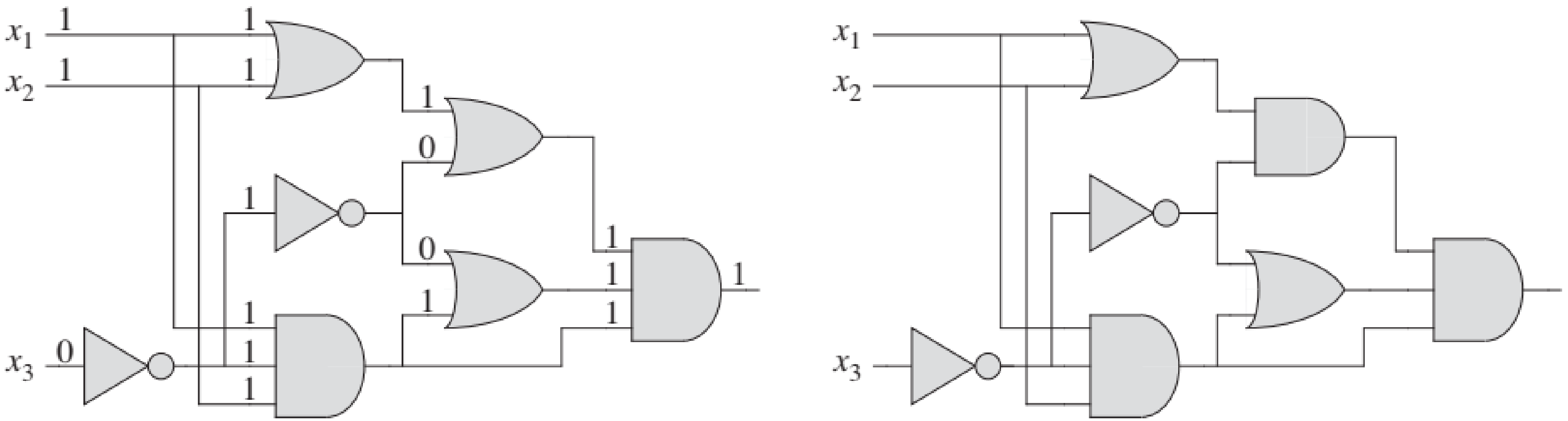
- The structure of NP-completeness
- Reduction from the NP-completeness of CIRCUIT-SAT



Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (Vol. 3, pp. 624-642). Cambridge: MIT press.

Circuit-Satisfiability problem is NP-Complete

- "Given a Boolean combinational circuit composed of AND, OR, and NOT gates, is it satisfiable?" $O(n^k)$
- **(a)** The assignment $<x_1 = 1, x_2 = 1, x_3 = 0>$ to the inputs of this circuit causes the output of the circuit to be 1. The circuit is therefore satisfiable.
- **(b)** No assignment to the inputs of this circuit can cause the output of the circuit to be 1. The circuit is therefore unsatisfiable



Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (Vol. 3, pp. 624-642). Cambridge: MIT press.

Two similar problems on Boolean

$\boxed{(x_1 \lor \neg x_3) \;\land\; (x_2 \lor x_5) \land (x_3 \lor \neg x_4)}$

2-SAT

Input: 2-CNF formula

Output: "yes" if formula can be satisfied, "no" otherwise

O(#clauses)

$\boxed{(x_1 \lor \neg x_2 \lor x_3) \;\land\; (x_2 \lor x_3 \lor \neg x_5) \land (x_1 \lor x_3 \lor x_4)}$

3-SAT

Input: 3-CNF formula

Output: "yes" if formula can be satisfied, "no" otherwise

no polynomial-time algorithm known

Boolean Formula Satisfiability (SAT) is NPC

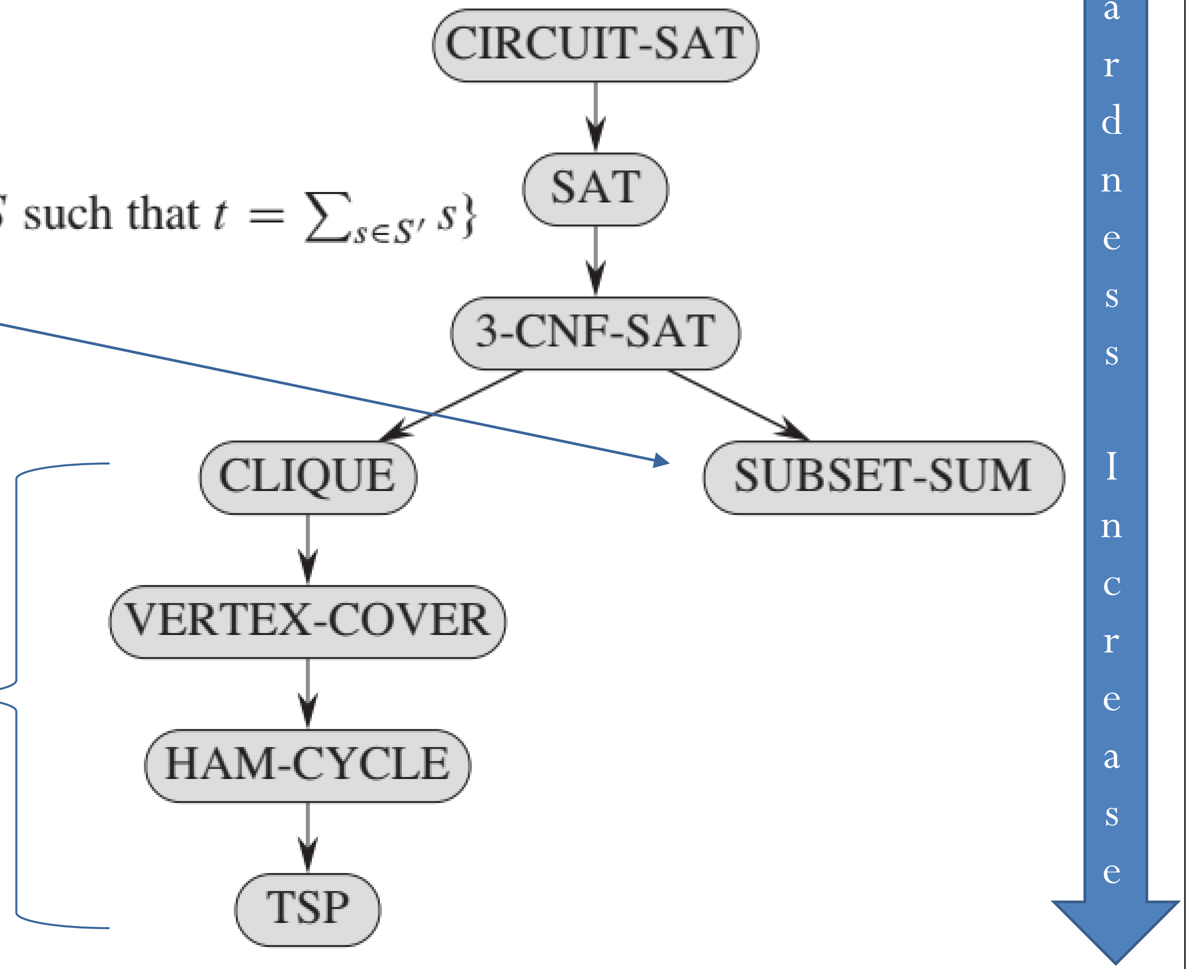
- Satisfiability of boolean formulas is NP-Complete.
  1. n boolean variables: $x_1, x_2, \ldots.\ x_n$
  2. m boolean connectives
  3. parentheses
- $\Omega(2^n)$ asymptotic lower bound is 2^n
- CIRCUIT-SAT $\leq_P$ SAT
- Satisfiability of boolean formulas in 3-conjunctive normal form is NP-Complete.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (Vol. 3, pp. 624-642). Cambridge: MIT press.

Subset-Sum Problem is NP-Complete

3-CNF-SAT $\leq_P$ SUBSET-SUM

SUBSET-SUM $= \{\langle S, t\rangle$: there exists a subset $S' \subseteq S$ such that $t = \sum_{s \in S'} s\}$

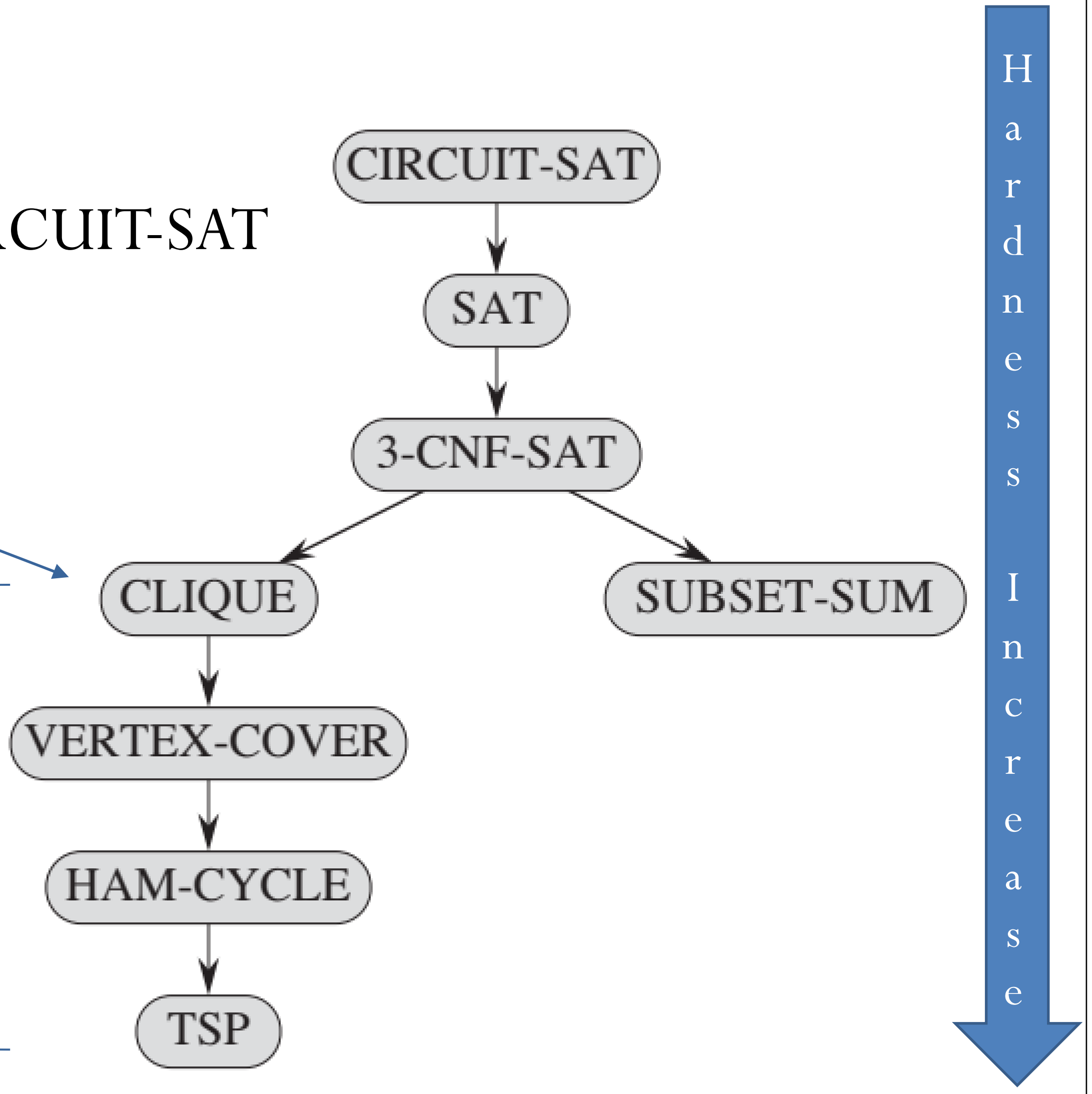


We will only explore Graph Problems!

