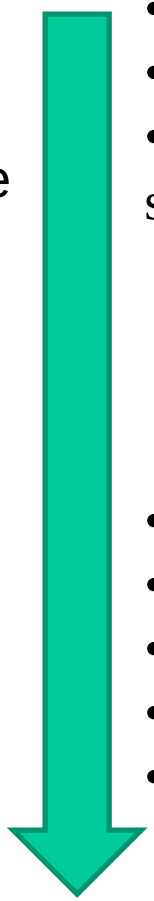
Blind Control strategy

MCA II Semester

Problem solving

- Simple
  - Structured, well defined procedure
  - certainty
- Complex
  - unstructured and not well defined procedure
  - uncertainty



•Roots of quadratic equation
•Calculating income tax
• inventory management system

•Game playing
•Solving puzzle
•Question-answer system
•Medical diagnosis system
•Weather forecasting system

Simple programs and database

Algorithms which will search for a goal in state space

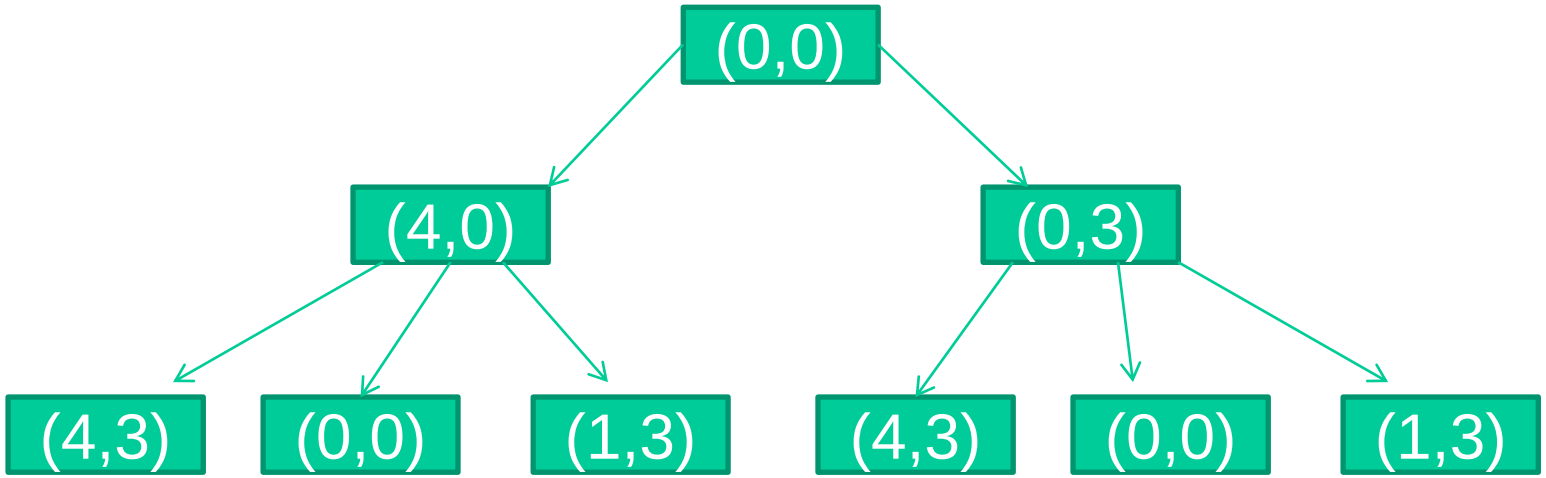
Pattern classification , Neural Network and fuzzy

Control strategies

- Method which specify the order or sequence in which rules or operators are applied to get the goal state
  - Good control strategy must cause motion : operators applied must allow movement in the state space
  - It must be systematic : sequence or order of application of operators must be well defined
- State space is represented by the *tree or graph* where
  - State : is represented by a node
  - Link : by line:represents the change in state after application of an operator
- Types
  - Blind (uninformed search)
  - Heuristic (Informed Search)

