DESIGN AND ANALYSIS OF ALGORITHMS (DAA)

Unit-5

Dr. G. S. Naveen Kumar,

Dean, Quality

Associate Professor, CSE

Syllabus

Branch and Bound- General Method, applications-0/1 Knapsack problem, LC Branch and Bound solution, FIFO Branch and Bound solution, Traveling sales person problem.

NP-Hard and NP-Complete problems- Basic concepts, Non-deterministic algorithms, NP - Hard and NP-Complete classes, Cook's theorem.

Branch and bound

What is Branch and bound?

- Branch and bound is one of the techniques used for problem solving.
- It is similar to the backtracking since it also uses the state space tree.
- It is used for solving the optimization problems and minimization problems.
- If we have given a maximization problem then we can convert
- it using the Branch and bound technique by simply converting the problem into a maximization problem.

Key points

- The B&B strategy is very similar to backtracking in that a state space tree is used to solve a problem.
- The differences are that the B&B method
- Does not limit us to any particular way of traversing the tree.
- It is used only for optimization problem
- It is applicable to a wide variety of discrete combinatorial problem.
- B&B is rather general optimization technique that applies where the greedy method & dynamic programming fail.
- It is much slower, indeed (truly), it often (rapidly) leads to exponential time complexities in the worst case.

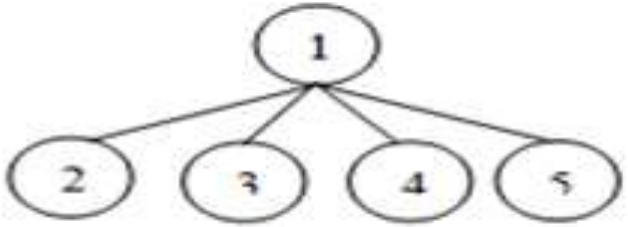
Key points

- The term B&B refers to all state space search methods in which all children of the "E-node" are generated before any other "live node" can become the "E-node"

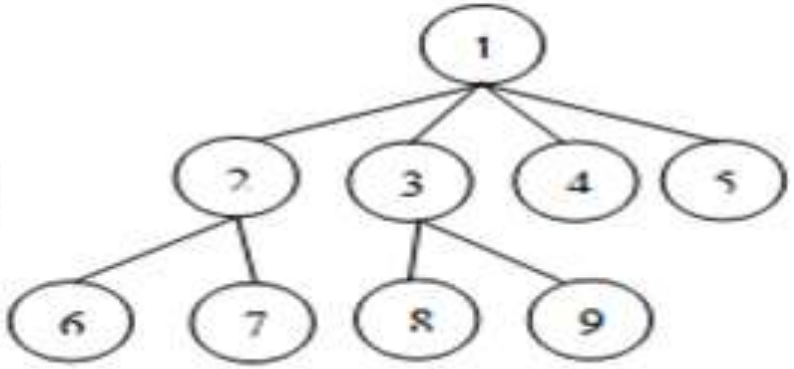
✓ **Live node→** is a node that has been generated but whose children have not yet been generated.

✓ **E-node→** is a live node whose children are currently being explored.

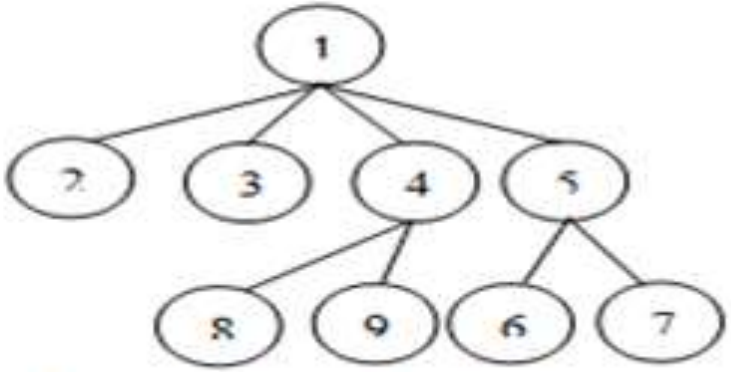
✓ **Dead node→** is a generated node that is not to be expanded or explored any further. All children of a dead node have already been expanded.



Live Node: 2, 3, 4, and 5



FIFO Branch & Bound (BFS)
Children of E-node are inserted in a queue.



