NP COMPLETENESS & APPROXIMATION

Partha P Chakrabarti

Indian Institute of Technology Kharagpur

Algorithm Design by Recursion Transformation

- Algorithms and Programs
- Pseudo-Code
- Algorithms + Data Structures = Programs
- Initial Solutions + Analysis + Solution Refinement + Data Structures = Final Algorithm
- Use of Recursive Definitions as Initial Solutions
- Recurrence Equations for Proofs and Analysis
- Solution Refinement through Recursion Transformation and Traversal
- Data Structures for saving past computation for future use

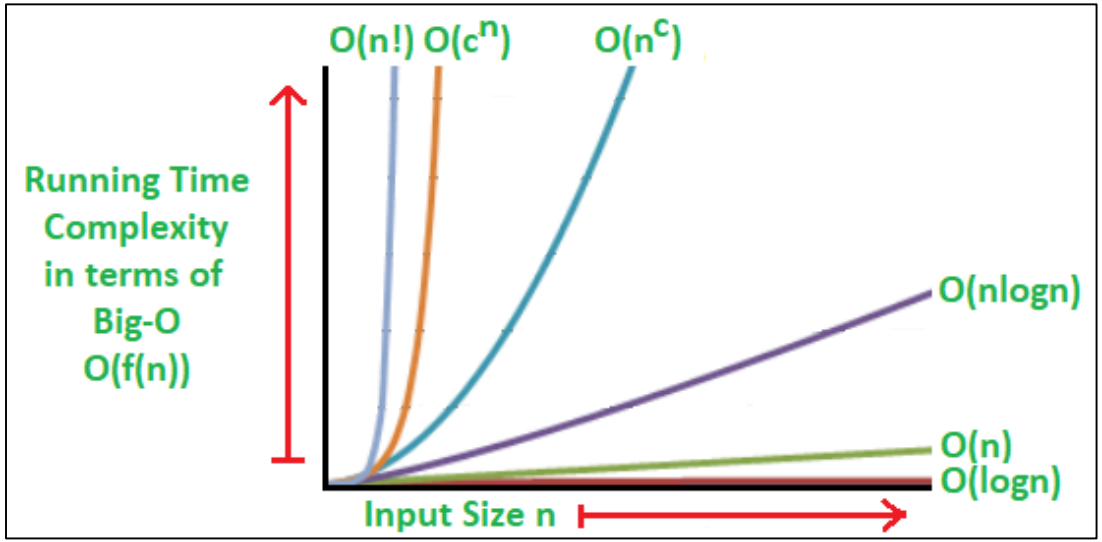
1. Initial Solution
   a. Recursive Definition – A set of Solutions
   b. Inductive Proof of Correctness
   c. Analysis Using Recurrence Relations
2. Exploration of Possibilities
   a. Decomposition or Unfolding of the Recursion Tree
   b. Examination of Structures formed
   c. Re-composition Properties
3. Choice of Solution & Complexity Analysis
   a. Balancing the Split, Choosing Paths
   b. Identical Sub-problems
4. Data Structures & Complexity Analysis
   a. Remembering Past Computation for Future
   b. Space Complexity
5. Final Algorithm & Complexity Analysis
   a. Traversal of the Recursion Tree
   b. Pruning
6. Implementation
   a. Available Memory, Time, Quality of Solution, etc

Overview of Algorithm Design

1. Initial Solution
   a. Recursive Definition – A set of Solutions
   b. Inductive Proof of Correctness
   c. Analysis Using Recurrence Relations
2. Exploration of Possibilities
   a. Decomposition or Unfolding of the Recursion Tree
   b. Examination of Structures formed
   c. Re-composition Properties
3. Choice of Solution & Complexity Analysis
   a. Balancing the Split, Choosing Paths
   b. Identical Sub-problems
4. Data Structures & Complexity Analysis
   a. Remembering Past Computation for Future
   b. Space Complexity
5. Final Algorithm & Complexity Analysis
   a. Traversal of the Recursion Tree
   b. Pruning
6. Implementation
   a. Available Memory, Time, Quality of Solution, etc

1. Core Methods
   a. Divide and Conquer
   b. Greedy Algorithms
   c. Dynamic Programming
   d. Branch-and-Bound
   e. Analysis using Recurrences
   f. Advanced Data Structuring
2. Important Problems to be addressed
   a. Sorting and Searching
   b. Strings and Patterns
   c. Trees and Graphs
   d. Combinatorial Optimization
3. Complexity & Advanced Topics
   a. Time and Space Complexity
   b. Lower Bounds
   c. Polynomial Time, NP-Hard
   d. Parallelizability, Randomization