State Space representation by tree or graph

- Tree : for water Jug problem



- Depth = 2
- Leaf nodes =6

Search Algorithms

- For solving any problem of complex type we require
  - Formal definition of problem
    - State Space Representation
  - Identifying the starting point
    - Start State
  - When the problem is solved
    - Goal State
  - A search strategy

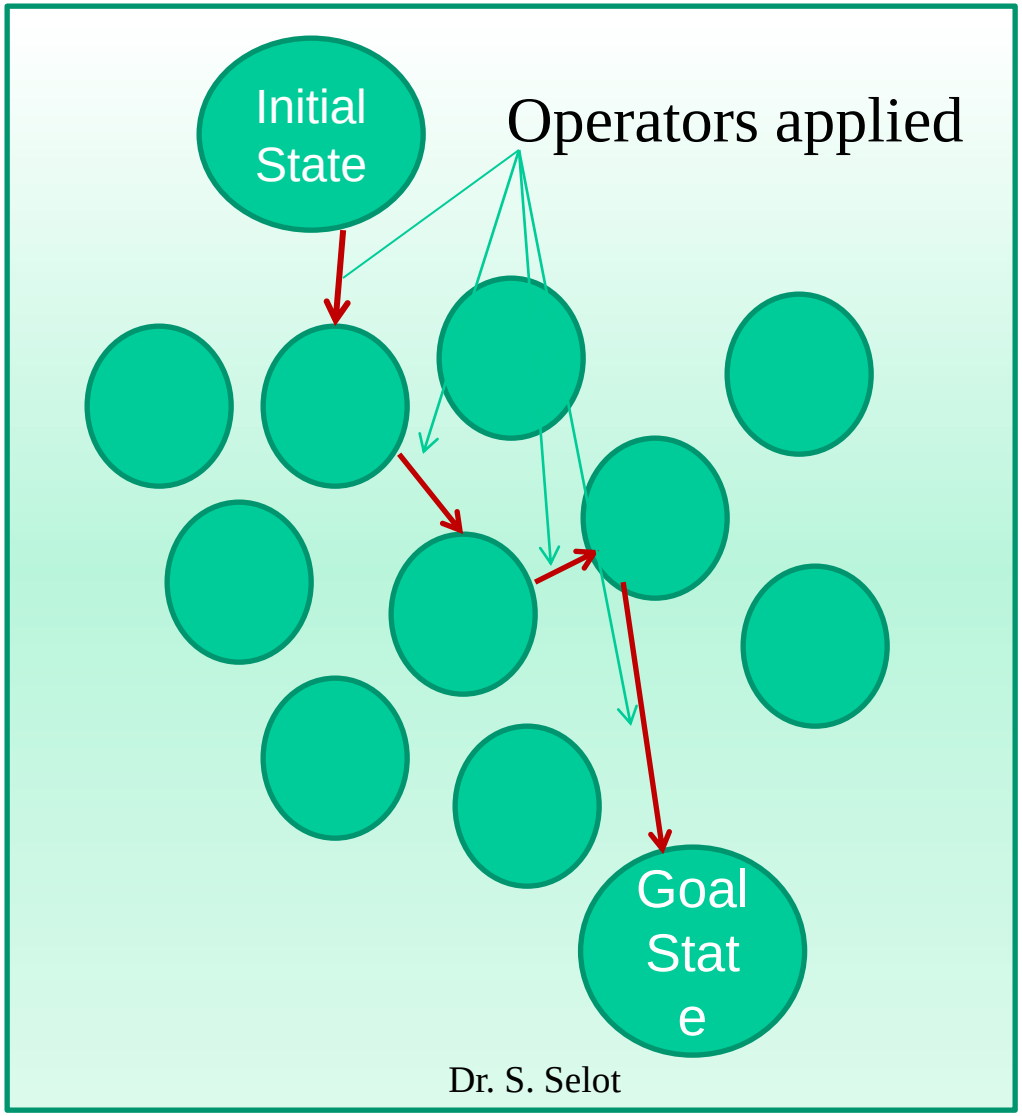
What Are search algorithm?