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (Vol. 3, pp. 624-642). Cambridge: MIT press.

NP-Complete problems

- The structure of NP-completeness
- Reduction from the NP-completeness of CIRCUIT-SAT
- 3-CNF-SAT $\leq_P$ CLIQUE



We will only explore Graph Problems!

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (Vol. 3, pp. 624-642). Cambridge: MIT press.

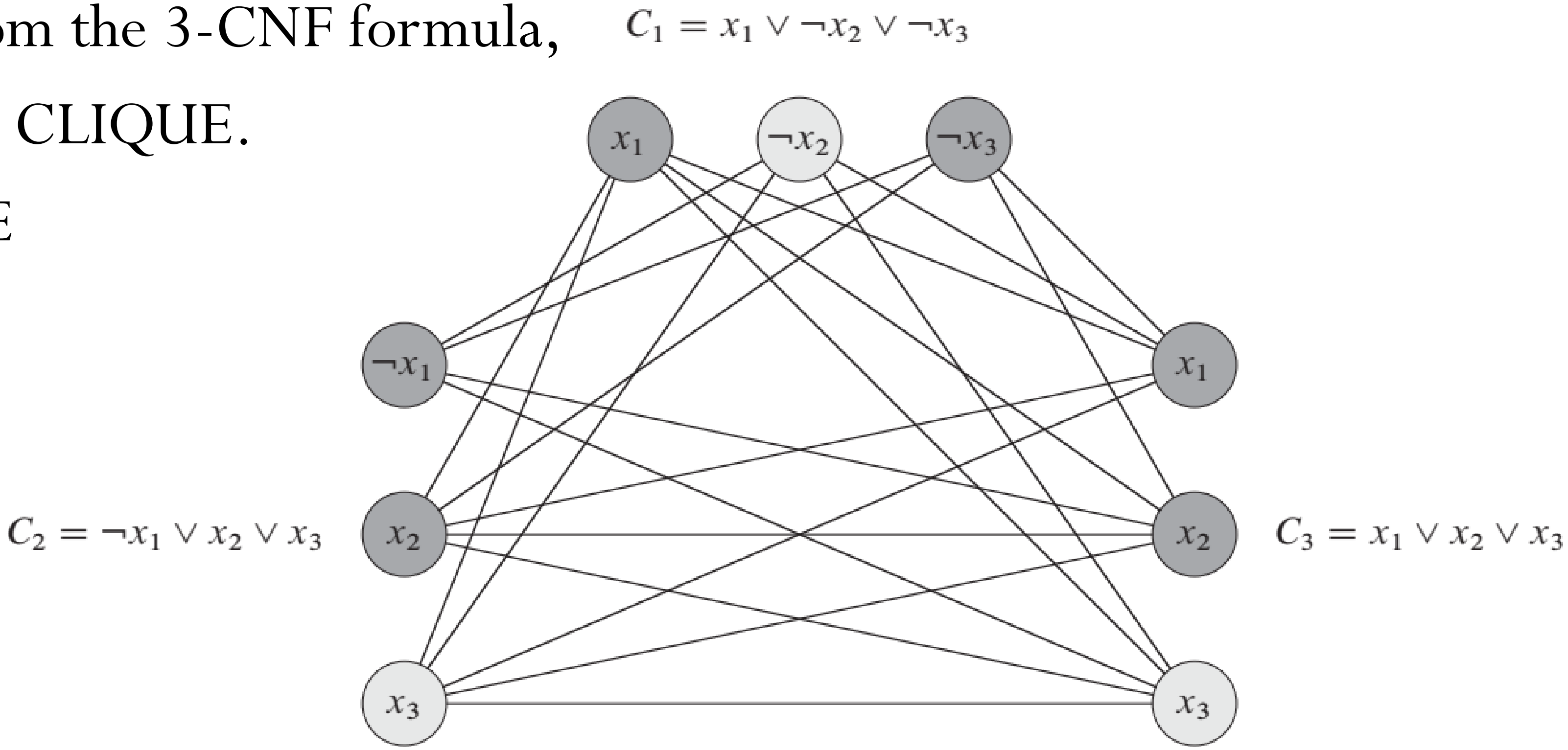
Clique Problem is NP-Complete

- A clique in an undirected graph is a subset $V' \subseteq V$ of vertices, each pair of which is connected by an edge in E.
- In other words, a clique is a complete subgraph of G. The size of a clique is the number of vertices it contains.
- The clique problem is the optimization problem of finding a clique of maximum size in a graph. As a decision problem, we ask simply
- "Whether a clique of a given size k exists in the graph".

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (Vol. 3, pp. 624-642). Cambridge: MIT press.

Clique Problem is NP-Complete

- "Whether a clique of a given size k exists in the graph".
- CLIQUE = {<G, k> : G is a graph containing a clique of size k}
- The graph G derived from the 3-CNF formula,
- reducing 3-CNF-SAT to CLIQUE.
- 3-CNF-SAT $\leq_P$ CLIQUE

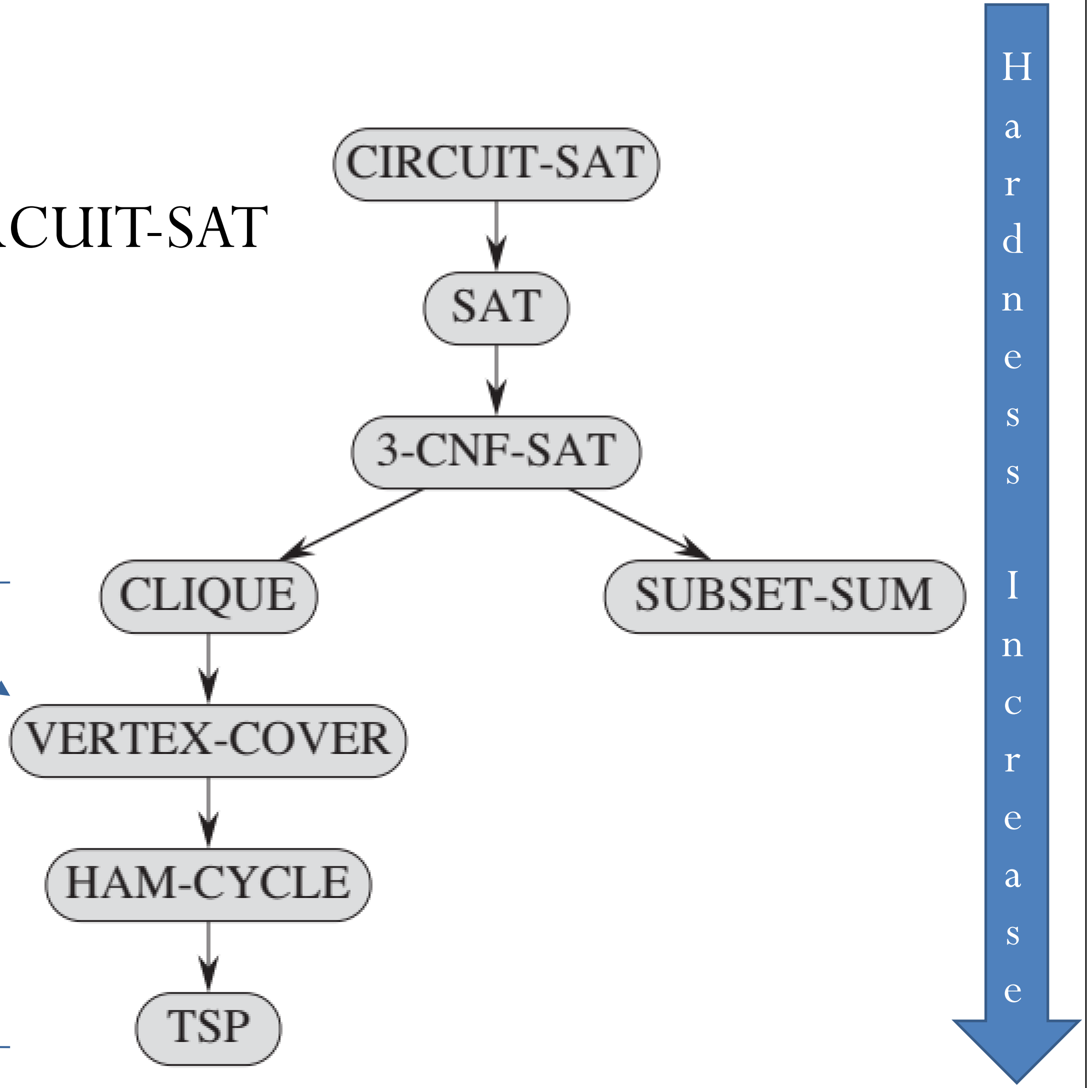


Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (Vol. 3, pp. 624-642). Cambridge: MIT press.

NP-Complete problems

- The structure of NP-completeness
- Reduction from the NP-completeness of CIRCUIT-SAT
- CLIQUE $\le_P$ VERTEX-COVER

We will only explore Graph Problems!



Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (Vol. 3, pp. 624-642). Cambridge: MIT press.

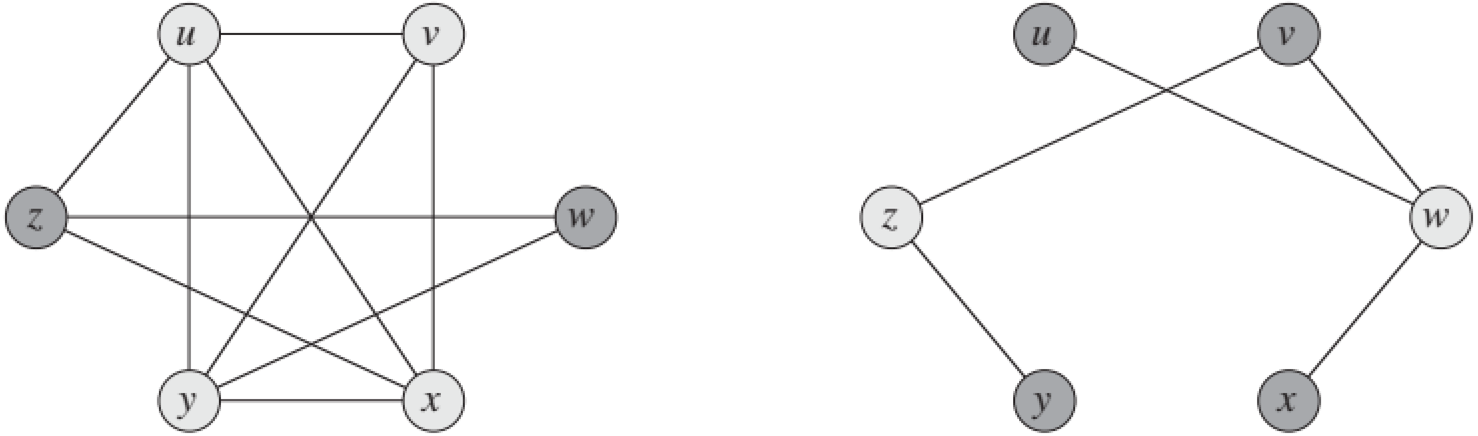
Vertex cover problem is NP-Complete

- A ***vertex cover*** of an undirected graph is a subset $V' \subseteq V$ such that if $(u, v) \in E$, then $u \in V'$ or $v \in V'$ (or both). That is, each vertex "covers" its incident edges, and a vertex cover for G is a set of vertices that covers all the edges in E.
- The ***size*** of a vertex cover is the number of vertices in it.
- The ***vertex-cover problem*** is to find a vertex cover of minimum size in a given graph.
- "Determine whether a graph has a vertex cover of a given size k."
- VERTEX-COVER = {<G, k>: graph G has a vertex cover of size k}.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (Vol. 3, pp. 624-642). Cambridge: MIT press.