Complexity: Order of Growth: Graphically



O(n!),O(c^n),O(n^c) - Worst

O(nlogn) - Bad

O(n) - Fair

O(logn) - Good

O(1) - Best

Problem Complexity Vs Algorithm Complexity

Decidable / Undecidable Problems

Hard Problems / Easy Problems

Algorithms and Lower Bounds

Optimal Algorithms

Polynomial / Exponential Algorithms

NP Compete or NP Hard Problems

Optimization and Decision Problems

- **Optimization Problem**: Optimization problems are those for which the objective is to maximize or minimize some values. For example:
  - Finding the minimum number of colors needed to color a given graph
  - Finding the shortest path between two vertices in a graph
- **Decision Problem**: There are many problems for which the answer is a Yes or a No. These are known as decision problems. For example:
  - Whether a given graph can be colored by only 4-colors
  - Finding Hamiltonian cycle in a graph is not a decision problem, whereas checking a graph is Hamiltonian or not is a decision problem
    - Hamiltonian Path in an undirected graph is a path that visits each vertex exactly once
    - A Hamiltonian cycle is a Hamiltonian Path such that there is an edge (in the graph) from the last vertex to the first vertex of the Hamiltonian Path
    - Determine whether a given graph contains Hamiltonian Cycle or not
    - If there exists a closed walk in the connected graph that visits every vertex of the graph exactly once. (except starting vertex) without repeating the edges, then such a graph is called as a Hamiltonian graph.

Decision Problems and Languages

- **Language**:
  Every decision problem can have only two answers, yes or no. Hence, a decision problem may belong to a language if it provides an answer 'yes' for a specific input.
- A language is the totality of inputs for which the answer is Yes.
- Decision Problem
  - Problem X is a set of strings
  - Instance s is one string
  - Algorithm A solves problem X:

$$\begin{aligned} A(s) &= \textit{Yes}, \ \textit{if} \ s \in X \\ &= \textit{No}, \ \textit{if} \ s \notin X \end{aligned}$$

The Class P (Problems with Polynomial Time Solutions)

- **P-Class**:
  The class P consists of those problems that are solvable in polynomial time $O(n^k)$ in worst-case, where k is constant.

  These problems are called *tractable*, while others are called *intractable* or *superpolynomial*.

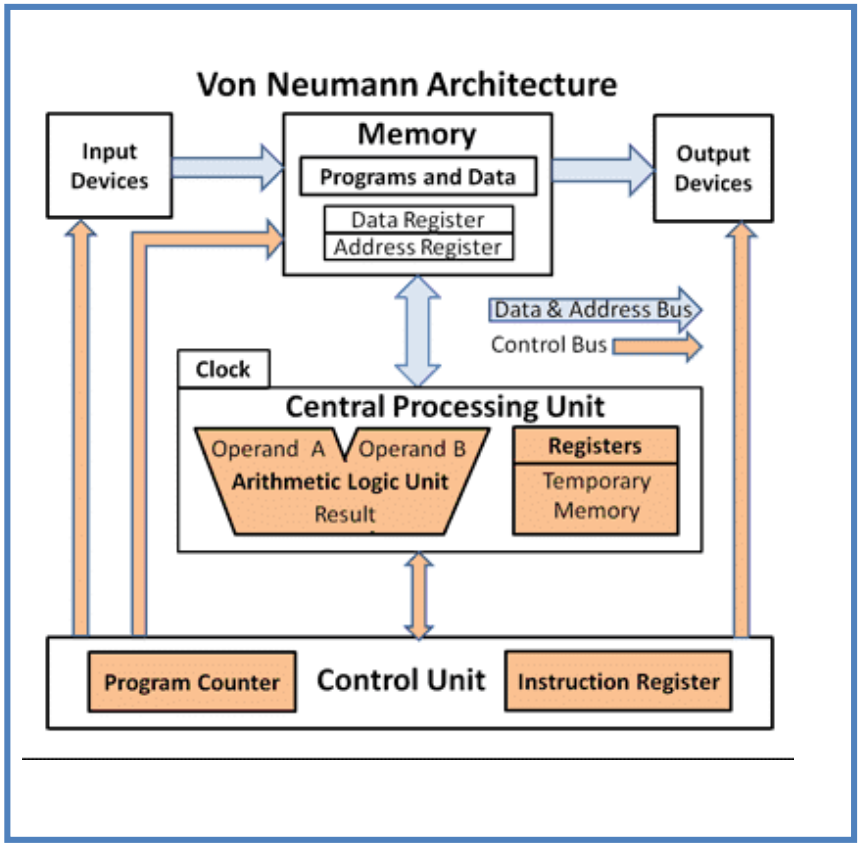
  Formally, an algorithm is polynomial time algorithm, if there exists a polynomial $p(n)$ such that the algorithm can solve any instance of size n in a time $O(p(n))$.

  Problem requiring $\Omega(n^{50})$ time to solve are essentially intractable for large n. Most known polynomial time algorithm run in time $O(n^k)$ for fairly low value of k.