LIFO Branch & Bound (D-Search)
Children of E-node are inserted in a stack.

Key points

- Two graph search strategies, BFS & D-search (DFS) in which the exploration of a new node cannot begin until the node currently being explored is fully explored.
- Both BFS & D-search (DFS) generalized to B&B strategies.
- ✓ **BFS→**like state space search will be called FIFO (First In First Out) search as the list of live nodes is "First-in-first-out" list (or queue).
- ✓ **D-search (DFS)→** Like state space search will be called LIFO (Last In First Out) search as the list of live nodes is a "last-in-first-out" list (or stack).
- In backtracking, bounding function are used to help avoid the generation of sub-trees that do not contain an answer node.

General Method

- In branch and bound method a state space tree is built and all the children of E nodes (a live node whose children are currently being generated) are generated before any other node can become a live node.
- For exploring new nodes either a BFS or D-search technique can be used.
- In Branch and bound technique, BFS-like state space search will be called FIFO (First In First Out) search. This is because the list of live node is First In First Out list(queue). On the other hand the D-search like state space search will be called LIFO search because the list of live node is Last in First out list(stack).
- In this method a space tree of possible solutions is generated. Then partitioning (called as branching) is done at each node of the tree. We compute lower bound and upper bound at each node. This computation leads to selection of answer node.
- Bounding functions are used to avoid the generation of subtrees that do not contain an answer node.

Applications

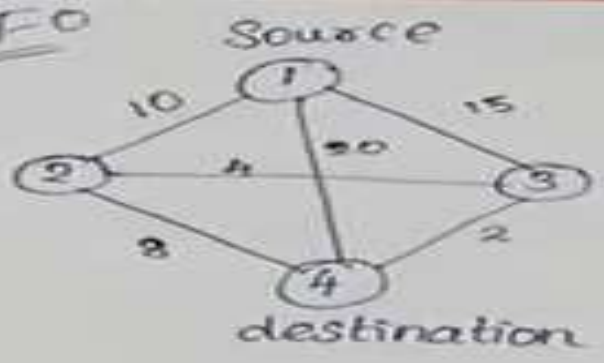
1. Integer programming
2. Nonlinear programming
3. Travelling salesman problem
4. Quadratic assignment problem (QAP)
5. Maximum satisfiability problem (MAX-SAT)
6. Nearest neighbor search
7. Flow shop scheduling
8. Cutting stock problem
9. Computational phylogenetics
10. Set inversion
11. Parameter estimation
12. 0/1 knapsack problem
13. Set cover problem
14. Feature selection in machine learning
15. Structured prediction in computer vision
16. Arc routing problem, including Chinese Postman problem

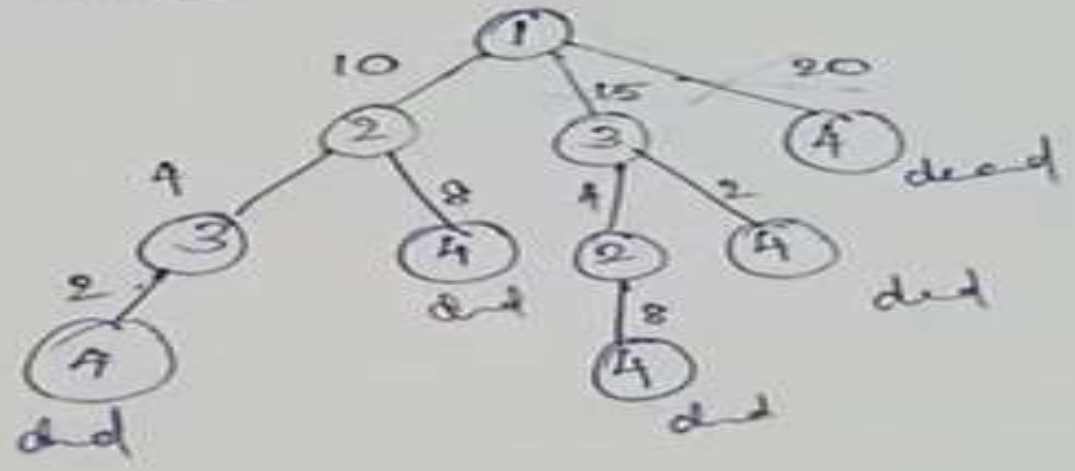
Parameter	Backtracking	Branch and Bound
Approach	Backtracking is used to find all possible solutions available to a problem. When it realises that it has made a bad choice, it undoes the last choice by backing it up. It searches the state space tree until it has found a solution for the problem.	Branch-and-Bound is used to solve optimisation problems. When it realises that it already has a better optimal solution that the pre-solution leads to, it abandons that pre-solution. It completely searches the state space tree to get optimal solution.
Traversal	Backtracking traverses the state space tree by DFS(Depth First Search) manner.	Branch-and-Bound traverse the tree in any manner, DFS or BFS.
Function	Backtracking involves feasibility function.	Branch-and-Bound involves a bounding function.
Problems	Backtracking is used for solving Decision Problem.	Branch-and-Bound is used for solving Optimisation Problem.
Searching	In backtracking, the state space tree is searched until the solution is obtained.	In Branch-and-Bound as the optimum solution may be present any where in the state space tree, so the tree need to be searched completely.
Efficiency	Backtracking is more efficient.	Branch-and-Bound is less efficient.
Applications	Useful in solving N-Queen Problem, Sum of subset.	Useful in solving Knapsack Problem, Travelling Salesman Problem.
Solve	Backtracking can solve almost any problem. (chess, sudoku, etc).	Branch-and-Bound can not solve almost any problem.