Vertex cover problem is NP-Complete

- The ***vertex-cover problem*** is to find a vertex cover of minimum size in a given graph.
- "Determine whether a graph has a vertex cover of a given size k."
- Reducing CLIQUE to VERTEX-COVER
- CLIQUE $\leq_P$ VERTEX-COVER

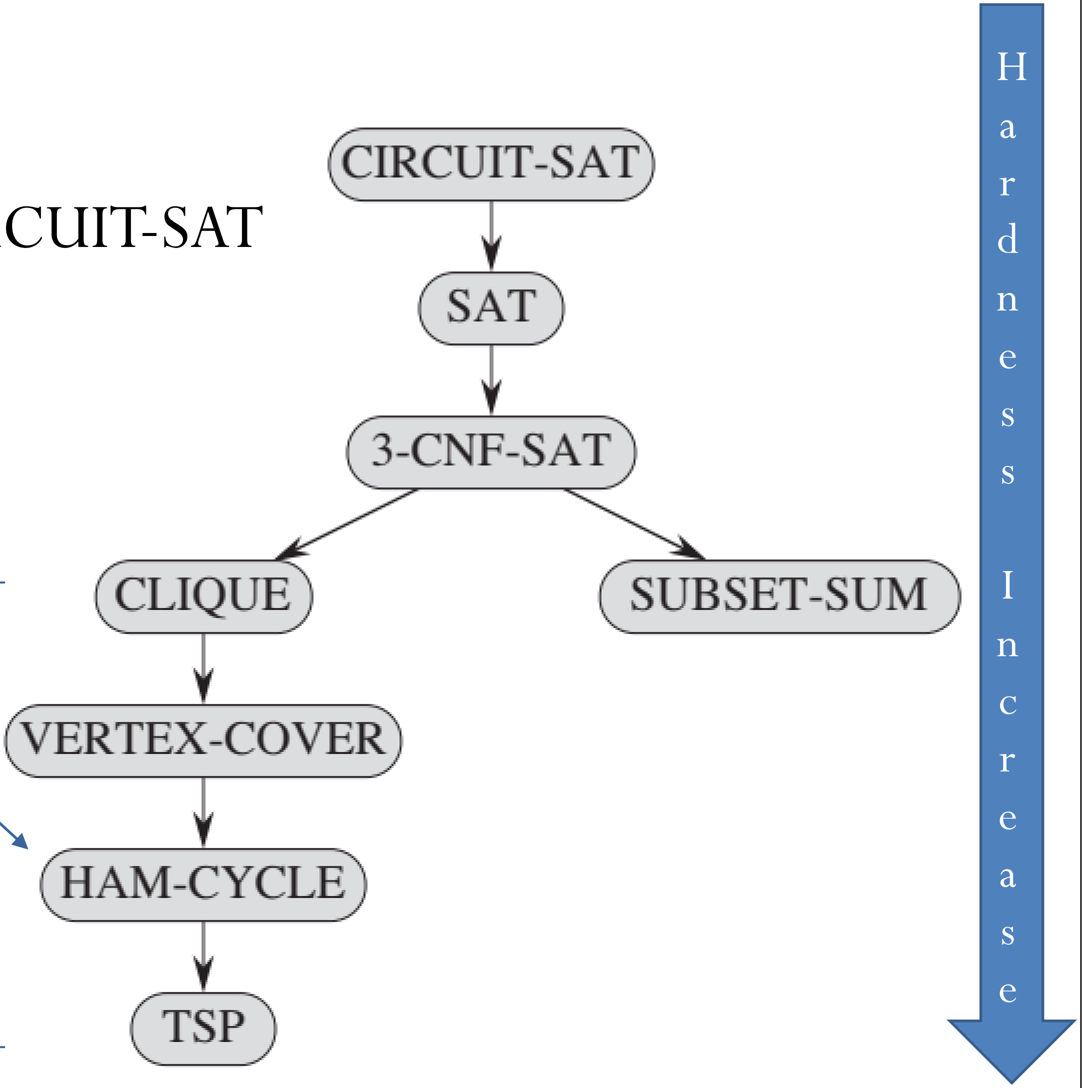


Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (Vol. 3, pp. 624-642). Cambridge: MIT press.

NP-Complete problems

- The structure of NP-completeness
- Reduction from the NP-completeness of CIRCUIT-SAT
- VERTEX-COVER $\leq_P$ HAM-CYCLE

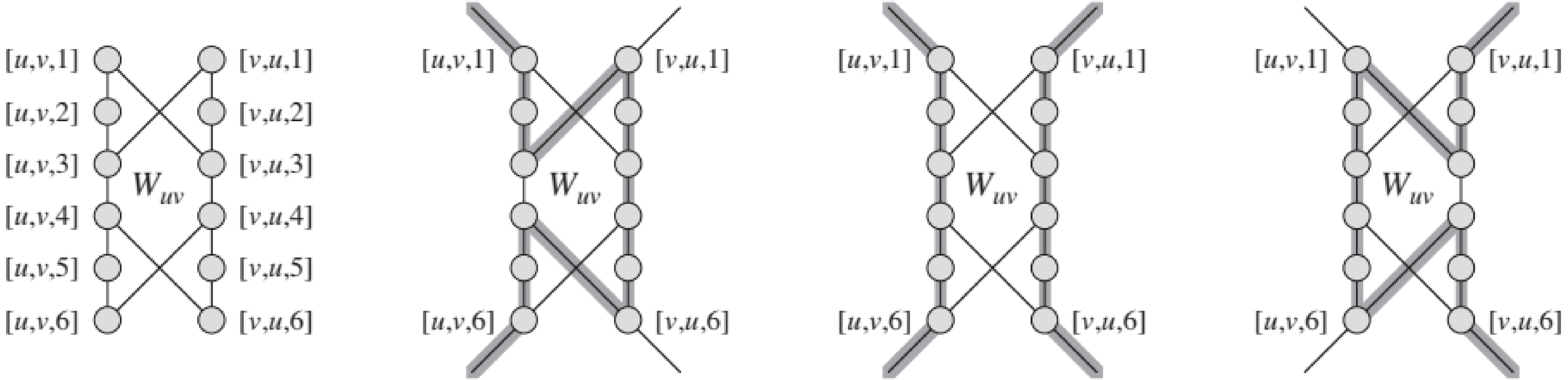
We will only explore Graph Problems!



Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (Vol. 3, pp. 624-642). Cambridge: MIT press.

Hamiltonian cycle problem is NP-Complete

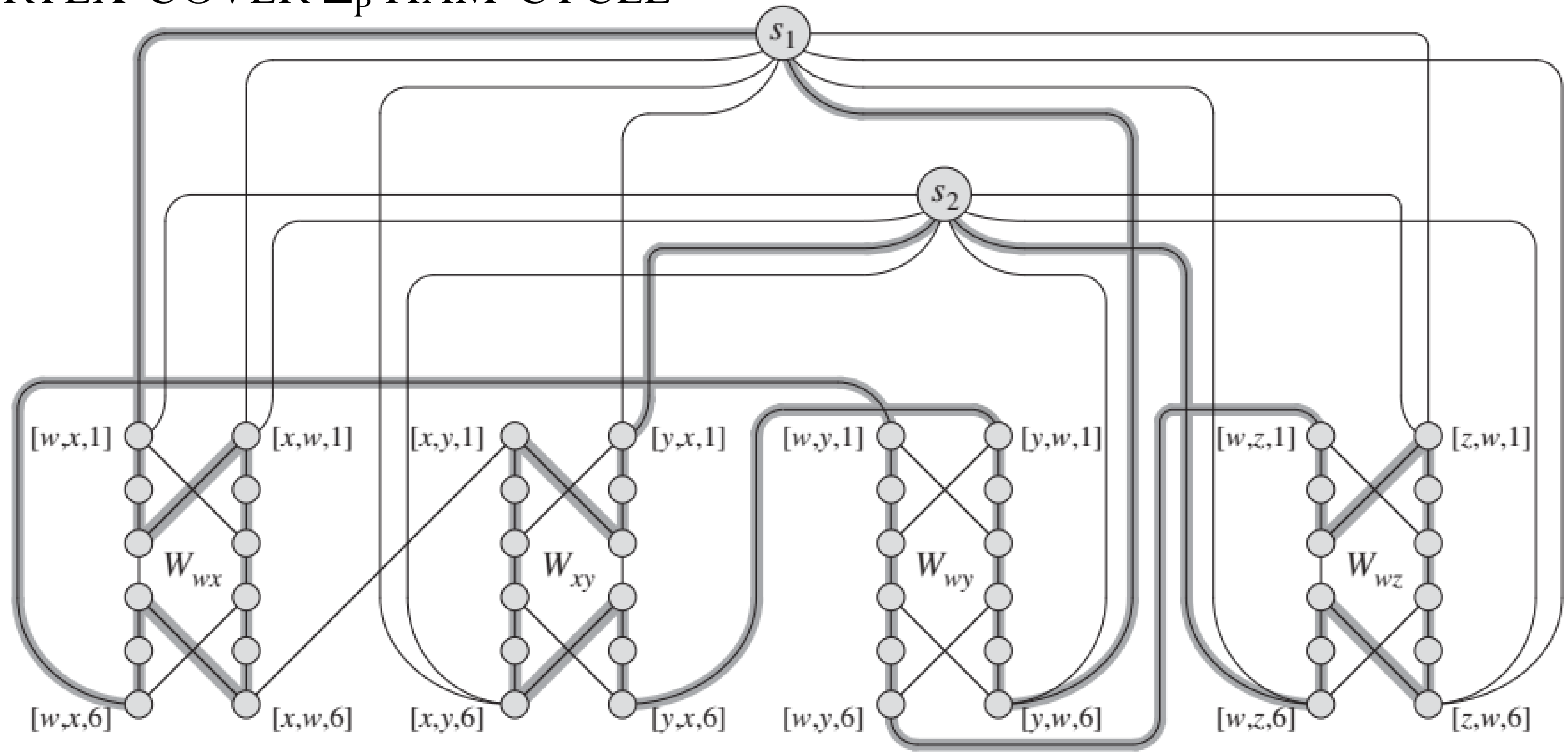
- VERTEX-COVER $\leq_P$ HAM-CYCLE



Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (Vol. 3, pp. 624-642). Cambridge: MIT press.