- They allow us to find a solution to the problem if following are known
  - State space representation
  - Set of operation that will allow movement in state space
  - Start state
  - Goal state
- Search algorithm will find a path from initial state to the goal state using sequence of operations

Search algorithm

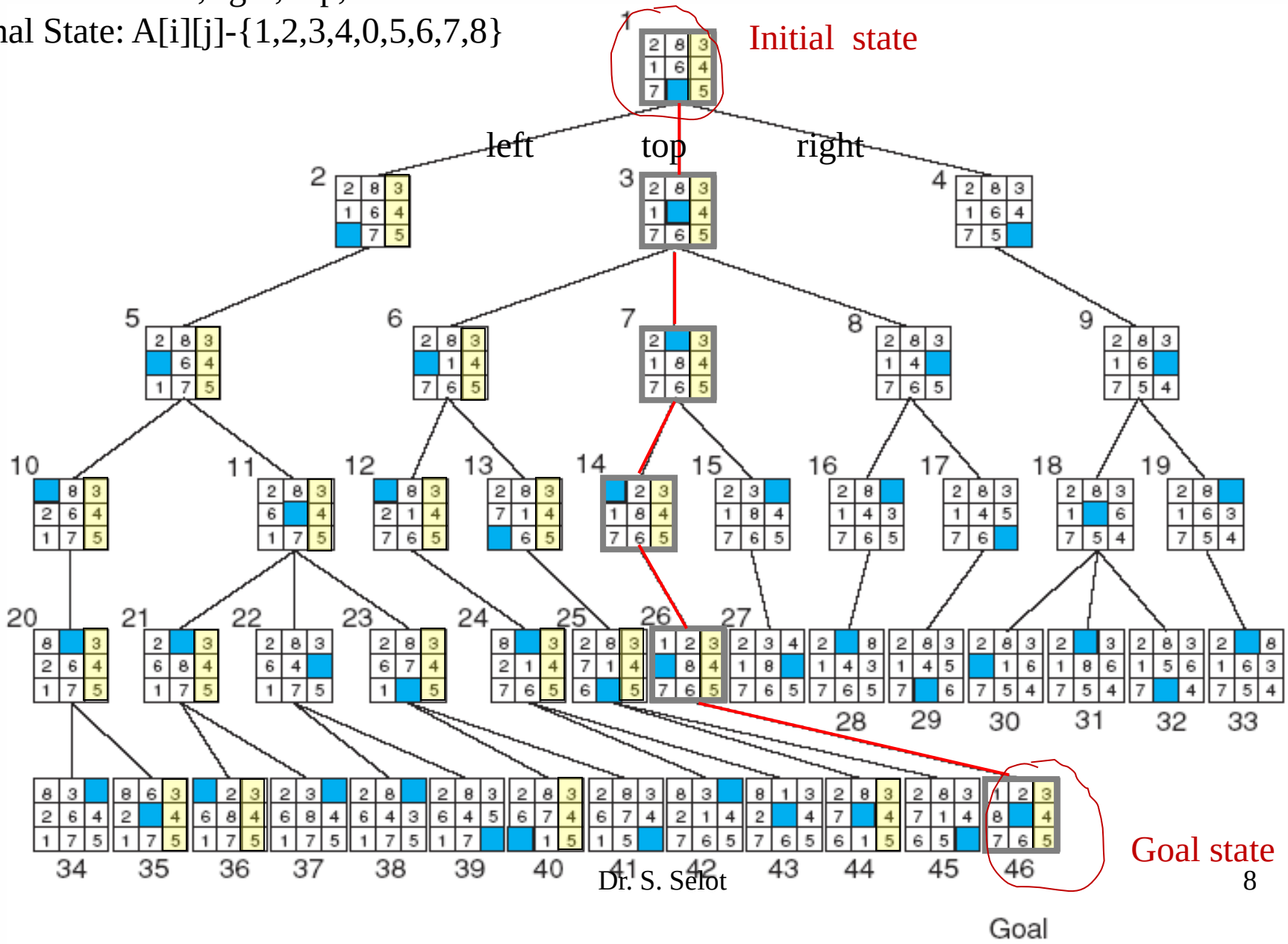
State Space Representation

Intermediate state

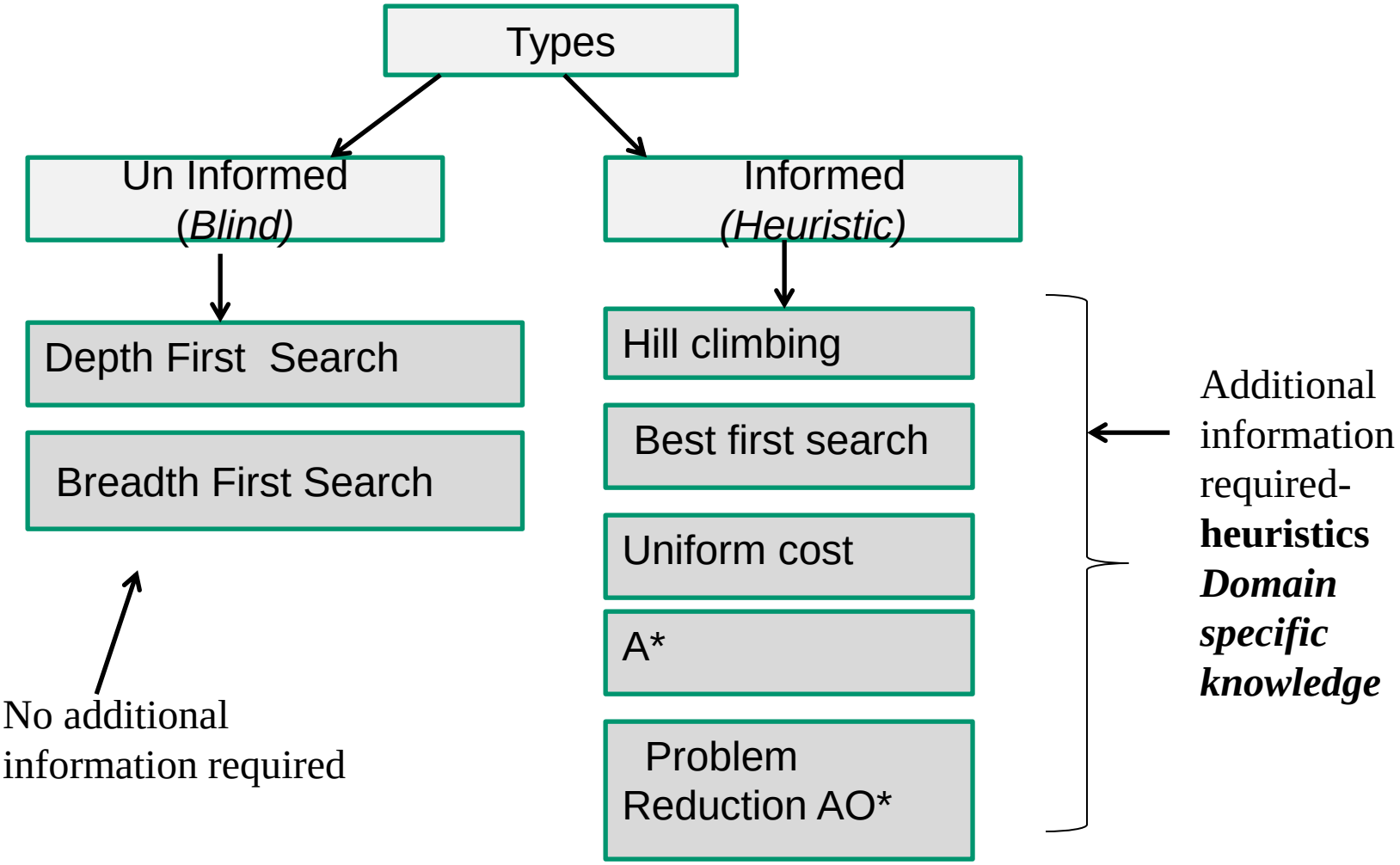


State space : 3X3 matrix
Initial State: A[i][j]-{2,8, 3,1,6,4,7,0,5}
Operation: left,right, top, bottom
Final State: A[i][j]-{1,2,3,4,0,5,6,7,8}