  The advantages in considering the class of polynomial-time algorithms is that all reasonable deterministic single processor model of computation can be simulated on each other with at most a polynomial slow-down

- **Tractable Problems**:
  Matrix Chain Multiplication, Single Source Shortest Path, All Pair Shortest Path, Minimum Spanning Tree

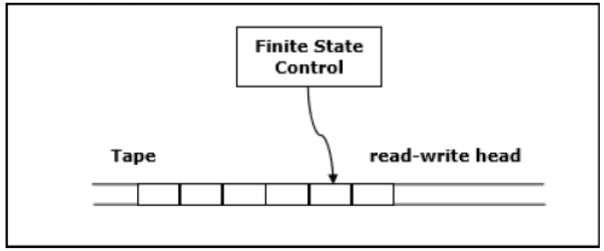
Computer Architecture, Sample Instruction Set



Instruction	*Code*	*Operands*			*Action*
	d_1	d_2	d_3	d_4	
HALT	0				Stops the execution of the program
LOAD	1	R_D	M_S		$R_D \leftarrow M_S$
STORE	2	M_D		R_S	$M_D \leftarrow R_S$
ADDI	3	R_D	R_{S1}	R_{S2}	$R_D \leftarrow R_{S1} + R_{S2}$
ADDF	4	R_D	R_{S1}	R_{S2}	$R_D \leftarrow R_{S1} + R_{S2}$
MOVE	5	R_D	R_S		$R_D \leftarrow R_S$
NOT	6	R_D	R_S		$R_D \leftarrow \overline{R_S}$
AND	7	R_D	R_{S1}	R_{S2}	$R_D \leftarrow R_{S1}$ AND R_{S2}
OR	8	R_D	R_{S1}	R_{S2}	$R_D \leftarrow R_{S1}$ OR R_{S2}
XOR	9	R_D	R_{S1}	R_{S2}	$R_D \leftarrow R_{S1}$ XOR R_{S2}
INC	A	R			$R \leftarrow R + 1$
DEC	B	R			$R \leftarrow R - 1$
ROTATE	C	R	*n*	0 or 1	$Rot_n R$
JUMP	D	R	*n*		IF $R_0 \neq R$ then PC = *n*, otherwise continue

Deterministic Turing Machines and Class P

One of these models is one-tape Deterministic Turing Machine (DTM). This machine consists of a finite state control, a read-write head and a two-way tape with infinite sequence as schematized below:

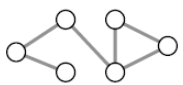
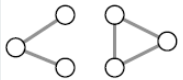


A program for a deterministic Turing machine specifies the following information:

- A finite set of tape symbols (input symbols and a blank symbol)
- A finite set of states
- A transition function

In algorithmic analysis, if a problem is solvable in polynomial time by a deterministic one tape Turing machine, the problem belongs to P class.

Tractable Problems with Polynomial Time Algorithms

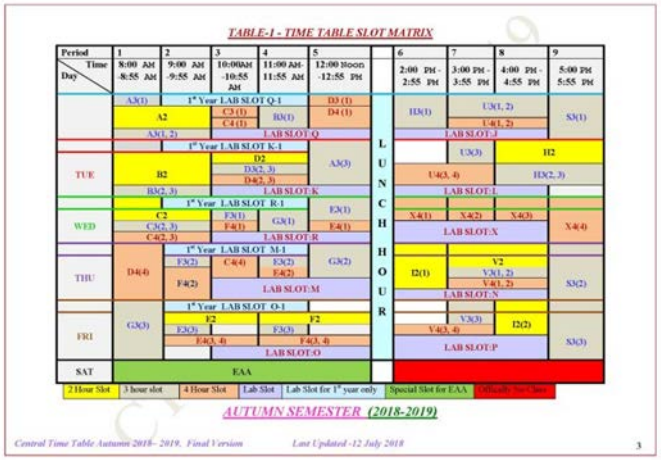
problem	description	poly-time algorithm	yes	no
MULTIPLE	Is x a multiple of y ?	grade-school division	51, 17	51, 16
REL-PRIME	Are x and y relatively prime ?	Euclid's algorithm	34, 39	34, 51
PRIMES	Is x prime ?	Agrawal–Kayal–Saxena	53	51
EDIT-DISTANCE	Is the edit distance between x and y less than 5 ?	Needleman–Wunsch	niether neither	acgggt ttttta
L-SOLVE	Is there a vector x that satisfies $Ax = b$?	Gauss–Edmonds elimination	$\begin{bmatrix} 0 & 1 & 1 \\ 2 & 4 & -2 \\ 0 & 3 & 15 \end{bmatrix}, \begin{bmatrix} 4 \\ 2 \\ 36 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$
U-CONN	Is an undirected graph G connected?	depth-first search		

Intractable Problems

- Evaluation of a Game instance of Chess
- Cryptarithmetic Problem

```
    B O B
  x B O B
  -------
    M E O Y
  M I L O
M E O Y
-------
M A R L E Y
```