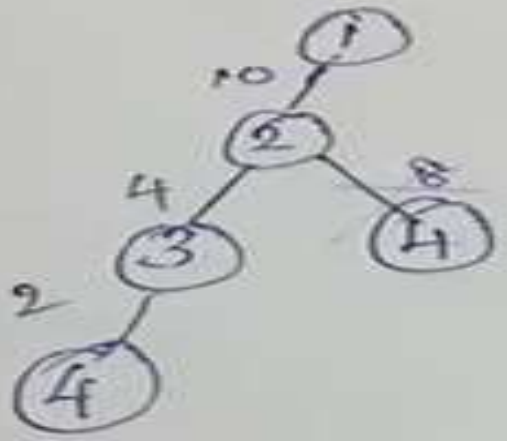
There are 3-types of search strategies in branch and bound

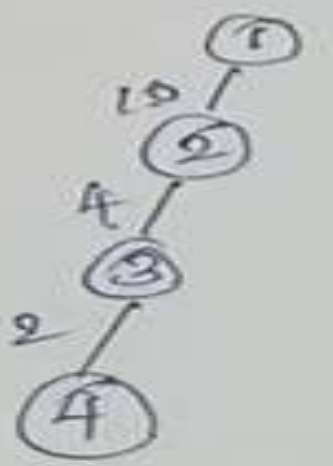
1. FIFO (First In First Out) search
2. LIFO (Last In First Out) search
3. LC (Least Count) search

FIFO


Source
10
15
20
4
8
2
destination


10
15
20
4
8
2
dead
dead
dead
dead


10
4
8
2


10
4
2

= 16

• FIFO (First-In-First-Out) Search

```
Initialize queue Q
Let v be the root of T
Let the current value of v be 'best'
Insert v into Q
While (Q is not empty) do
    remove node v
    for each child u of v do
        if bound(u) is better than best then
            insert u onto Q
            update the best value as value(u)
```

LIFO Search - Branch & Bound

Depth First Search (DFS) with Stack based B&B is called LIFO B&B.

In LIFO Search the children of the root node are generated in the first iteration

In the next step, children of the first child is generated.

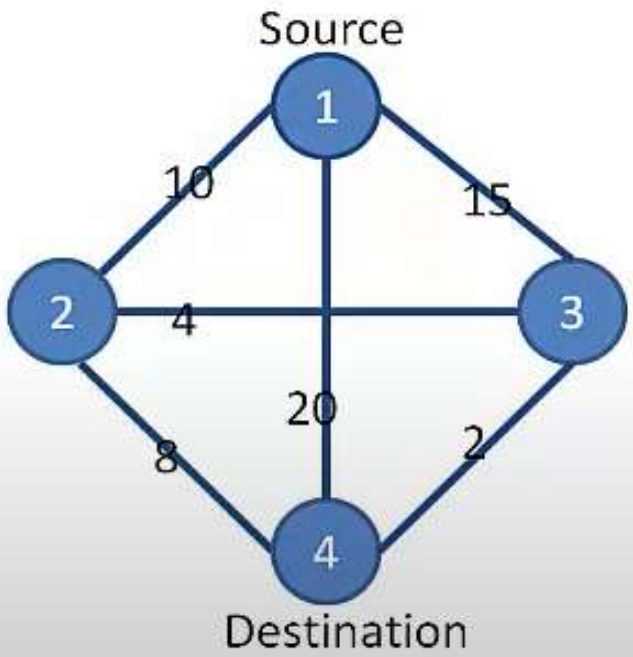
If the children not already killed by bounding function then put it in the Stack. Then the children of second child is explored.