Hamiltonian cycle problem is NP-Complete

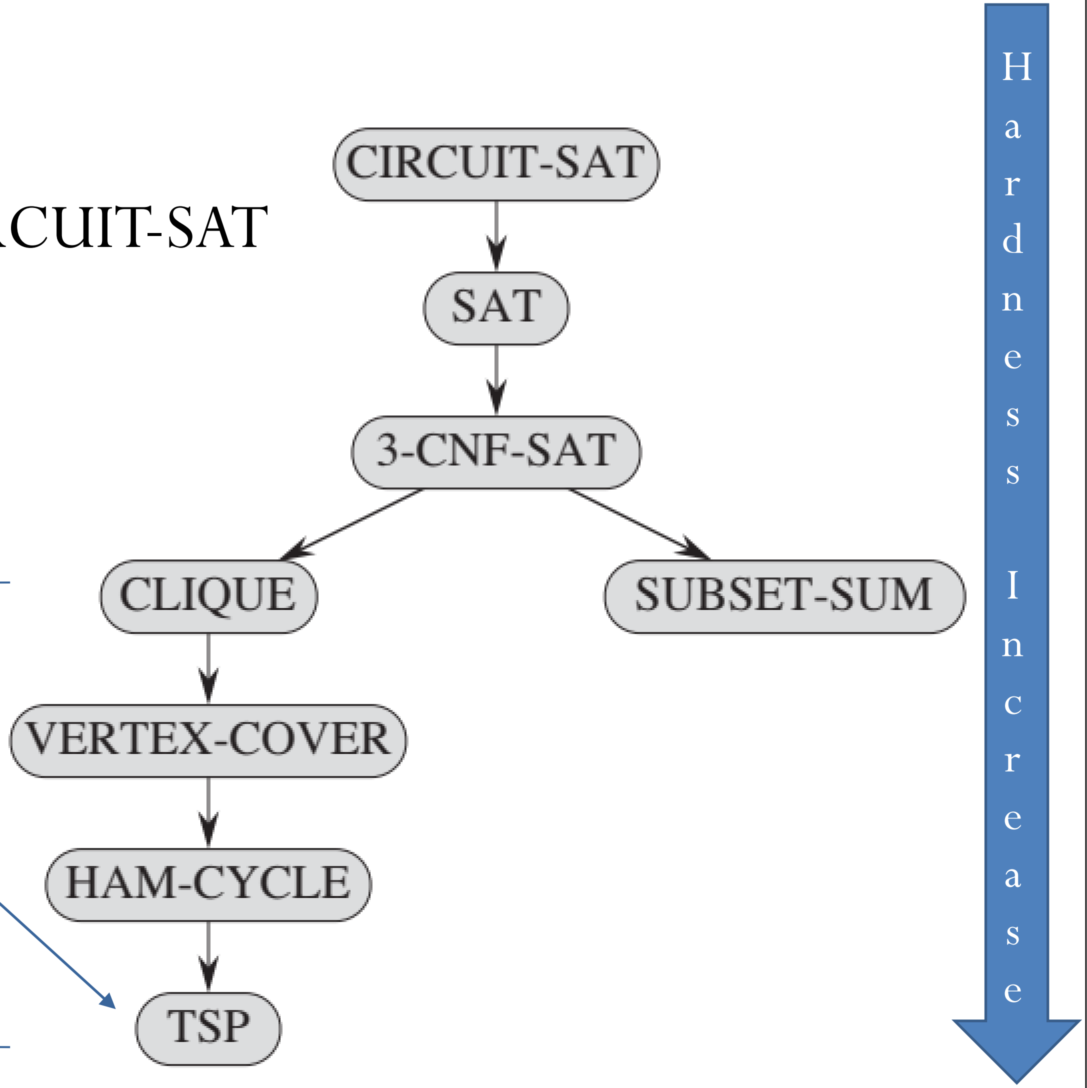
- VERTEX-COVER $\leq_P$ HAM-CYCLE



Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (Vol. 3, pp. 624-642). Cambridge: MIT press.

NP-Complete problems

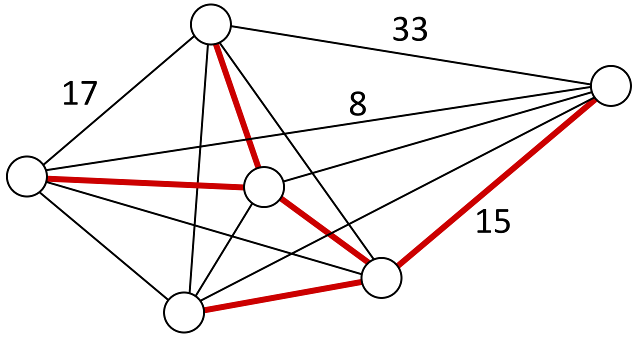
- The structure of NP-completeness
- Reduction from the NP-completeness of CIRCUIT-SAT
- HAM-CYCLE $\leq_P$ TSP

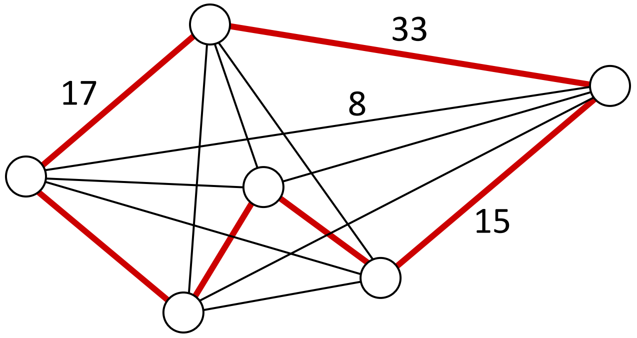


We will only explore Graph Problems!

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (Vol. 3, pp. 624-642). Cambridge: MIT press.

Two similar graph problems





<u>Min Spanning Tree</u>

Input: weighted graph

Output: minimum-weight tree connecting all nodes

greedy: $O(\ |E|\ +\ |V|\ \log\ |V|\ )$

<u>Traveling Salesman (TSP)</u>

Input: complete weighted graph

Output: minimum-weight tour connecting all nodes

backtracking: $O(\ |V|\ !\ )$

Traveling-Salesman Problem is NP-Complete

- "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?"

$$\begin{aligned} \text{TSP} = \{\langle G, c, k\rangle :\ & G = (V, E) \text{ is a complete graph}, \\ & c \text{ is a function from } V \times V \to \mathbb{Z}, \\ & k \in \mathbb{Z}, \text{ and} \\ & G \text{ has a traveling-salesman tour with cost at most } k\} . \end{aligned}$$

- HAM-CYCLE $\leq_P$ TSP

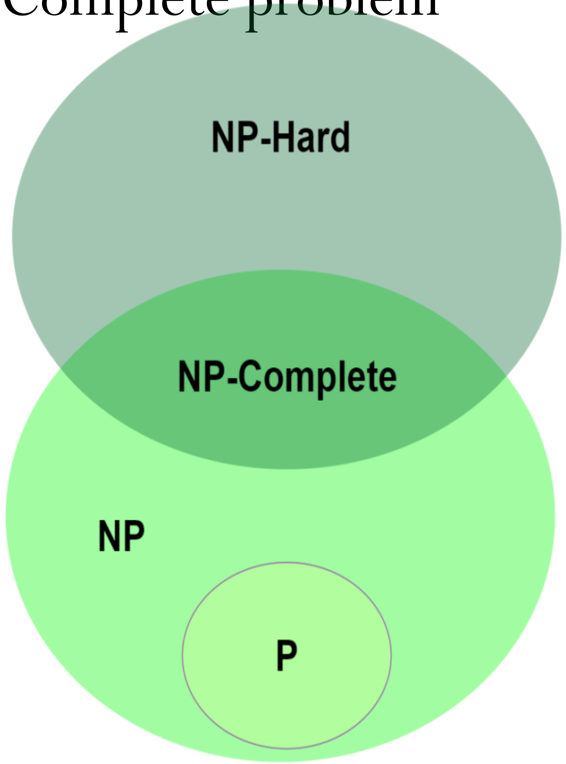
Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (Vol. 3, pp. 624-642). Cambridge: MIT press.

NP-Hard Problem

- L is **NP-Complete** *if* $L \in NP$ *then for all other* $L' \in NP, L' \propto L$
- L is **NP-Hard** if *for all other* $L' \in NP, L' \propto L$
- Note that an NP-Hard problem is a problem which is as hard as an NP-Complete problem and it's not necessary a decision problem.
- So, if an NP-complete problem is in P then P=NP
- if P != NP then all NP-complete problems are in NP-P
- For 3-SAT, TSP, Longest Path
  - Are we capable to write fast (polynomial-time) algorithms

    OR are these problems unsolvable ? we don't know: $P \neq NP$?
  - 3-SAT, TSP, Longest Path are so-called NP-hard problems