8-puzzle problem

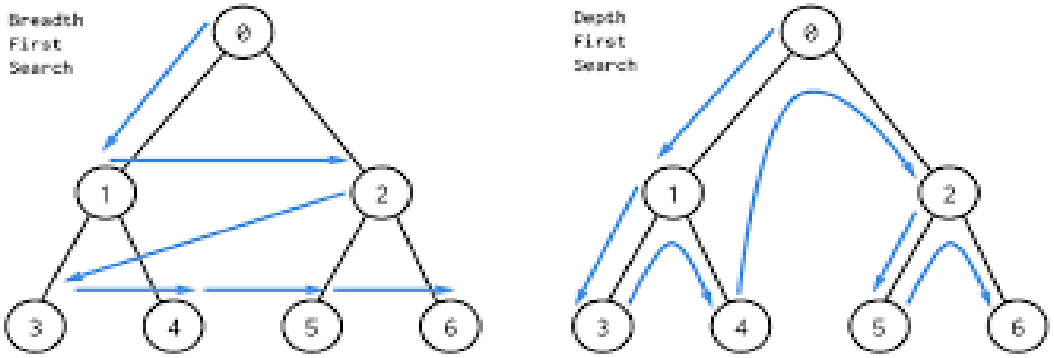


Types of Search Algorithm



DFS and BFS

- State space either tree or graph
- It is navigated either in depth first or breadth first manner till goal state is reached or all state space is exhausted
- DFS used STACK data structure to store the nodes while navigating a, where as BFS uses QUEUE data structure



Algorithm for DFS

- If initial node is goal state , quit and return success
- Else
  - Remove the element E from stack
  - Generate the successor
    - If no successor report failure
  - Call DFS on E as initial state
- If success is returned success else continue

Algorithm for DFS-non recursive

OPEN is stack , S=start state, n= any state, G=Goal state, O=set of operators

1. **Initialize:**
   1. Set OPEN ={S}
2. **Fail:**
   1. if OPEN={ } terminate with failure
3. **Select**:
   1. Select a state n from OPEN
4. **Terminate:**
   1. If n € G terminate with success
5. **Expand:**
   1. Generate successor of n using O and insert them into OPEN
6. **Loop**
   – Go to step 2

Algorithm for DFS-recursive

- [Step 1] If initial node is goal state , quit and return success
- [Step 2] Else do the following till success or failure is reached
  - Generate the successor E of initial state
    - If no successor report failure
  - Call DFS on E as initial state
  - If success is returned success else continue

DFS

- **Advantages**
  - Requires less memory
  - May find solution early, if goal is present in initial branches of search tree
- **Disadvantages**
  - May get trapped for following reason:
    - Single unfruitful path for a long time
      - No limit in depth
      - Depth limited search imposes limit of depth upto which state space tree should be expanded
    - Loops :
      - when new state generated is same as it ancestor.
      - Requires special processing overhead to identify it
  - Backtracking is required if goal is not present in current branch

BFS

OPEN is queue , S=start state, n= any state, G=Goal state, O=set of operators

1. **Initialize:**
   1. Set OPEN ={S}
2. **Fail:**
   1. if OPEN={ } terminate with failure
3. **Select**:
   1. Select a state n from OPEN
4. **Terminate:**
   1. If n € G terminate with success
5. **Expand:**
   1. Generate successor of n using O and insert them into OPEN
6. **Loop**
   – Go to step 2