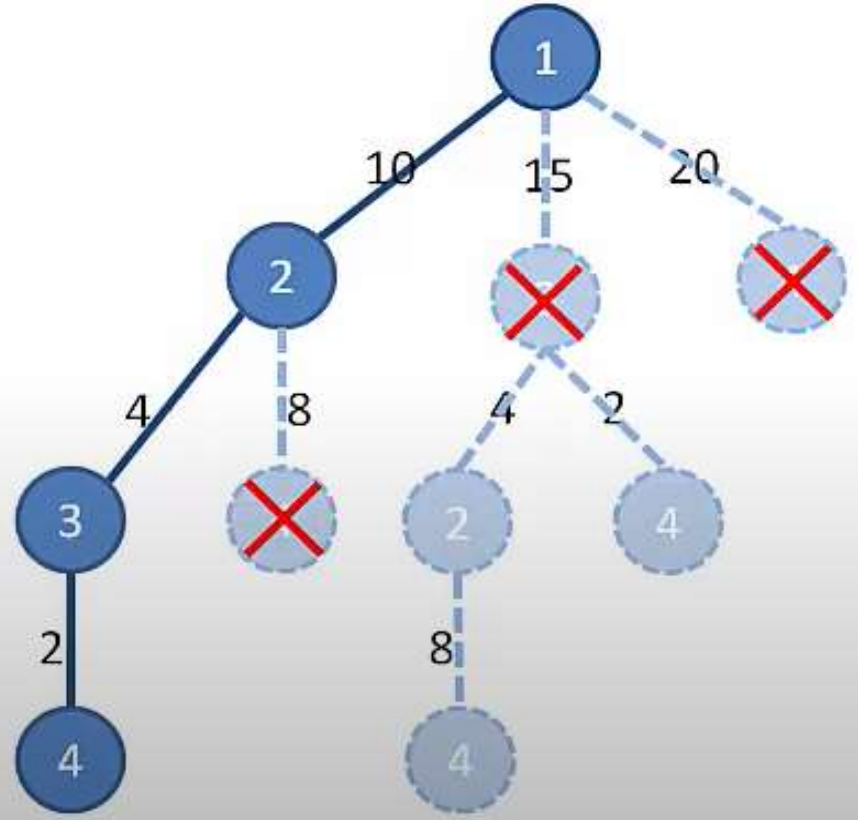
Put all the children in the stack except for those children which are killed.

• LIFO(Last-In-Last-Out) Search

- DFS with stack based Branch & Bound is called LIFO Branch & Bound.
- In LIFO search the children of the root node are generated in the first iteration.
- In the next step, children of the first child is generated.
- If the children not already killed by bounding function then put it in the stack. Then the children of second child is explored.
- Put all the children in the stack except for those children which are killed.

- LIFO(Last-In-Last-Out) Search





• LC(Least Cost Search)

- The ideal way to assign ranks would be on the basis of additional computational effort needed to reach the answer node from the live node.
- For any node x, this cost could be :
  - The number of nodes in the subtree x that needed to be generated before an answer node is generated.
  - The number of levels the nearest answer node is from 'x'.

• LC(Least Cost Search)

- The ranking function is

$$C(x)=f(x)+g(x)$$

Where C(x) is the cost of x.

f(x) is the cost of reaching x from root and is non-decreasing.

g(x) is an estimate of the additional effort needed to reach an answer node x.

To select the next E-node would always choose for its next E-node a live node with least C(x).