- Time-Table Scheduling

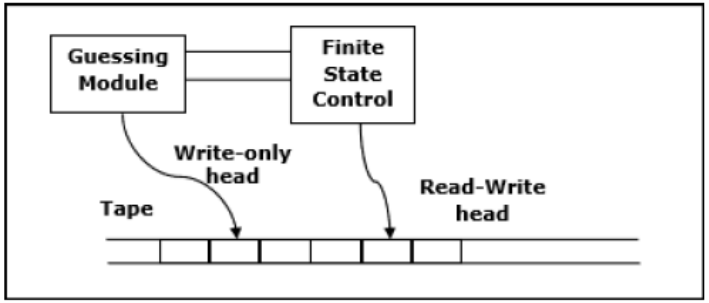


- Boolean Satisfiability
- Vertex Cover
- Graph Colouring / Independent Set / Max Clique
- Subset Sum / Knapsack
- Hamiltonian Cycle / Travelling Salesperson
- Multiprocessor Task Scheduling
- Cutting and Packing Problems

Non-Deterministic Turing Machines and Class NP

To solve the computational problem, another model is the Non-deterministic Turing Machine (NDTM).

The structure of NDTM is similar to DTM, however here we have one additional module known as the guessing module, which is associated with one write-only head. NDTM is schematized as:



If the problem is solvable in polynomial time by a non-deterministic Turing machine, the problem belongs to NP class

Choose Instruction:
Choose(1:n)

Revisiting the Coins Problem

Given a set C of n coins having denomination values $\{c_1, c_2, \ldots, c_n\}$ and a desired final value of V, find the minimum number of coins to be chosen from C to get an exact value of V from the sum of denominations of the chosen subset

example: $C = \{8, 6, 5, 2, 1\}$, $V = 11$

$S_1 = \{8, 3, 1\}$, $S_2 = \{6, 5\}$ ← minimum

Coins (S, T, x, z, n)

S: set of coins selected till now

T: remaining set of coins from which we can select

x: value of set S

z: remaining value desired to be chosen from T

n: the number of coins selected

Coins (NULL, C, 0, V, 0)

→ Base condition

→ Recursive condition

Revisiting the Coins Problem: Non-Deterministic Choice

$\langle P, d \rangle$ = coins (S, T, x, z, n)