Algorithm for BFS

- [**Intialize**]
  - Create OPEN as FIFO based list - queue
  - set it to initial state S
- [**loop**]
  - Repeat till goal is found or OPEN list is empty
  - [**Select]** Remove first element N from OPEN queue
  - [**Fail**] If OPEN is empty
    - quit
  - For each operator that match with state N do
    - [**Expand]** Apply operator and generate new state
    - [**Terminate**] If New state = goal state quit and return the state
    - Else add new state to end of list

BFS

- Advantages
  - BFS will never be trapped DFS may be trapped in a loop
  - In case of multiple goal BFS is guaranteed to find a solution – **minimal solution** as longer path are never explored until shorter path are examined.
  - On the other hand,DFS may find long path to solution where as shorter path exist in some where in state space.
- Disadvantage
  - Requires more space as compared to DFS
  - Takes more time if goal is present at greater depth of search tree

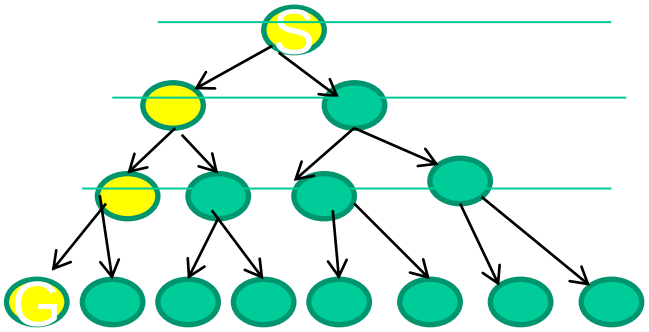
Analysis of search algorithm

- Effectiveness of any search strategy is measured in terms of
  - *Completeness*: Means that an algorithm guarantees a solution if it exists
  - *Time Complexity* Time required by an algorithm to find the solution
  - *Space Complexity:* Space required by an algorithm to find a solution
  - *Optimality*: An algorithm is optimal if it finds the highest quality solution when there exist several different solutions for the problem

BFS and DFS

Head	BFS	DFS
traversal	Breadthwise	Depthwise
Data structure	Queue	Stack
Memory	More	Less
Best case	When goal is present in initial level of tree or graph	When goal is present in initial branch
Order of execution	$O(b^d)$ b-branching factor d-depth	$O(b^d)$ Best case: O(bd) b-branching factor d-depth
Optimum goal	Yes	No
Trap	No	Gets trapped in dead ends
Order of application of rules or operator	Does not matter	Matters a lot at it traverses branchwise. If goal containing branch is explored first, time to obtain goal is less and vice versa

Time and Space Complexity



Depth 0 No of nodes = 2^0

Depth 1 No of nodes = 2^1

Depth 2 No of nodes = 2^2

Depth 3 No of nodes = 2^3

Time Complexity

Number of comparison for depth 4 = 4

Number of comparison for depth d = d

If b= average branching factor of search tree

For **Best Case** Scenario

Time Complexity ≈ O(d)

Space Complexity is O(bd)

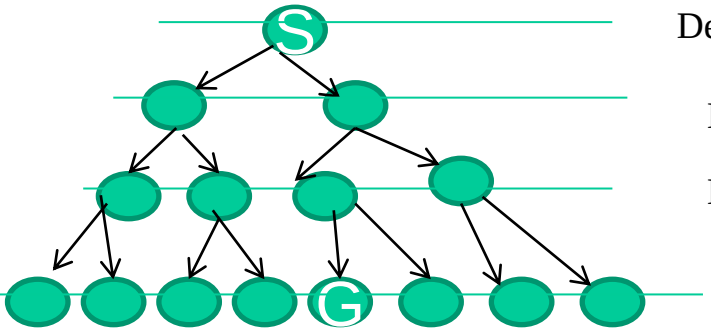
For **Worst Case** Scenario Time and space complexity is same as that of BFS ≈ $O(b^d)$

Iterative deepening

- DFS requires less storage and BFS finds the optimal goal
- Trade off between space and time
- **Solution: iterative deepening**
- It is a general strategy which uses both DFS and BFS to find the best depth
- It uses DFS to do BFS
- Perform DFS repeatedly in increasing depth bound
- Perform DFS at 1^{st} level
- Perform DFS at 2^{nd} level and so on till goal is found
- Time $O(b^d)$ Space: O(bd)