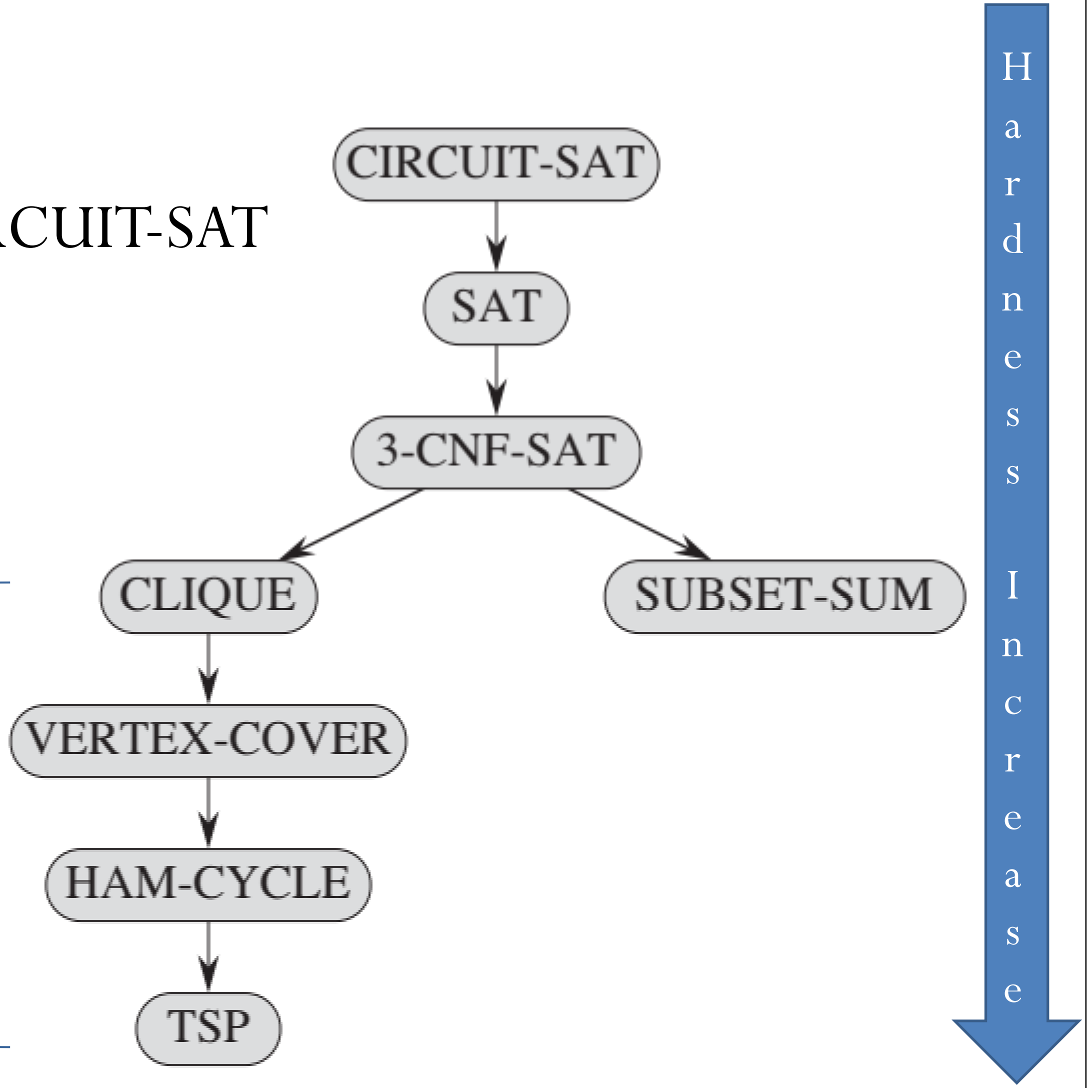
    if $P \neq NP$ (which most researchers believe to be the case)
    then NP-hard problems cannot be solved in polynomial time



NP-Complete problems

- The structure of NP-completeness
- Reduction from the NP-completeness of CIRCUIT-SAT
- Done!

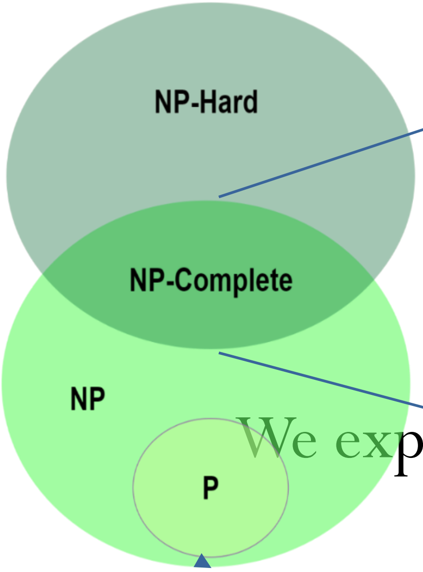
We explored all Graph Problems!

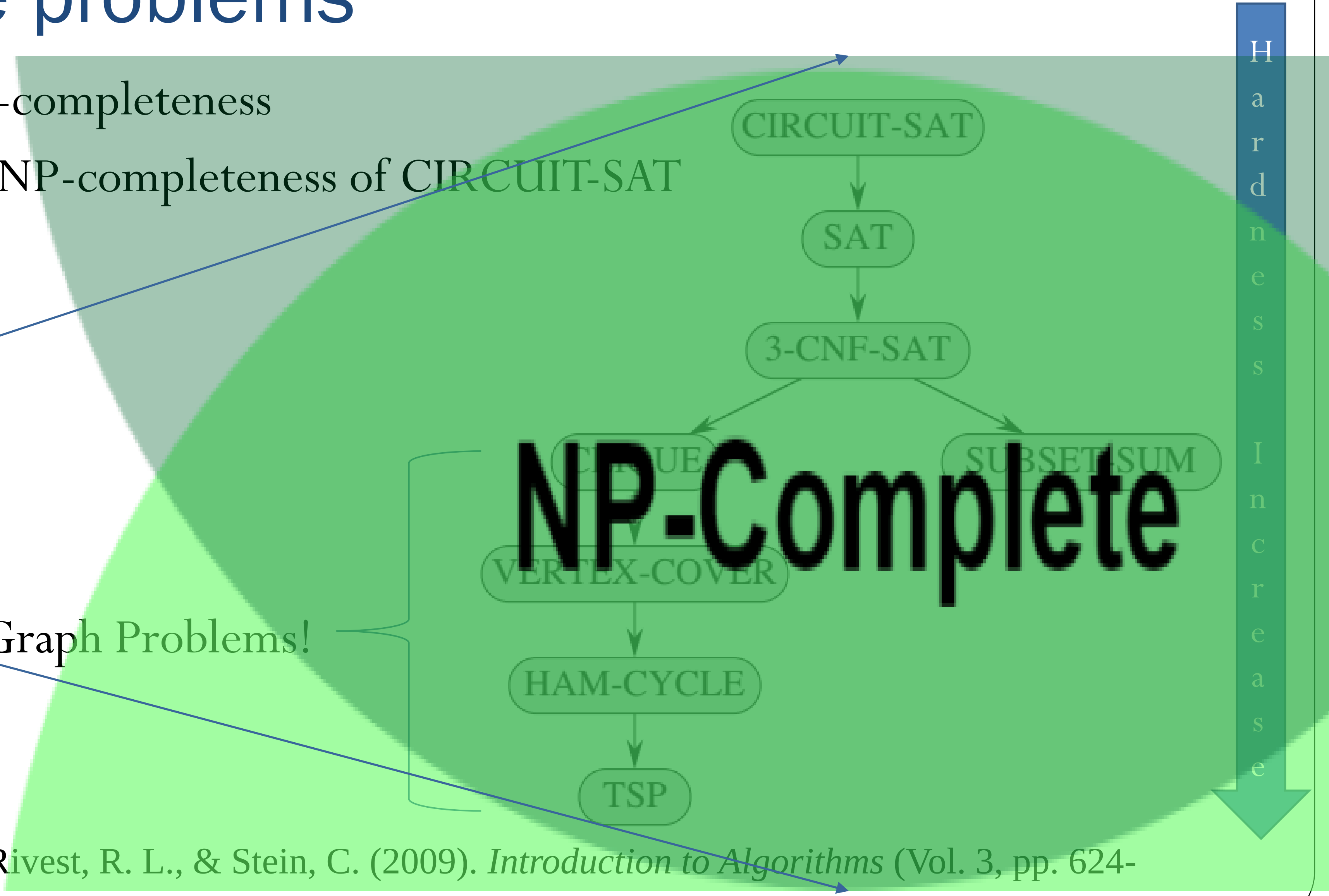


Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (Vol. 3, pp. 624-642). Cambridge: MIT press.

NP-Complete problems

- The structure of NP-completeness
- Reduction from the NP-completeness of CIRCUIT-SAT
- Done!





Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (Vol. 3, pp. 624-642). Cambridge: MIT press.

Travelling Salesman Problem (TSP)

Travelling Salesman Problem (TSP)

- "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?"
- **Routing problem:** Is there a route of at most 2000 kilometers passing through all of Germany's 15 largest cities?

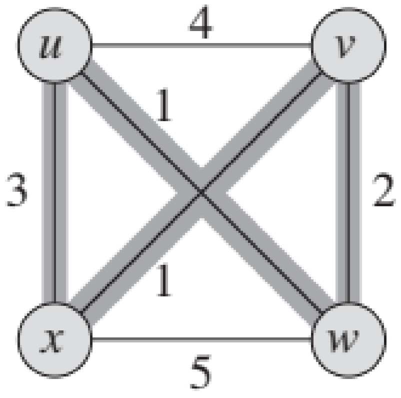
- **DNA sequencing:**
  - to determine the order of

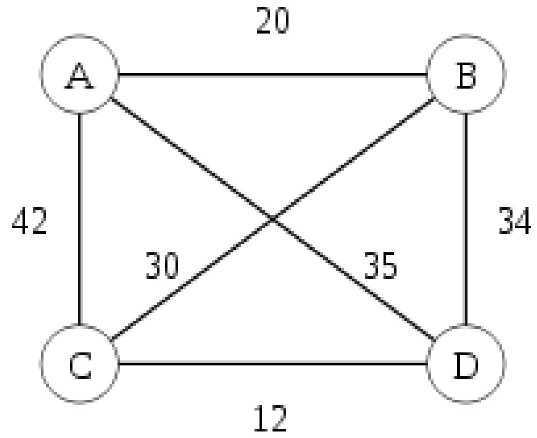
    Adenine, Thymine, Guanine, Cytosine (ATGC)
  - slightly modified Travelling Salesman Problem (TSP),
  - it appears as a sub-problem in DNA sequencing

https://en.wikipedia.org/wiki/Travelling_salesman_problem

Travelling Salesman Problem (TSP)

- TSP can be modelled as an undirected weighted graph, such that
  - cities are the graph's vertices,
  - paths are the graph's edges, and
  - a path's distance is the edge's weight.
- **Symmetric TSP:** the distance between two cities is the same in each opposite direction, forming an undirected graph.
- **Asymmetric TSP:** paths may not exist in both directions, or the distances might be different, forming a directed graph.