Knapsack Problem - Branch & Bound

S.no	Weight	Value	Value / weight
1	4	$ 40	
2	7	$ 42	
3	5	$ 25	
4	3	$ 12	

Total Weight / Capacity $W = 10$

Soln

Formula : $ub = v + (W - w)\left(v_{i+1} / w_{i+1}\right)$

Node 0 $i=0,\ w=0,\ v=0$

$ub = v + (W - w)(v_{i+1} \mid w_{i+1})$

$= 0 + (10-0)(v_1 \mid w_1)$

$= 0 + (10)(10)$

$\boxed{ub = 100}$

Node 1 $i=1,\ w=4,\ v=40$

$ub = 40 + (10-4)(v_2 \mid w_2)$

$= 40 + (6)(6)$

$= 40 + 36$

$\boxed{ub = 76}$

Node 3 $i=2,\ w=4,\ v=40$

$ub = 40 + (10-4)(v_3 \mid w_3)$

$= 40 + (6)(5)$

$= 40 + 30$

$\boxed{ub = 70}$

Node 4 $i=3,\ w=9,\ v=65$

$ub = 65 + (10-9)(v_4 \mid w_4)$

$= 65 + (1)(A)$

$\boxed{ub = 69}$

Node 2 $i=1,\ w=0,\ v=0$

$ub = V + (W - w)(V_{i+1} \mid w_{i+1})$

$= 0 + (10-0)(V_2 \mid w_2)$

$= 0 + (10)(6)$

$= 60$

Node 4 $i=2,\ w=4,\ v=40$

$ub = 40 + (10-4)(v_3 \mid w_3)$

$= 40 + (6)(5)$

$= 40 + 30$

$\boxed{ub = 70}$

Node 5 $i=3,\ w=9,\ v=65$

$ub = 65 + (10-9)(v_4 \mid w_4)$

$= 65 + (1)(A)$

$\boxed{ub = 69}$

Node 7 i = 3, w = 4, v = 40

ub = 40 + (6)(4)