{ Let $S = \{s_1, s_2, \ldots, s_n\}$

$T = \{t_1, t_2, \ldots, t_m\}$

BASE CONDITIONS

if $(z = 0)$ return $(\langle S, n \rangle)$

if $(z < 0)$ return $(\langle NULL, \infty \rangle)$

if $(T = NULL)$ return $(\langle NULL, \infty \rangle)$

$P_{min} = NULL$

$min = \infty$

RECURSIVE CONDITION

for $(i = 1$ to $m)$ do

{ $W = S + \{t_i\}$

$U = T - \{t_i\}$

$\langle P', d' \rangle$ = coins $(W, U, x + t_i, z - t_i, n + 1)$

if $(d' < min)$

{ $min = d'$

$P_{min} = P'$

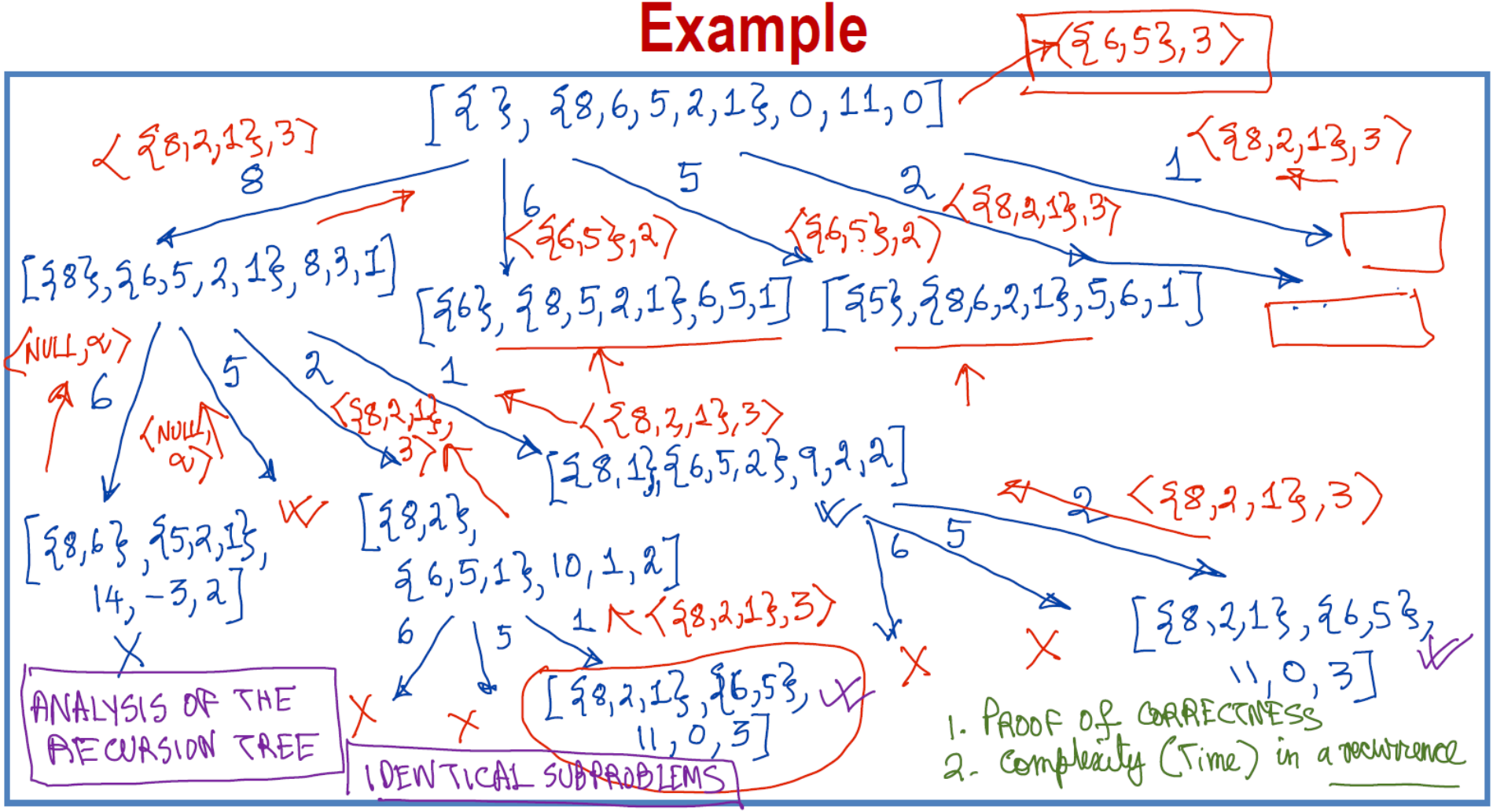
}

}

return $(\langle P_{min}, min \rangle)$

}

Example


[{}, {8,6,5,2,1}, 0, 11, 0]
⟨{6,5}, 3⟩
⟨{8,2,1}, 3⟩
[{8}, {6,5,2,1}, 8, 3, 1]
[{6}, {8,5,2,1}, 6, 5, 1]
[{5}, {8,6,2,1}, 5, 6, 1]
[{8,1}, {6,5,2}, 9, 2, 2]
[{8,6}, {5,2,1}, 14, -3, 2]
[{8,2}, {6,5,1}, 10, 1, 2]
[{8,2,1}, {6,5}, 11, 0, 3]
[{8,2,1}, {6,5}, 11, 0, 3]
⟨NULL, ∞⟩
ANALYSIS OF THE RECURSION TREE
IDENTICAL SUBPROBLEMS
1. PROOF of CORRECTNESS
2. Complexity (Time) in a recurrence

Other Examples of Problems in NP

- Boolean Satisfiability
- Vertex Cover
- Graph Colouring / Independent Set / Max Clique
- Subset Sum / Knapsack
- Hamiltonian Cycle / Travelling Salesperson
- Multiprocessor Task Scheduling
- Cutting and Packing Problems

Other Examples of Problems in NP

- Boolean Satisfiability
- Vertex Cover
- Graph Colouring / Independent Set / Max Clique
- Subset Sum / Knapsack
- Hamiltonian Cycle / Travelling Salesperson
- Multiprocessor Task Scheduling
- Cutting and Packing Problems

Other Examples of Problems in NP

- Boolean Satisfiability
- Vertex Cover
- Graph Colouring / Independent Set / Max Clique
- Subset Sum / Knapsack
- Hamiltonian Cycle / Travelling Salesperson
- Multiprocessor Task Scheduling
- Cutting and Packing Problems

Class NP and Polynomial Time Verifiability

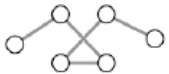
The class NP consists of those problems that are verifiable in polynomial time.

NP is the class of decision problems for which it is easy to check the correctness of a claimed answer, with the aid of a little extra information.

Hence, we are not asking for a way to find a solution, but only to verify that an alleged solution really is correct.

Every problem in this class can be solved in exponential time using exhaustive search.