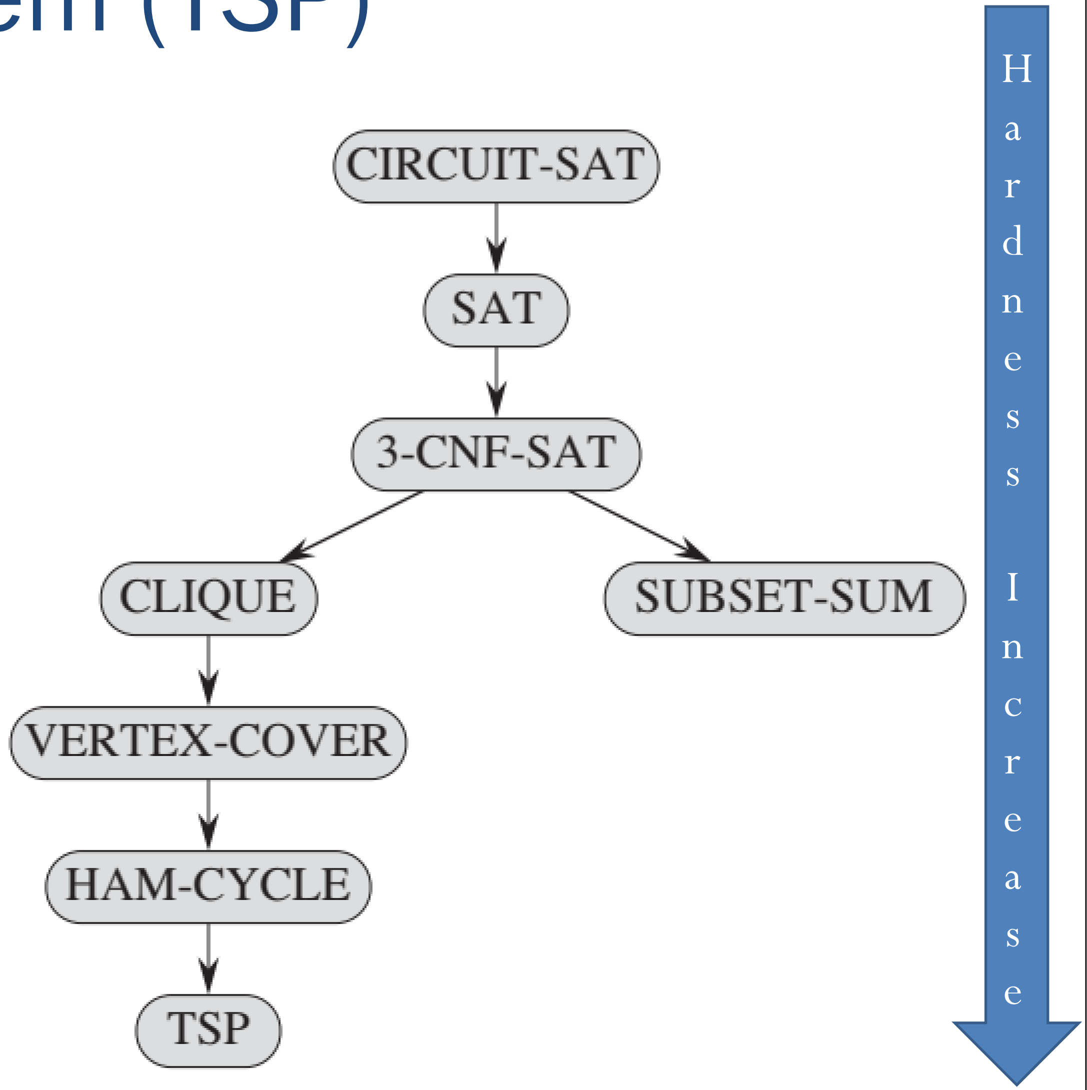


https://en.wikipedia.org/wiki/Travelling_salesman_problem

Travelling Salesman Problem (TSP)

- Hamiltonian-Cycle Problem is not more than a polynomial factor harder than Traveling Salesman Problem
  - HAM-CYCLE $\leq_P$ TSP
  - TSP is NP-Complete

- **An NP-hard problem:**
  - **Brute-Force Search:** Try all permutations (ordered combinations) to find shortest route.
  - Running time is O(n!), the factorial of the number of cities,
  - This solution becomes impractical even for 20 cities.



https://en.wikipedia.org/wiki/Travelling_salesman_problem

TSP - Approximation Algorithms

- TSP is an NP-hard problem
- TSP is solvable for few vertices problem about $V = 15$, depending upon computational power
- Even polynomial algorithms also fails on Big Graph for large value of V and E
  - Minimum Spanning Tree: Kruskal's $O(E \lg V)$ and Prim's $O(E + V \lg V)$ algorithm
  - Shortest Path : Djikstra's $O(E\lg V)$ and Bellman-Ford's $O(EV)$ algorithm
  - Limitation of computing power of single machine may make the polynomial algorithm fail on Big Graphs.
  - Parallel Computing environment might be the solution
    - for example, **Apache Spark GraphX**.
- TSP – Approximation Algorithm provides approximate solution to given Graph.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (Vol. 3, pp. 624-642). Cambridge: MIT press.

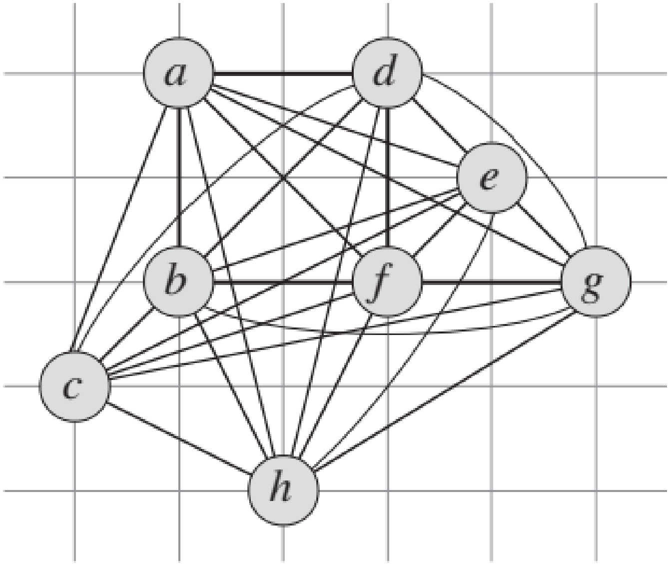
Travelling Salesman Problem (TSP)
Approximation Algorithms

Travelling Salesman Problem: Approximation

APPROX-TSP-TOUR(G, c)

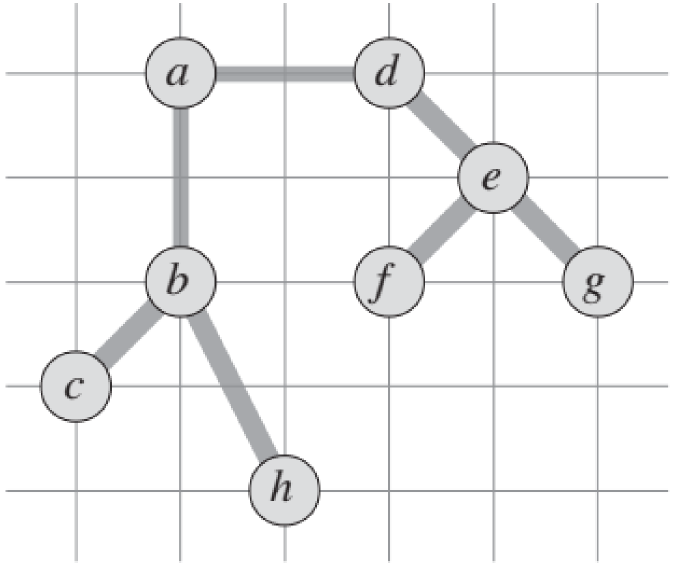
select a vertex $r \in G.V$ to be a "root" vertex
compute a minimum spanning tree T for G from root r
    using MST-PRIM(G, c, r)
let H be a list of vertices, ordered according to when they are first visited
    in a preorder tree walk of T
return the hamiltonian cycle H

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (Vol. 3, pp. 624-642). Cambridge: MIT press.



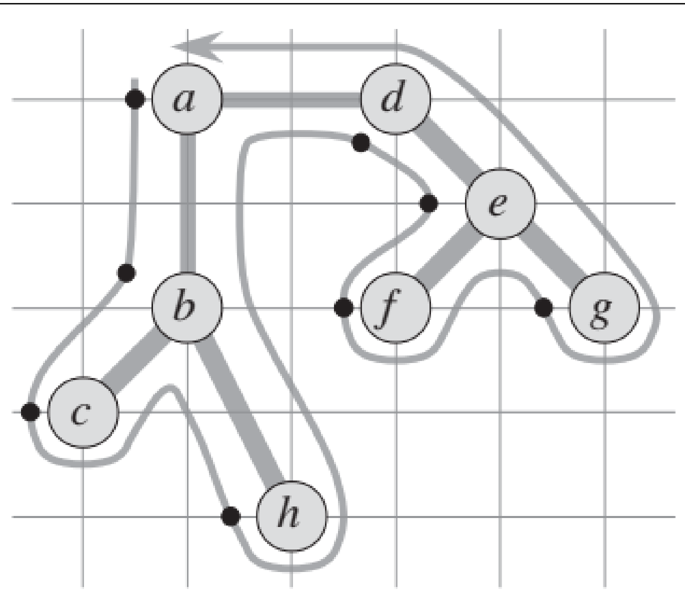
(a)

A complete undirected graph with ordinary Euclidean distance.



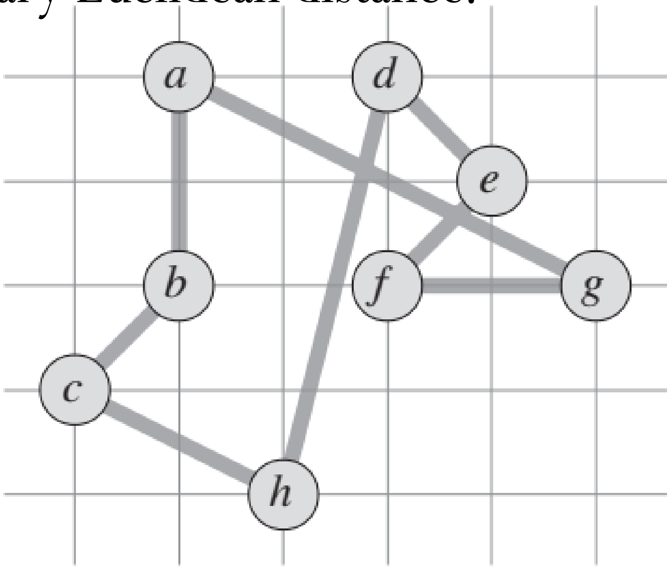
(b)

MST computed by MST-PRIM
'a' is the root vertex

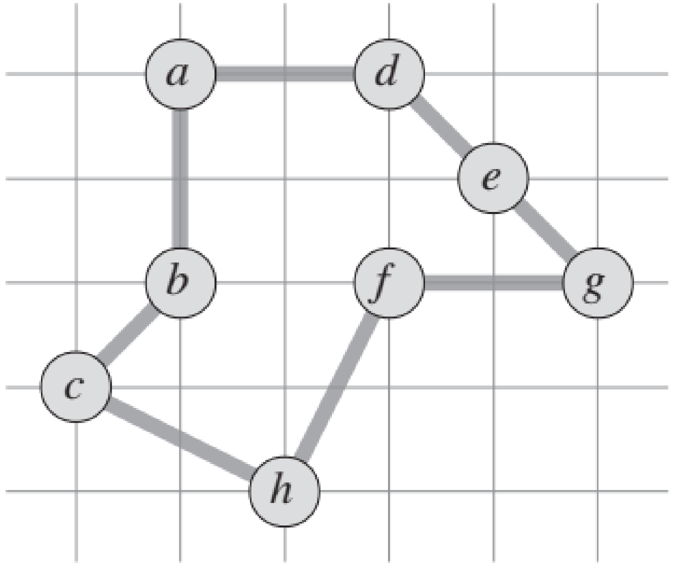


(c)

A walk of T , starting at a.
Yielding the ordering a; b; c; h; d; e; f; g.



(d)



(e)

(d) Tour H by APPROX-TSP-TOUR approximately 19.074.

(e) An optimal tour approximately 14.715.

APPROX-TSP-TOUR is $\theta(V^2)$

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (Vol. 3, pp. 624-642). Cambridge: MIT press.