= 40 + 24

ub = 64

Node 8 i = 4, w = 7, V = 52

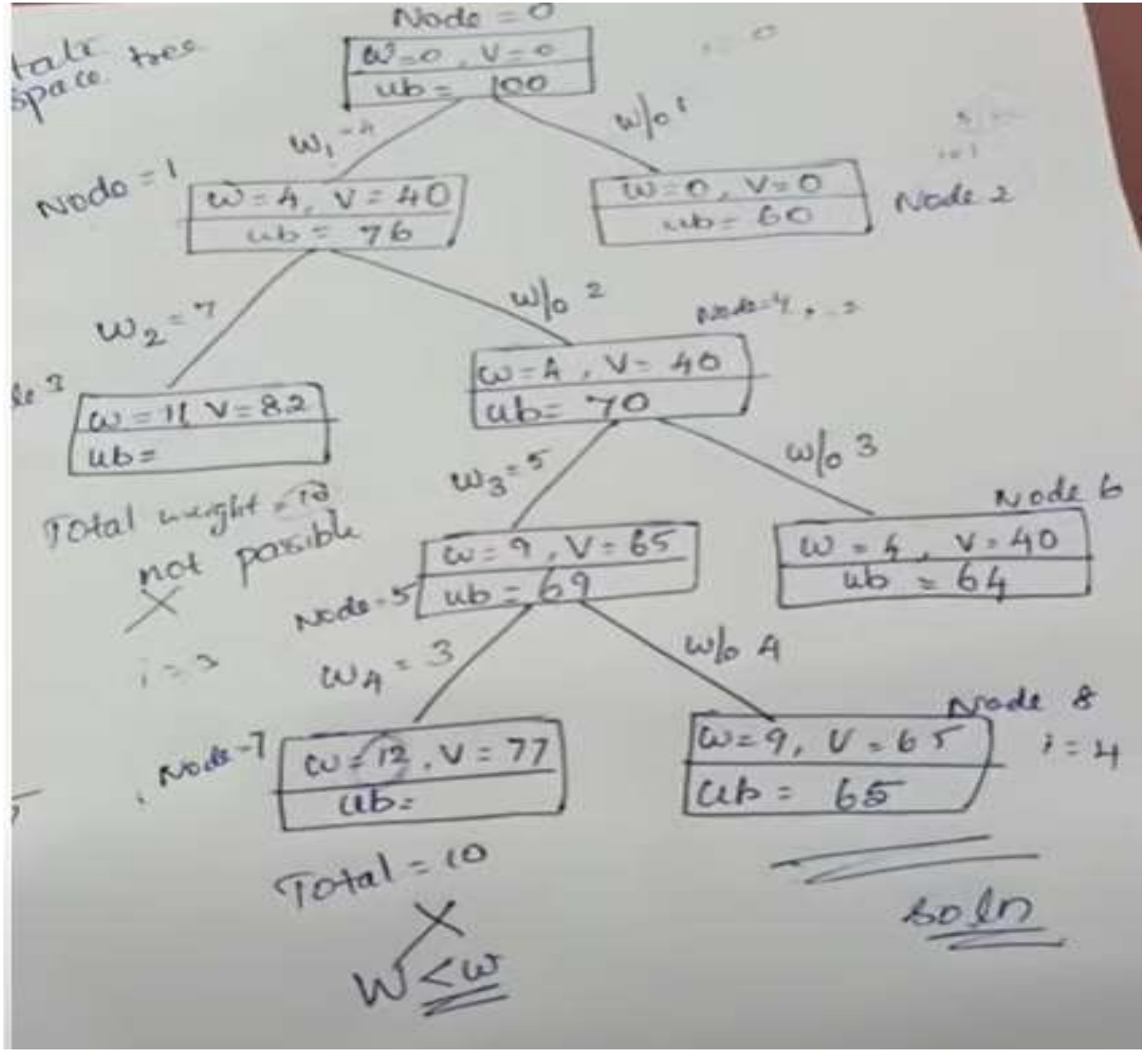
ub = 52 + (3)(0) = 52

Node 9 i = 4, w = 4, V = 40

ub = 40 + (6)(0) = 40

Node 10 w = 0, V = 0, i = 1

ub = 0 + (10)(6) = 60



W1	w2	w3	w4
1	0	1	0

Knapsack Problem - Branch & Bound

S.no	Weight	Value	Value / weight
1	4	$40	10
2	7	$42	6
3	5	$25	5
4	3	$12	4

Total Weight / Capacity $W = 10$

Soln

Formula : $ub = v + (W - w)(v_{i+1} / w_{i+1})$