Class NP and Intractable Problems

problem	description	poly-time algorithm	yes	no
L-SOLVE	Is there a vector x that satisfies $Ax = b$?	Gauss–Edmonds elimination	$\begin{bmatrix} 0 & 1 & 1 \\ 2 & 4 & -2 \\ 0 & 3 & 15 \end{bmatrix}, \begin{bmatrix} 4 \\ 2 \\ 36 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$
COMPOSITES	Is x composite ?	Agrawal–Kayal–Saxena	51	53
FACTOR	Does x have a nontrivial factor less than y ?	???	(56159, 50)	(55687, 50)
SAT	Given a CNF formula, does it have a satisfying truth assignment?	???	$\neg x_1 \vee x_2 \vee \neg x_3$ $x_1 \vee \neg x_2 \vee x_3$ $\neg x_1 \vee \neg x_2 \vee x_3$	$\neg x_2$ $x_1 \vee x_2$ $\neg x_1 \vee x_2$
HAMILTON-PATH	Is there a simple path between u and v that visits every node?	???		

P and NP

Every decision problem that is solvable by a deterministic polynomial time algorithm is also solvable by a polynomial time non-deterministic algorithm.

All problems in P can be solved with polynomial time algorithms, whereas all problems in NP - P are intractable.

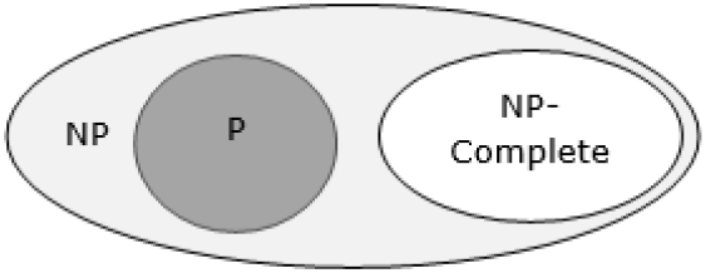
It is not known whether P = NP. However, many problems are known in NP with the property that if they belong to P, then it can be proved that P = NP.

If P $\neq$ NP, there are problems in NP that are neither in P nor in NP-Complete.

The problem belongs to class P if it is easy to find a solution for the problem. The problem belongs to NP, if it is easy to check a solution that may have been very tedious to find.