Travelling Salesman Problem: Approximation

Christofides-Serdyukov algorithm

- In the worst case, is at most 1.5 times longer than the optimal solution.
- This remains the method with the best worst-case scenario.
- This gives a TSP tour which is at most 1.5 times the optimal.
- It was one of the first approximation algorithms
- Drawn attention to approximation algorithms as a practical approach to intractable problems.

https://en.wikipedia.org/wiki/Christofides_algorithm

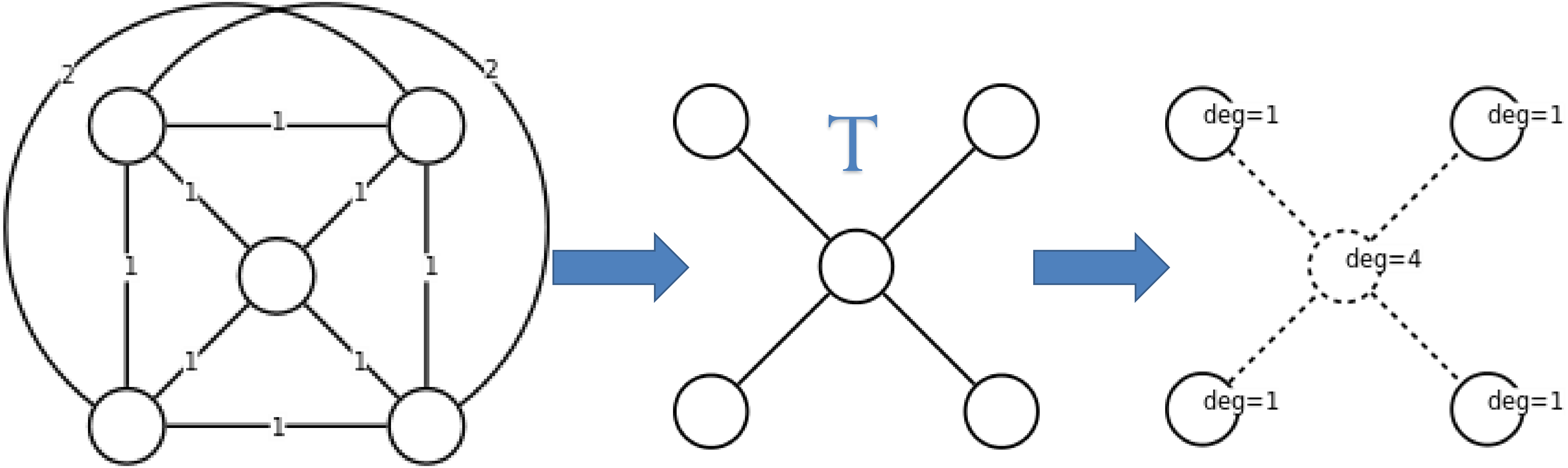
Christofides-Serdyukov algorithm

- Create a minimum spanning tree T of G.
- Let O be the set of vertices with odd degree in T. By the handshaking lemma, O has an even number of vertices.
- Find a minimum-weight perfect matching M in the induced subgraph given by the vertices from O.
- Combine the edges of M and T to form a connected multigraph H in which each vertex has even degree.
- Form an Eulerian circuit in H.
- Make the circuit found in previous step into a Hamiltonian circuit by skipping repeated vertices (shortcutting).

https://en.wikipedia.org/wiki/Christofides_algorithm

Christofides-Serdyukov algorithm

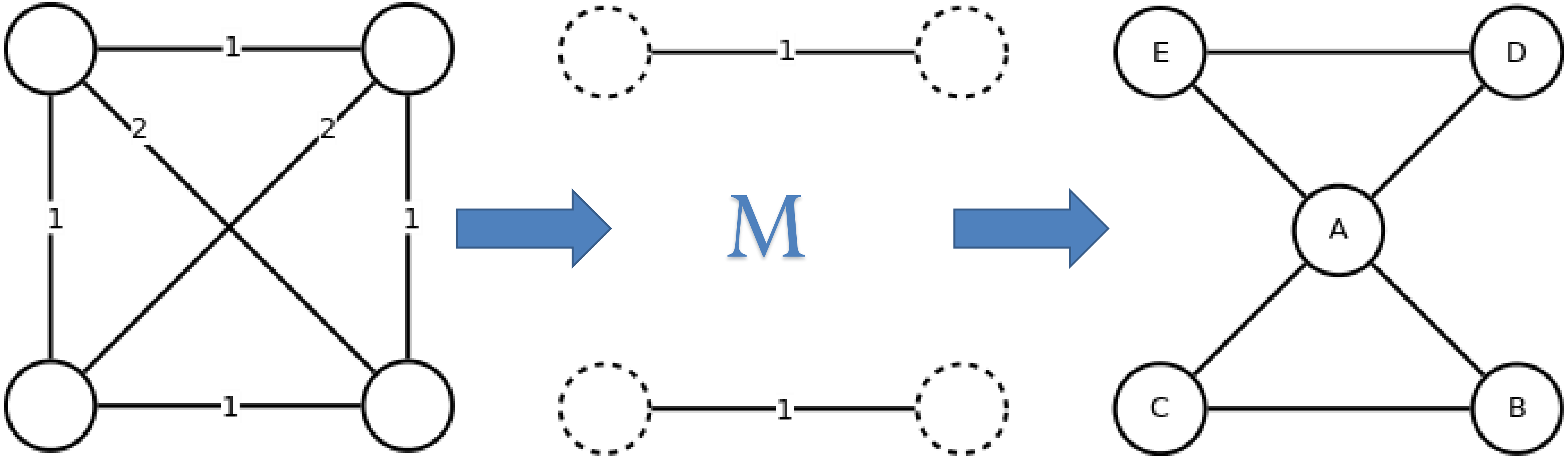
- Given: complete graph whose edge weights obey the triangle inequality
- Calculate minimum spanning tree T
- Calculate the set of vertices O with odd degree in T



https://en.wikipedia.org/wiki/Christofides_algorithm

Christofides-Serdyukov algorithm

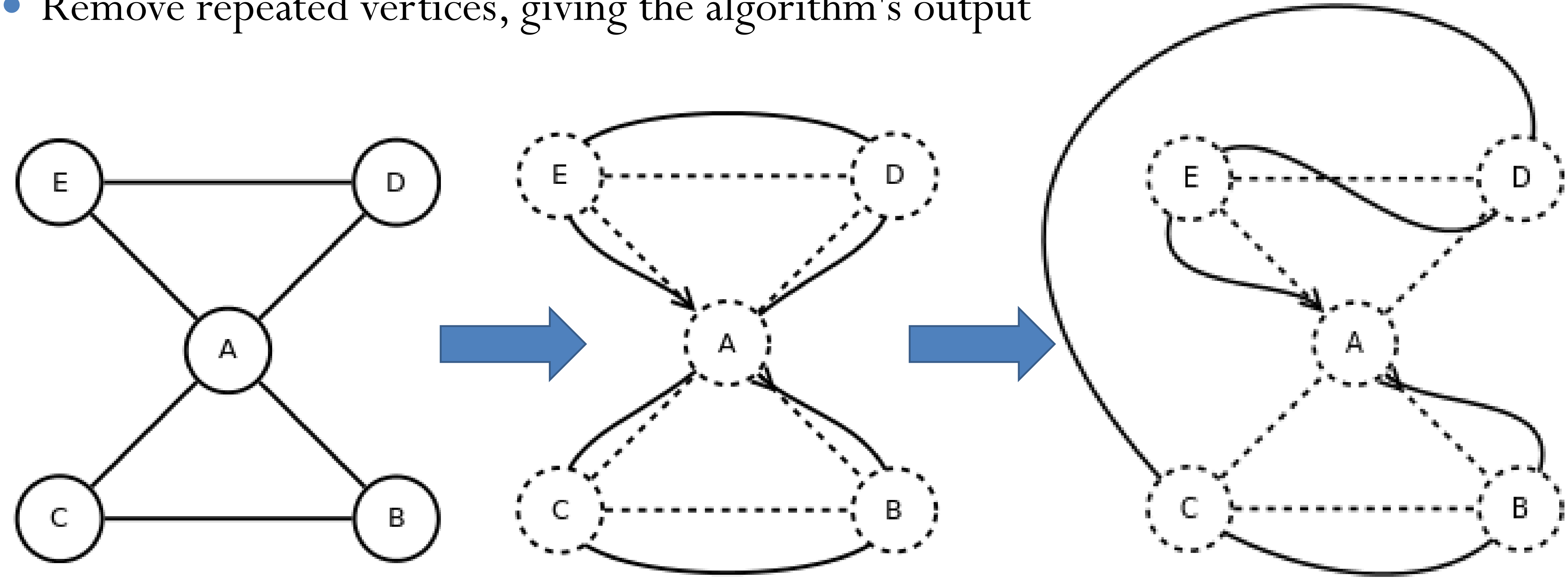
- Form the subgraph of G using only the vertices of O
- Construct a minimum-weight perfect matching M in this subgraph
- Unite matching and spanning tree $T \cup M$ to form an Eulerian multigraph



https://en.wikipedia.org/wiki/Christofides_algorithm

Christofides-Serdyukov algorithm

- Unite matching and spanning tree $T \cup M$ to form an Eulerian multigraph
- Calculate Euler tour
- Remove repeated vertices, giving the algorithm's output



https://en.wikipedia.org/wiki/Christofides_algorithm

PRIMES is in P
(A hope for NP problems in P)

Basics on Prime number

- A **prime number** (or a prime) is a natural number greater than 1 that is not a product of two smaller natural numbers.
- A natural number greater than 1 that is not prime is called a **composite number.**
- For example,
  - 5 is prime because the only ways of writing it as a product, 1×5 or 5×1, involve 5 itself.
  - 4 is composite because it is a product (2×2) in which both numbers are smaller than 4.
- The first 25 prime numbers (all the prime numbers less than 100) are: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97
- Primes are used in several routines in information technology,
  - such as public-key cryptography, which relies on the difficulty of factoring large numbers into their prime factors.

https://en.wikipedia.org/wiki/Prime_number
https://en.wikipedia.org/wiki/Prime_(disambiguation)

PRIMES is in P

- In 2002, it was shown that the problem of determining if a number is prime is in P.
- AKS (Agrawal–Kayal–Saxena) primality test, which is a famous research of IIT Kanpur, and authors received the 2006 Gödel Prize and the 2006 Fulkerson Prize
- AKS primality test: "an unconditional deterministic polynomial-time algorithm that determines whether an input number is prime or composite"
- The key idea is to find the coefficient of x^i in $((x + a)^n - (x^n + a))$
  - if all coefficients are multiple of n, then n is prime
  - else composite number
- **We work out it for $a = -1$.**
  - **What are the coefficient of x^i in $((x - 1)^n - (x^n - 1))$?**

Agrawal, Manindra, Neeraj Kayal, and Nitin Saxena. "PRIMES is in P." *Annals of mathematics* (2004): 781-793.
https://en.wikipedia.org/wiki/AKS_primality_test