$4 + 5 = 9$

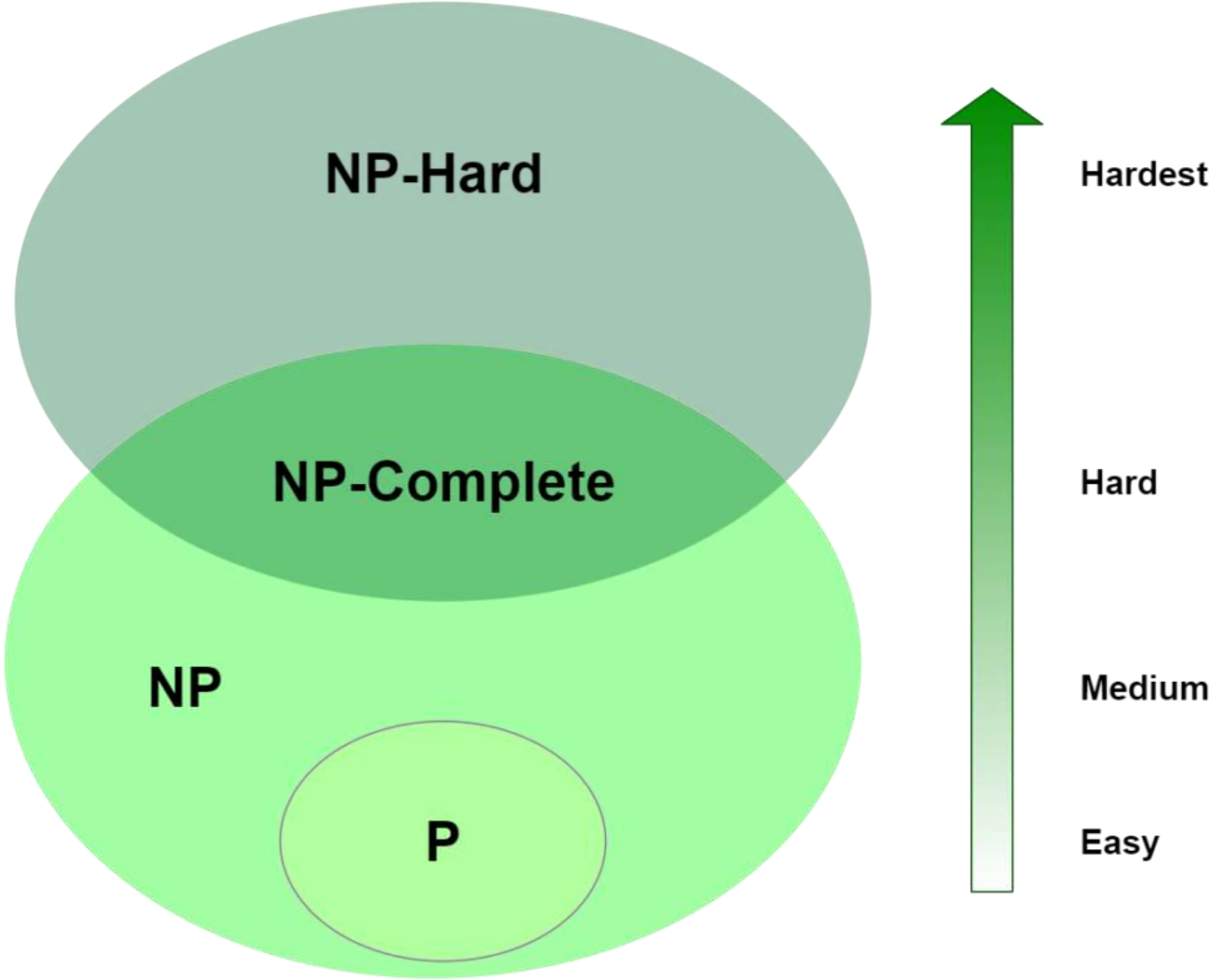
$40 + 25 = 65$

NP hard and NP Complete Problems

In computational complexity theory,

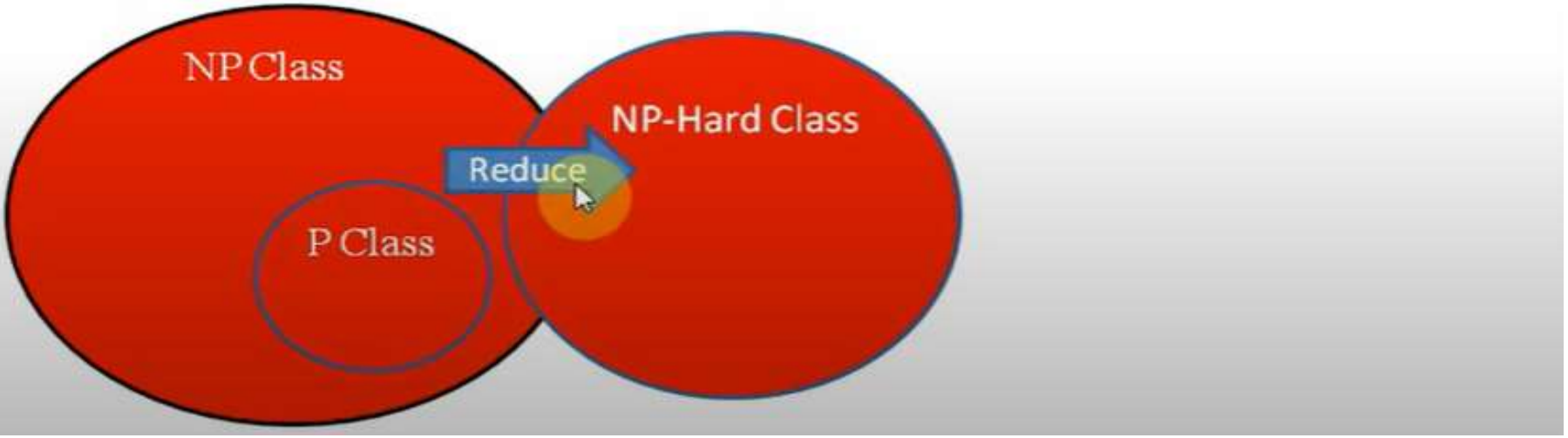
- NP-hardness (non-deterministic polynomial-time hardness) is the defining property of a class of problems that are informally "at least as hard as the hardest problems in NP".
- A simple example of an NP-hard problem is the subset sum problem.