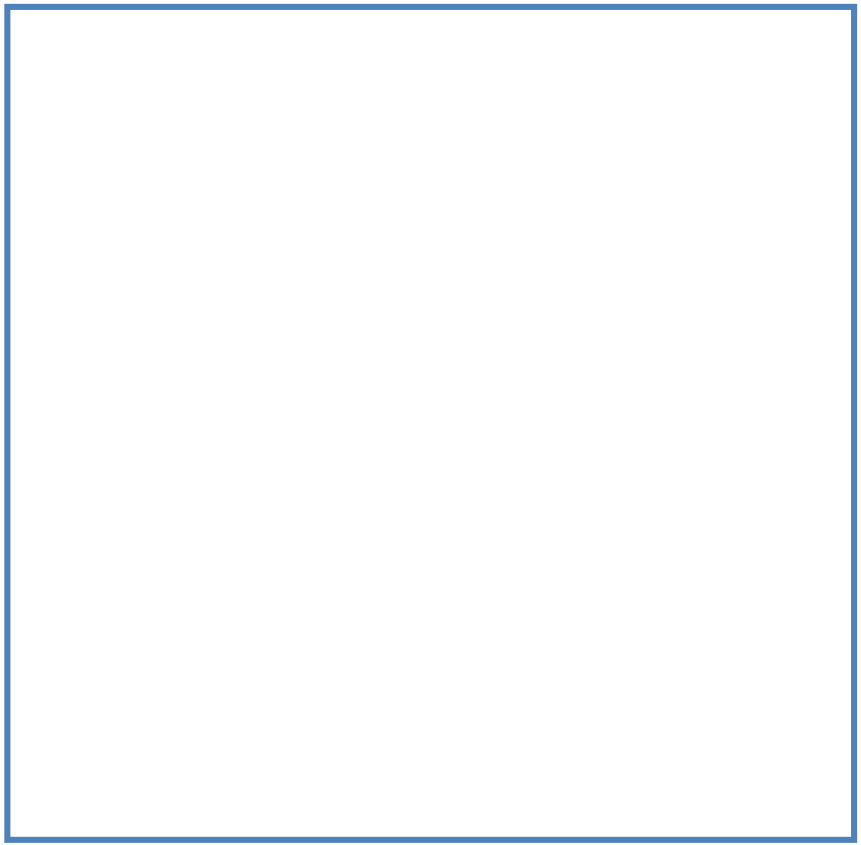
NP-Complete and NP-Hard Problems

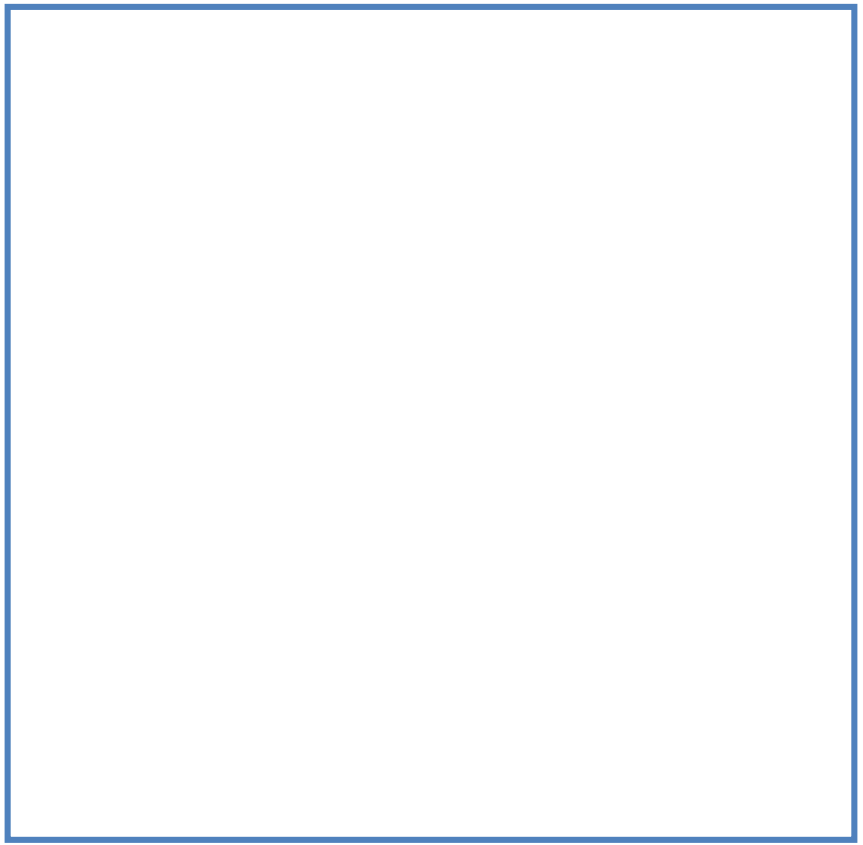
- A problem is in the class NPC if it is in NP and is as hard as any problem in NP.
- A problem is NP-hard if all problems in NP are polynomial time reducible to it, even though it may not be in NP itself.



- If a polynomial time algorithm exists for any of these problems, all problems in NP would be polynomial time solvable.
- These problems are called NP-complete.
- The phenomenon of NP-completeness is important for both theoretical and practical reasons.

Relating Coins Problem and 2-Processor Scheduling





NP-Completeness

- A language **B** is **NP-Complete** if it satisfies two conditions
  - **B** is in NP
  - Every **A** in NP is polynomial time reducible to **B**.
- If a language satisfies the second property, but not necessarily the first one, the language **B** is known as **NP-Hard**.
- Informally, a search problem **B** is NP-Hard if there exists some NP-Complete problem **A** that Turing reduces to **B**
- The problem in NP-Hard cannot be solved in polynomial time, until P = NP.
- If a problem is proved to be NPC, there is no need to waste time on trying to find an efficient algorithm for it. Instead, we can focus on design approximation algorithm.

Cook-Levine Theorem
(Boolean Satisfiability is NP-Complete)

Implications of Cook–Levin

SAT

INDEPENDENT-SET poly-time reduces to CIRCUIT-SAT

3-SAT

INDEPENDENT-SET

DIR-HAM-CYCLE

3-COLOR

SUBSET-SUM

VERTEX-COVER

HAM-CYCLE

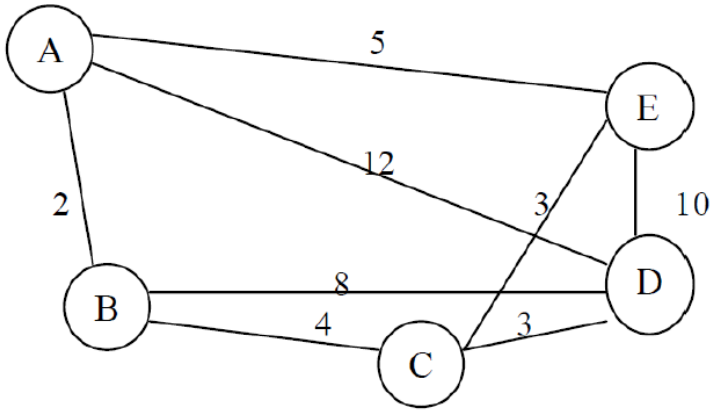
KNAPSACK

SET-COVER

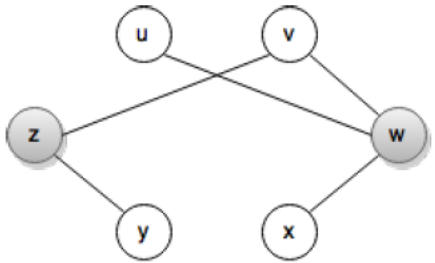
All of these problems (and many, many more) poly-time reduce to SAT.

