AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

CHAPTER – 2
PROBLEM STATE SPACE SEARCH & HEURISTIC SEARCH TECHNIQUES

Subject : AI Code : 2180703	Prepared By: Asst. Prof. Twinkal Panchal (CSE Department, ACET)	AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

State Spaces

- One general formulation of intelligent action is in terms of **state space**.
- A **state** contains all of the information necessary to predict the effects of an action and to determine if it is a goal state.

- ➢ The agent has perfect knowledge of the state space and can observe what state it is in
- ➢ The agent has a set of actions that have known deterministic effects;
- ➢ Some states are goal states, the agent wants to reach one of these goal states, and the agent can recognize a goal state;
- ➢ A **solution** is a sequence of actions that will get the agent from its current state to a goal state.

A **state-space problem** consists of

- A set of states;
- A distinguished set of states called the **start states**;
- A set of actions available to the agent in each state;
- An **action function** that, given a state and an action, returns a new state;
- A set of goal states, often specified as a Boolean function, *goal(s)*, that is true when *s* is a goal state; and
- A criterion that specifies the quality of an acceptable solution.

Water jug problem

- one having the capacity to hold 3 gallons of water and the other has the capacity to hold 4 gallons of water. There is no other measuring equipment available and the jugs also do not have any kind of marking on them. So, the agent's task here is to fill the 4-gallon jug with 2 gallons of water by using only these two jugs and no other material. Initially, both our jugs are empty.

Production rules for solving the water jug problem

- Here, let ***x*** denote the 4-gallon jug and ***y*** denote the 3-gallon jug.

S.No.	Initial State	Condition	Final state	Description of action taken
1.	(x,y)	If x<4	(4,y)	Fill the 4 gallon jug completely
2.	(x,y)	if y<3	(x,3)	Fill the 3 gallon jug completely
3.	(x,y)	If x>0	(x-d,y)	Pour some part from the 4 gallon jug
4.	(x,y)	If y>0	(x,y-d)	Pour some part from the 3 gallon jug
5.	(x,y)	If x>0	(0,y)	Empty the 4 gallon jug
6.	(x,y)	If y>0	(x,0)	Empty the 3 gallon jug
7.	(x,y)	If (x+y)<7	(4, y-[4-x])	Pour some water from the 3 gallon jug to fill the four gallon jug
8.	(x,y)	If (x+y)<7	(x-[3-y],y)	Pour some water from the 4 gallon jug to fill the 3 gallon jug.
9.	(x,y)	If (x+y)<4	(x+y,0)	Pour all water from 3 gallon jug to the 4 gallon jug
10.	(x,y)	if (x+y)<3	(0, x+y)	Pour all water from the 4 gallon jug to the 3 gallon jug

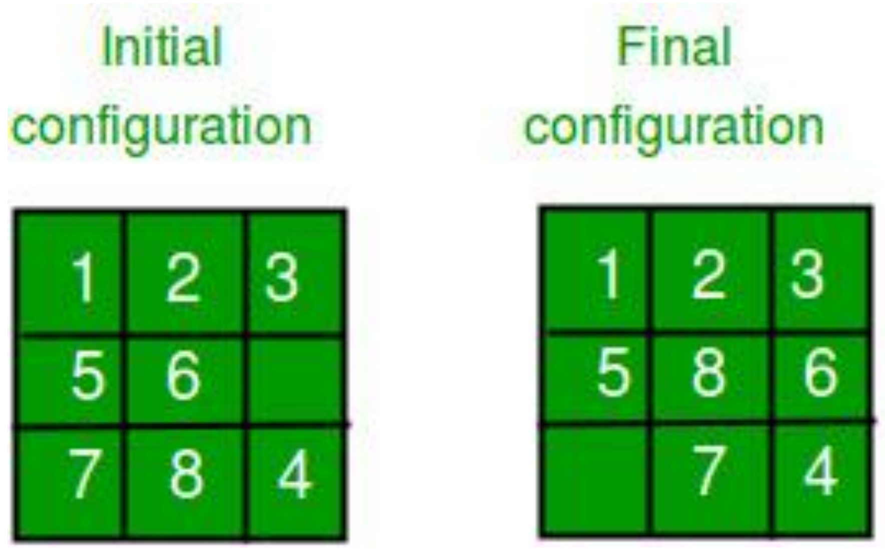
Solution of water jug problem according to the production rules:

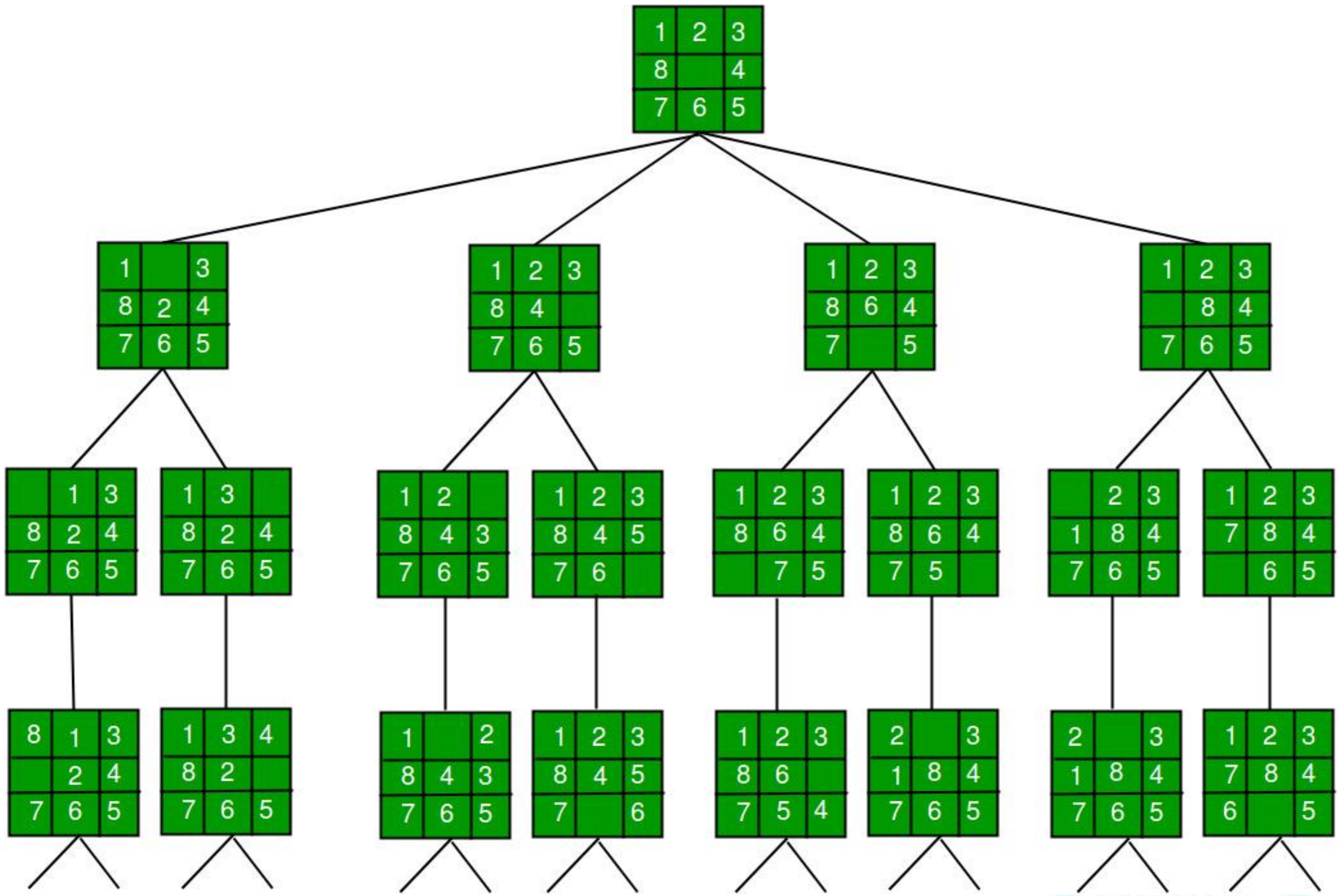
S.No.	4 gallon jug contents	3 gallon jug contents	Rule followed
1.	0 gallon	0 gallon	Initial state
2.	0 gallon	3 gallons	Rule no.2
3.	3 gallons	0 gallon	Rule no. 9
4.	3 gallons	3 gallons	Rule no. 2
5.	4 gallons	2 gallons	Rule no. 7
6.	0 gallon	2 gallons	Rule no. 5
7.	2 gallons	0 gallon	Rule no. 9