NP hard and NP Complete Problems



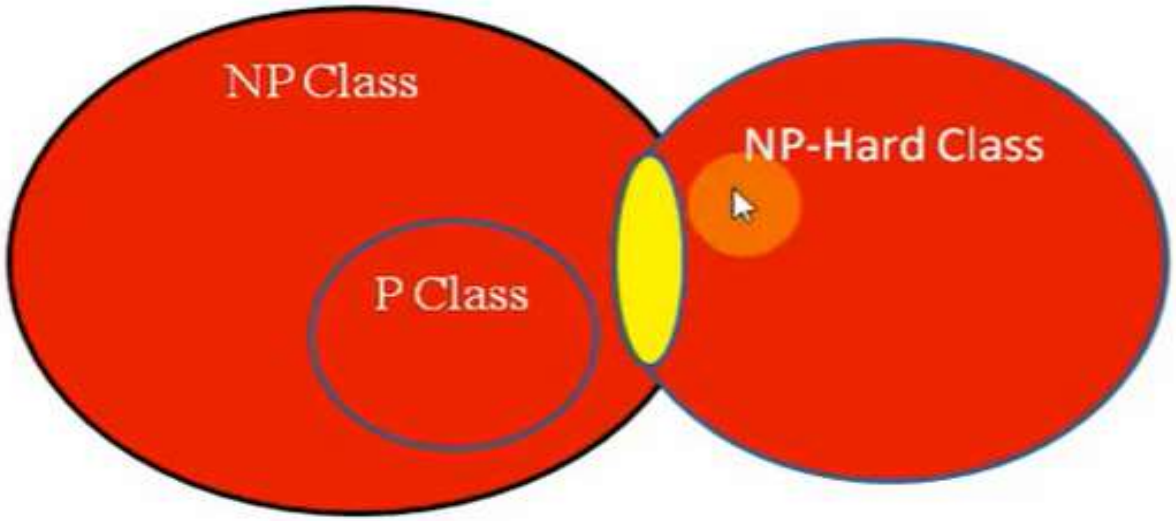
NP-Hard Problem:

- Every problem in NP class can be reduced into other set using polynomial time, then its called as NP-Hard problem.



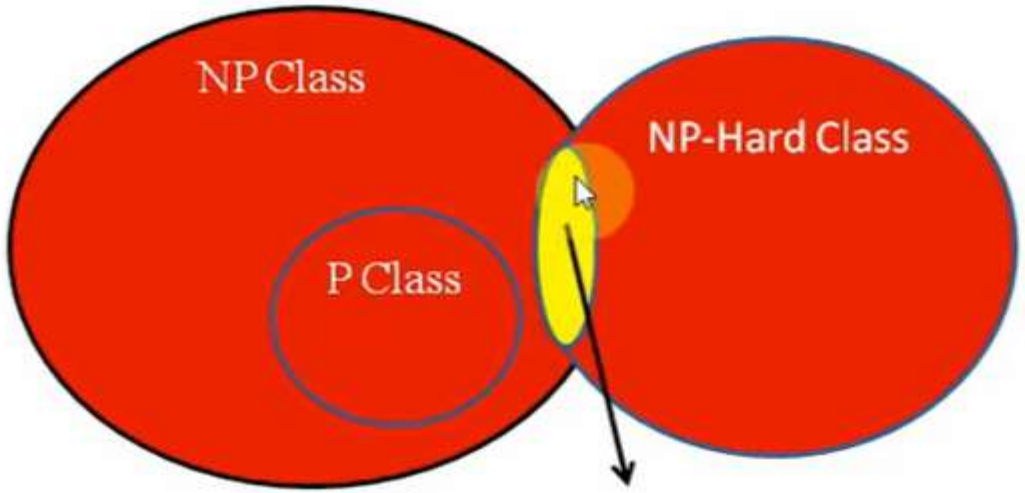
NP-Complete Problem:

- The group of problems which are both in NP and NP-hard are known as NP-Complete problem.
- All NP-Complete problems are NP-Hard but not all NP-Hard problems are not NP-Complete problem.



NP-Complete Problem:

- The group of problems which are both in NP and NP-hard are known as NP-Complete problem.
- All NP-Complete problems are NP-Hard but not all NP-Hard problems are not NP-Complete problem.



NP-Complete Problem