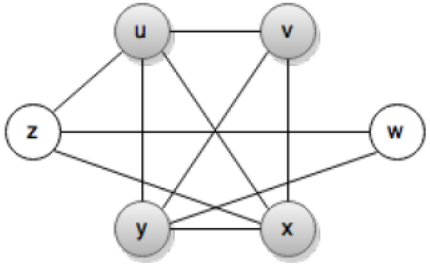
Karp's Reductions

Proving Travelling Salesperson is NP-Complete



Proving Max-Clique is NP-Complete





What does NP-Completeness Signify?

Polynomial Time Approximation Algorithms

Given an NP-Complete Optimization Problem, can we develop an algorithm that works in Polynomial Time and guarantees a worst-case / average-case bound on the solution quality?

A Polynomial Time Approximation Scheme (PTAS) is an algorithm which takes an instance of an optimization problem and a parameter $\varepsilon > 0$ and produces a solution that is within a factor $1 + \varepsilon$ of being optimal (or $1 - \varepsilon$ for maximization problems). For example, for the Euclidean traveling salesman problem, a PTAS would produce a tour with length at most $(1 + \varepsilon)L$, with L being the length of the shortest tour.

The running time of a PTAS is required to be polynomial in the problem size for every fixed ε, but can be different for different ε. Thus an algorithm running in time $O(n^{1/\varepsilon})$ or even $O(n^{\exp(1/\varepsilon)})$ counts as a PTAS.

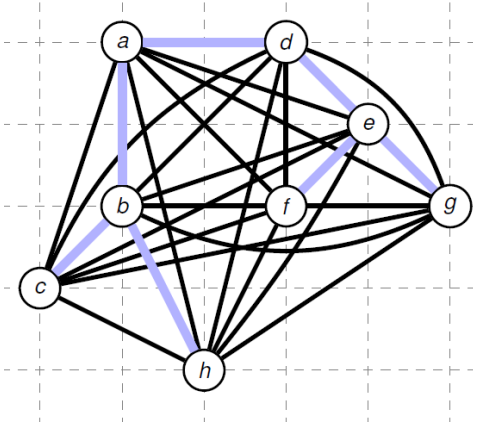
An efficient polynomial-time approximation scheme or EPTAS, in which the running time is required to be $O(n^c)$ for a constant c independent of ε.

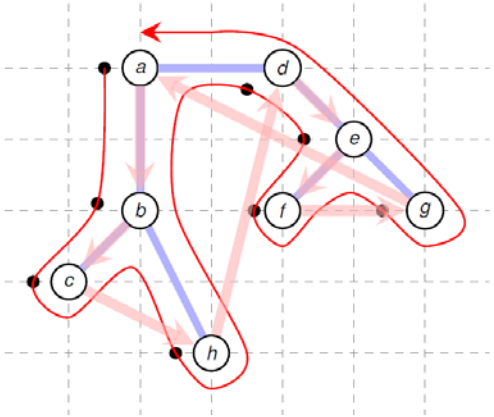
A *fully polynomial-time approximation scheme* or FPTAS, which requires the algorithm to be polynomial in both the problem size n and $1/\varepsilon$.

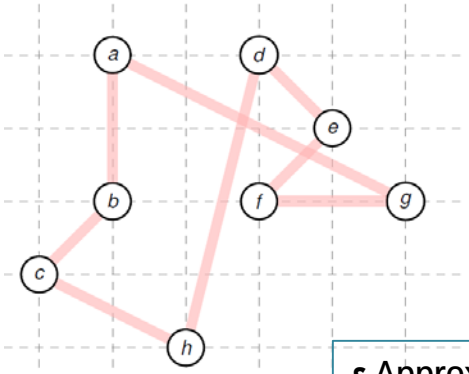
2-Approximation Algorithm for Euclidean TSP

APPROX-TSP-TOUR(G, c)

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







ε Approximation for General TSP is NP-Hard

2-Approximation Algorithm for Knapsack

Greedy Algorithm Redux

1. Sort items in non-increasing order of $\frac{P_i}{S_i}$.
2. Greedily add items until we hit an item a_i that is too big. $\left(\sum_{k=1}^{i} s_i > B\right)$
3. Pick the better of $\{a_1, a_2, ..., a_{i-1}\}$ and $\{a_i\}$.

(1+ε) -Approximation Algorithm for Knapsack

FPTAS for knapsack

1. For some given error parameter $\epsilon > 0$, set $k = \left\lfloor \frac{n}{\epsilon} \right\rfloor$
2. For every item a_i, define $\widehat{p}_i = \left\lfloor \frac{p_i}{p_{max}} k \right\rfloor$
3. Run a dynamic programming algorithm (using $\{\widehat{p}_1, \widehat{p}_2, ..., \widehat{p}_n\}$ as our profit inputs) to get some optimal set $\widehat{S}$.
4. Output $\widehat{S}$.

Thank you

Any Questions?