8 puzzle Problem

- Given a 3×3 board with 8 tiles (every tile has one number from 1 to 8) and one empty space. The objective is to place the numbers on tiles to match final configuration using the empty space. We can slide four adjacent (left, right, above and below) tiles into the empty space.

For example,





1 2 3 / 8 _ 4 / 7 6 5

Production System and Its Characteristics

1. A set of production rules, which are of the form A->B. Each rule consists of left hand side constituent that represent the current problem state and a right hand side that represent an output state. A rule is applicable if its left hand side matches with the current problem state.

2. A database, which contains all the appropriate information for the particular task. Some part of the database may be permanent while some part of this may pertain only to the solution of the current problem.

AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

3. A control strategy that specifies order in which the rules will be compared to the database of rules and a way of resolving the conflicts that arise when several rules match simultaneously.

4. A rule applier, which checks the capability of rule by matching the content state with the left hand side of the rule and finds the appropriate rule from database of rules.

Features of Production System

1. **Simplicity**

- The structure of each sentence in a production system is unique and uniform as they use “IF-THEN” structure.
- This structure provides simplicity in knowledge representation.

2. **Modularity**

- This means production rule code the knowledge available in discrete pieces. Information can be treated as a collection of independent facts which may be added or deleted from the system with essentially no delete side effects.

3. **Modifiability**

- This means the facility of modifying rules.
- It allows the development of production rules in a skeletal form first and then it is accurate to suit a specific application.

4. **Knowledge intensive**

- knowledge base of production system stores pure knowledge.
- Each production rule is normally written as an English sentence; the problem of semantics is solved by the very structure of the representation.

Disadvantages

1. **Opacity**

- This problem is generated by the combination of production rules. The opacity is generated because of less prioritization of rules. More priority to a rule has the less opacity.

2. **Inefficiency**

- During execution of a program several rules may active. A well devised control strategy reduces this problem.
- As the rules of the production system are large in number and they are hardly written in hierarchical manner, it requires some forms of complex search through all the production rules for each cycle of control program.