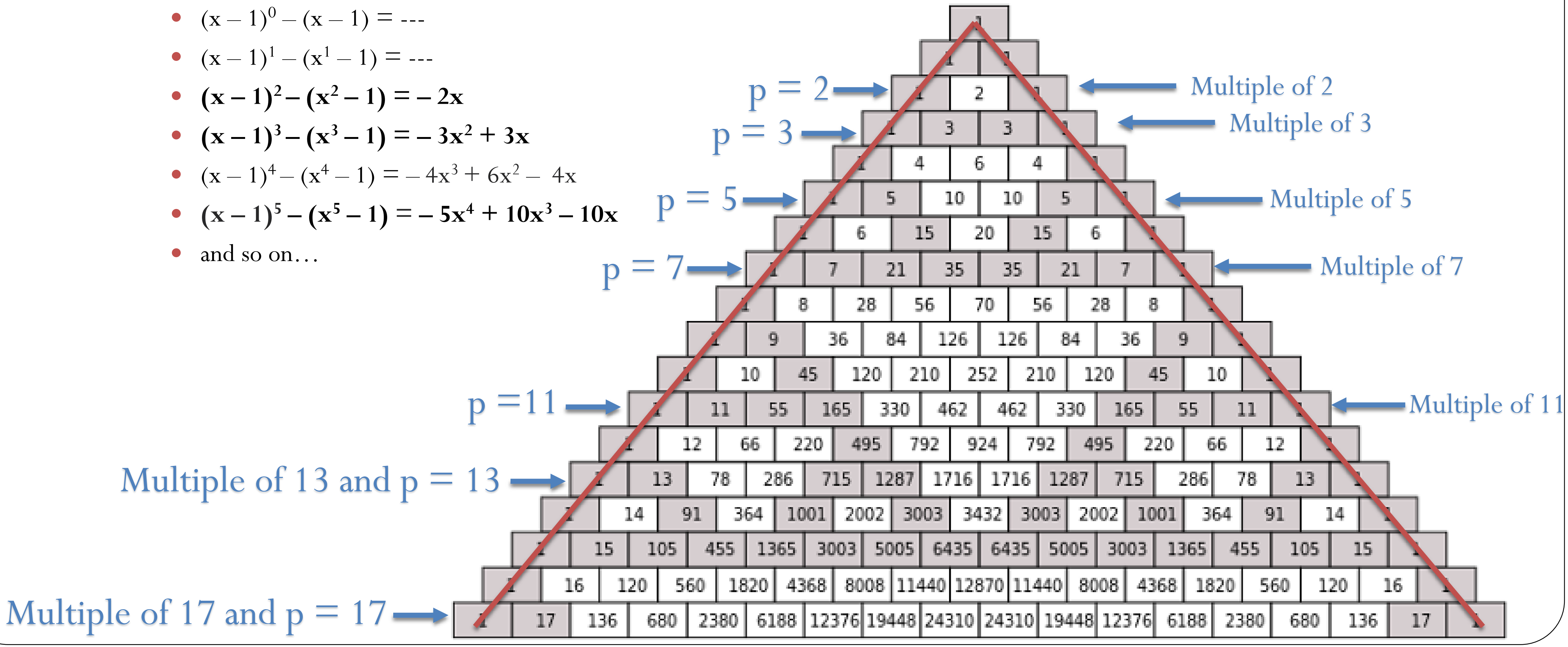
Pascal Triangle

- Coefficients of $(x - 1)^n$
  - $(x - 1)^0 = 1$
  - $(x - 1)^1 = x - 1$
  - $(x - 1)^2 = (x^2 - 2x + 1)$
  - $(x - 1)^3 = x^3 - 3x^2 + 3x - 1$
  - $(x - 1)^4 = x^4 - 4x^3 + 6x^2 - 4x + 1$
  - $(x - 1)^5 = x^5 - 5x^4 + 10x^3 - 10x + 1$
  - and so on…

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
1 10 45 120 210 252 210 120 45 10 1
1 11 55 165 330 462 462 330 165 55 11 1
1 12 66 220 495 792 924 792 495 220 66 12 1
1 13 78 286 715 1287 1716 1716 1287 715 286 78 13 1
1 14 91 364 1001 2002 3003 3432 3003 2002 1001 364 91 14 1
1 15 105 455 1365 3003 5005 6435 6435 5005 3003 1365 455 105 15 1
1 16 120 560 1820 4368 8008 11440 12870 11440 8008 4368 1820 560 120 16 1
1 17 136 680 2380 6188 12376 19448 24310 24310 19448 12376 6188 2380 680 136 17 1

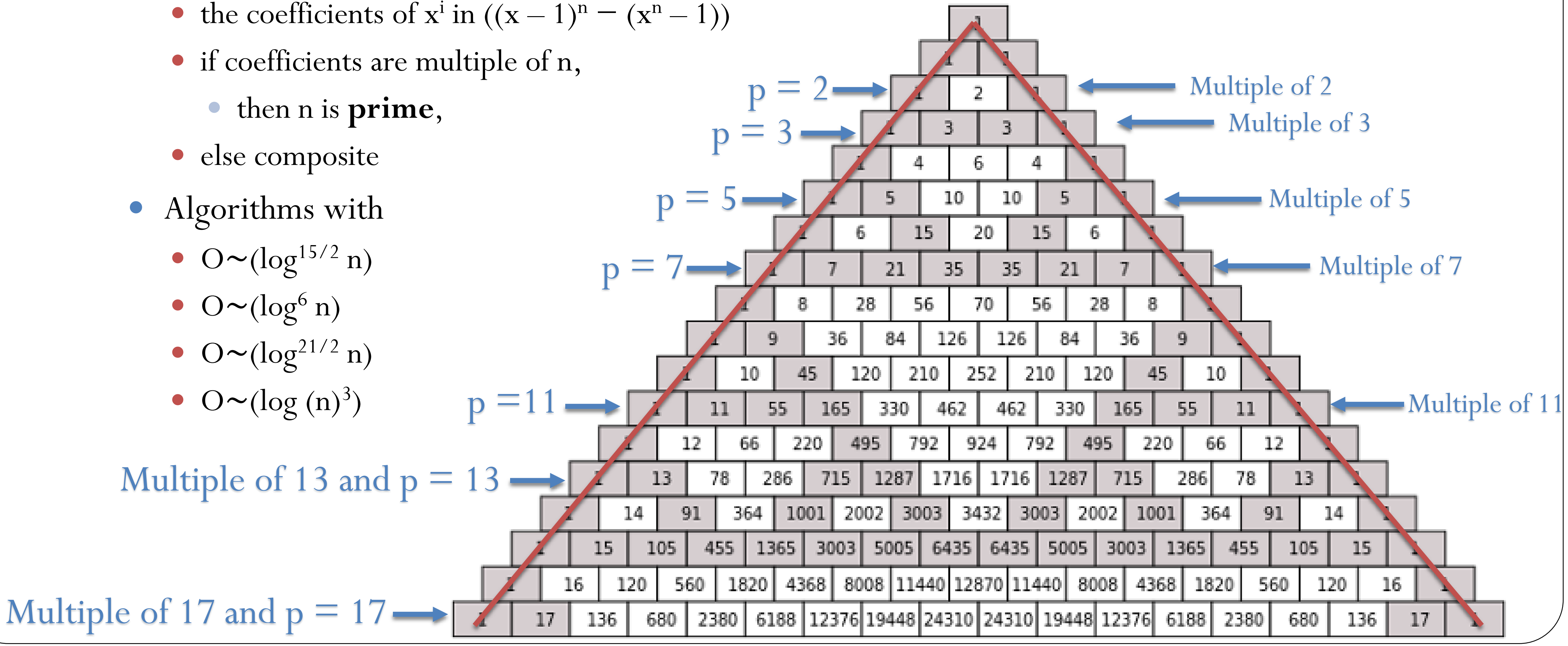
Prime and Pascals Triangle

- Coefficients of $(x - 1)^n - (x^n - 1)$
  - $(x - 1)^0 - (x - 1) =$ ---
  - $(x - 1)^1 - (x^1 - 1) =$ ---
  - $\mathbf{(x - 1)^2 - (x^2 - 1) = - 2x}$
  - $\mathbf{(x - 1)^3 - (x^3 - 1) = - 3x^2 + 3x}$
  - $(x - 1)^4 - (x^4 - 1) = - 4x^3 + 6x^2 - 4x$
  - $\mathbf{(x - 1)^5 - (x^5 - 1) = - 5x^4 + 10x^3 - 10x}$
  - and so on…



Prime and Pascals Triangle

- It is possible to write a polynomial time algorithm to find
  - the coefficients of x^i in $((x - 1)^n - (x^n - 1))$
  - if coefficients are multiple of n,
    - then n is **prime**,
  - else composite
- Algorithms with
  - $O{\sim}(\log^{15/2} n)$
  - $O{\sim}(\log^{6} n)$
  - $O{\sim}(\log^{21/2} n)$
  - $O{\sim}(\log (n)^3)$



More on Prime number

- Fast methods for primality test are available, such as Mersenne numbers.
- As of December 2018, the largest known prime number is a Mersenne prime with 24,862,048 decimal digits.

https://en.wikipedia.org/wiki/Prime_number
https://en.wikipedia.org/wiki/Prime_(disambiguation)

Millennium Problems

Millennium Problems

- The Millennium Prize Problems are seven problems in mathematics that were stated by the Clay Mathematics Institute on May 24, 2000.
- One of 7 Millennium Problems for which Clay Math Institute awards \$1,000,000 i.e., US\$1 million prize
- One-million dollar (*) question: $P = NP$?
- almost all researchers think $P \neq NP$

https://www.claymath.org/millennium-problems

https://www.claymath.org/millennium-problems/millennium-prize-problems

https://en.wikipedia.org/wiki/Millennium_Prize_Problems

Millennium Problems

- Yang–Mills and Mass Gap
- Riemann Hypothesis
- P vs NP Problem: If it is easy to check that a solution to a problem is correct, is it also easy to solve the problem? This is the essence of the P vs NP question. Typical of the NP problems is that of the Hamiltonian Path Problem: given N cities to visit, how can one do this without visiting a city twice? If you give me a solution, I can easily check that it is correct. But I cannot so easily find a solution.
- Navier–Stokes Equation
- Hodge Conjecture
- Poincaré Conjecture
- Birch and Swinnerton-Dyer Conjecture

https://www.claymath.org/millennium-problems

Millennium Problems

- To date, the only Millennium Prize problem to have been solved is the Poincaré conjecture,
- A century passed between its formulation in 1904 by Henri Poincaré and its solution by Grigoriy Perelman, announced in preprints posted on ArXiv.org in 2002 and 2003.
- Grigoriy Perelman is the Russian mathematician Grigori Perelman.
- He declined the prize money.
- Perelman was selected to receive the Fields Medal for his solution, but he declined the award.

https://www.claymath.org/millennium-problems/poincar%C3%A9-conjecture
https://en.wikipedia.org/wiki/Millennium_Prize_Problems

Conclusions

Conclusions

- This presentation is
  - more related to Computing Algorithms.
  - less related to Theory of Computation (Language, Turing Machine, etc.).
  - more focused toward modern Graph Analytics.
  - more focused toward applied mathematics i.e., Computer Science and Engineering.
  - less focused toward core Mathematics.

ขอบคุณ

Thai

Grazie

Italian

תודה רבה

Hebrew

Gracias

Spanish

Спасибо

Russian

English

Thank You

Obrigado

Portuguese

شكرا

Arabic

Merci

French

https://sites.google.com/site/animeshchaturvedi07

多謝

Traditional Chinese

Danke

German

धन्यवाद

Hindi

நன்றி

Tamil

Tamil

多谢

Simplified Chinese

ありがとうございました

Japanese

감사합니다

Korean