Time and space complexity

- For BFS if number of branches = 2 for each node



Depth 0 No of nodes = 2^0

Depth 1 No of nodes = 2^1

Depth 2 No of nodes = 2^2

Depth 3 No of nodes = 2^3

No of comparison required for goal G is =1+2+4+8

No of comparison required for depth d is

$2^0 + 2^1 + 2^2 \ldots\ldots\ldots\ldots + 2^d = (2^{d-1} - 1)/ (2-1)$

Time complexity $\approx O(2^d)$

If average branching for search tree is =b

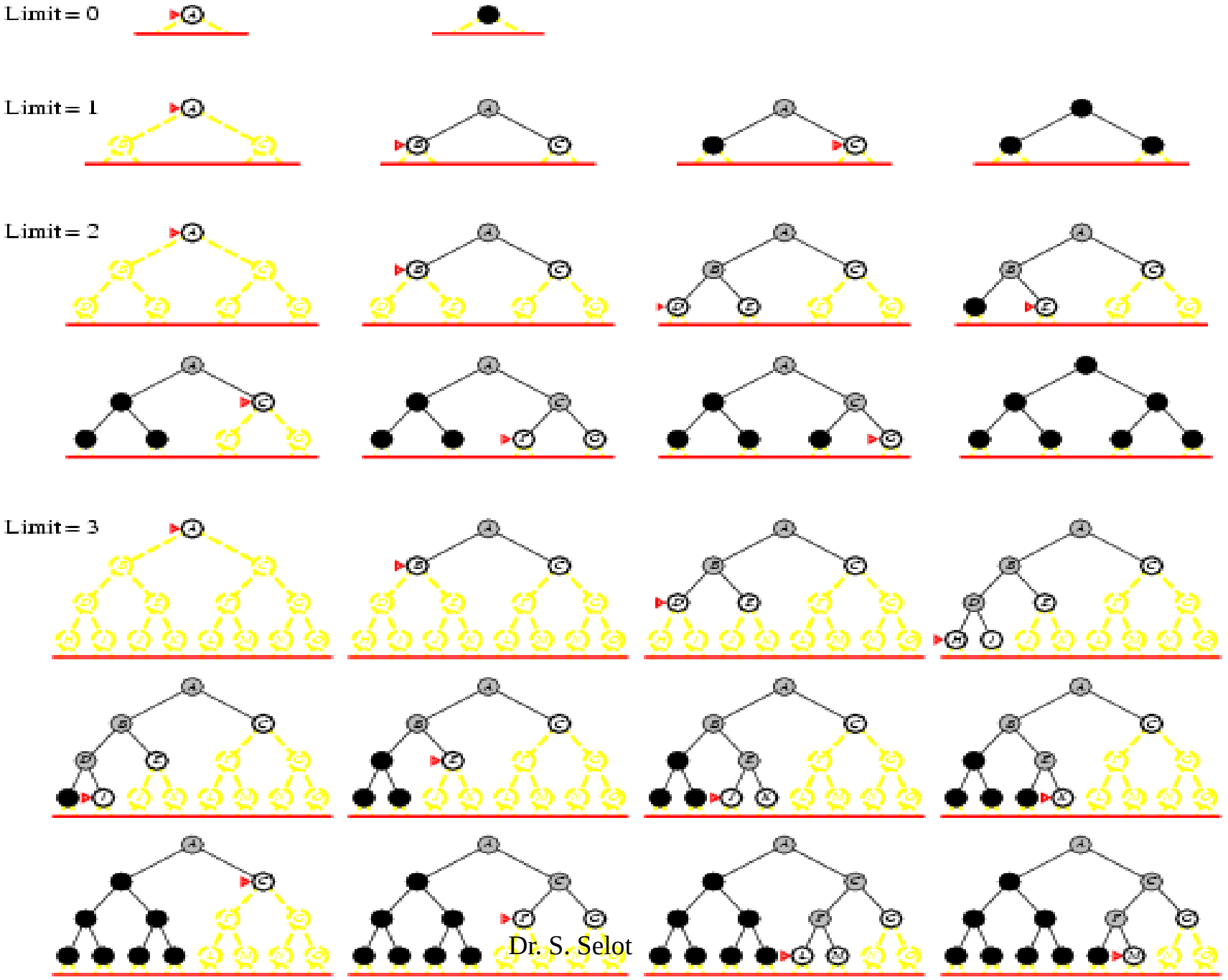
depth at which goal is obtained = d

Time complexity $= b^0 + b^1 + b^2 \ldots. + b^d = (b^{d-1} - 1)/ (b-1) \approx O(b^d)$

Space Complexity = maximum number of nodes in an open list (queue)

= maximum number of nodes at a particular level(d)

$\approx O(b^d)$


Limit = 0
Limit = 1
Limit = 2
Limit = 3

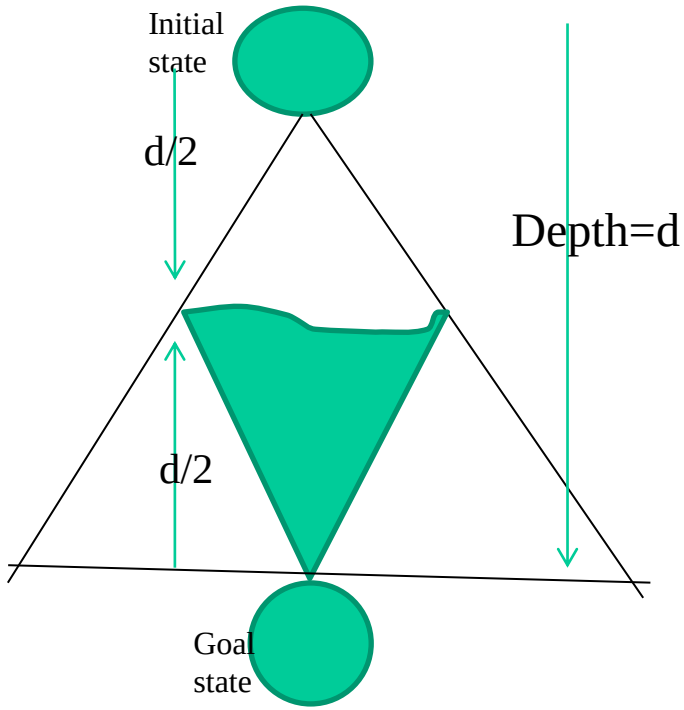
Iterative Deepening

- Nodes at depth d are generated once during the final iteration
- Nodes at depth d-1 are generated twice
- Nodes at depth d-2 are generated thrice and so on
- Total number of nodes generated are
  - $N= b^d + 2b^{d-1} + 3b^{d-2} \ldots\ldots +db$
  - $= b^d \ (1+ 2b^{-1} +3b^{-2} \ldots\ldots$
  - $= b^d \ (1+2x+3x^2 +\ldots\ldots)$ if $x=b^{-1}$
  - $=b^d (1-x)^{-2}$
  - $=O(b^d)$

Bidirectional Search

- Idea : To run two simultaneous search
  - One forward : from initial to goal state
  - Other backward: from goal to initial state
  - To stop when two search meet at middle

Time for entire State Space
$O(b^{d/2}) + O(b^{d/2})$
$= O(b^{d/2})$



Comparison

Search Technique	Time	Space	Solution
DFS	$O(b^d)$	$O(d)$	-
BFS	$O(b^d)$	$O(b^d)$	Optimal
ID	$O(b^d)$	$O(d)$	Optimal
Bi-directional	$O(b^{d/2})$	$O(b^{d/2})$	

All these search techniques are blind and not effective in real life applications

As the size of search increases, combinatorial explosion takes place

There is need of algorithm which takes into account problem based information and finds a faster solution

Example Travelling salesman problem