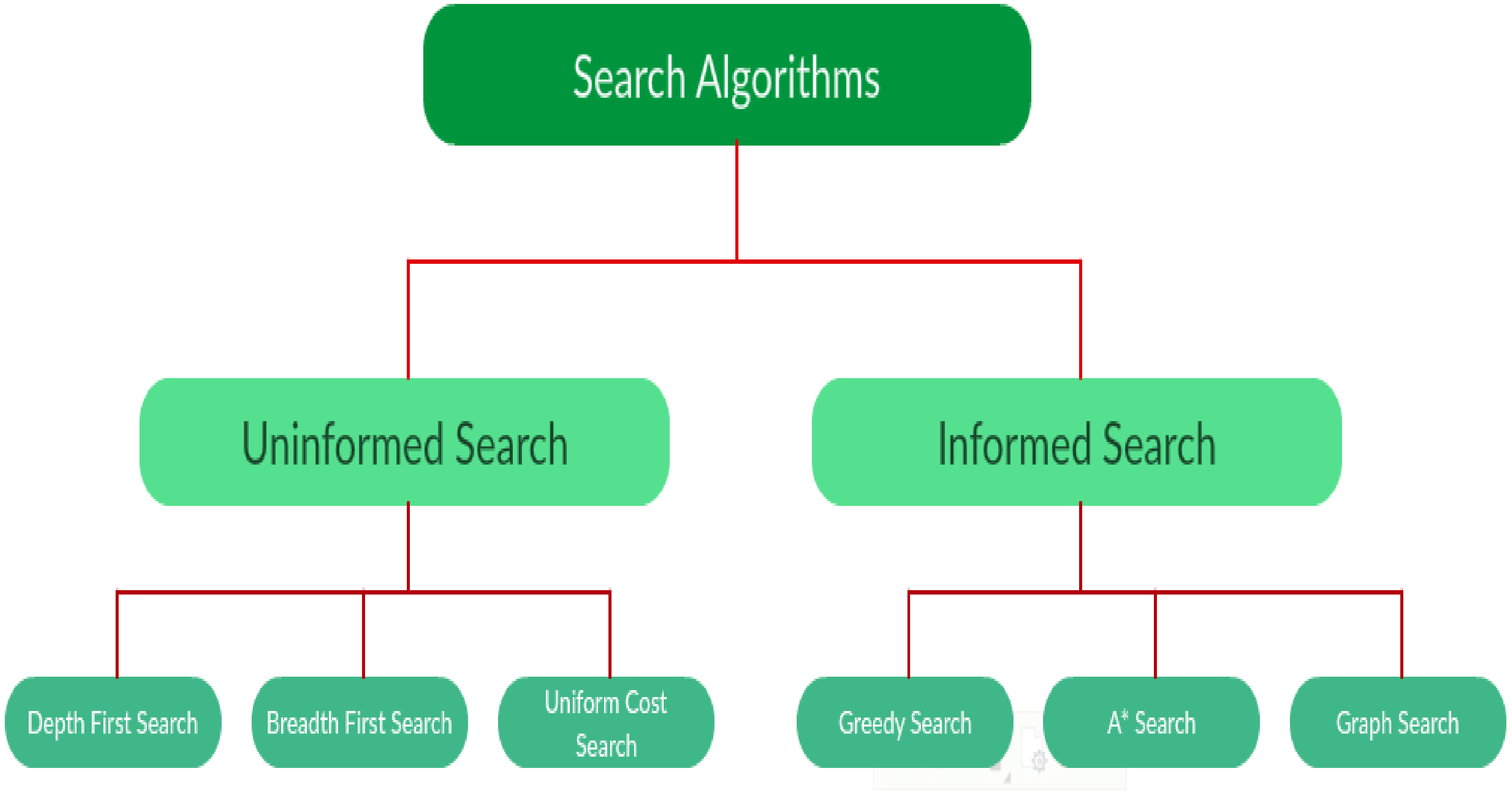
3. **Absence of learning**

- Rule based production systems do not store the result of the problem for future use.
- Hence, it does not exhibit any type of learning capabilities.

4. **Conflict resolution**

- The rules in a production system should not have any type of conflict operations.
- When a new rule is added to a database, it should ensure that it does not have any conflicts with the existing rules.

Types of Search Algorithms



AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

Uninformed Search Algorithms

- The search algorithms in this section have no additional information on the goal node other than the one provided in the problem definition.
- The plans to reach the goal state from the start state differ only by the order and/or length of actions. Uninformed search is also called **Blind search**.

1. Depth First Search
2. Breath First Search
3. Uniform Cost Search

Each of these algorithms will have:

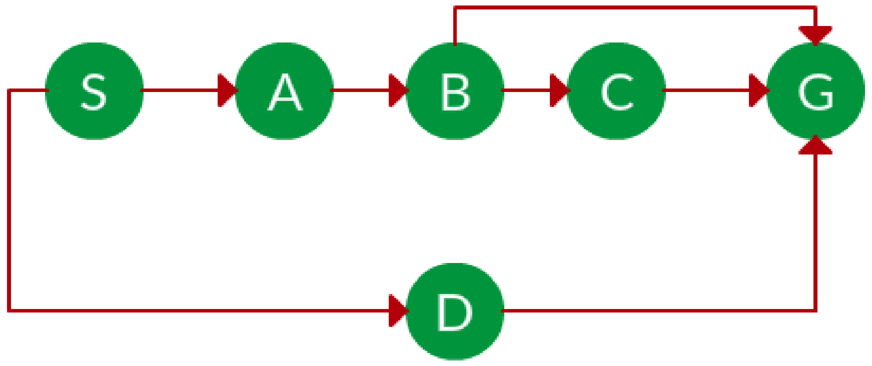
1. A problem **graph,** containing the start node S and the goal node G.
2. A **strategy,** describing the manner in which the graph will be traversed to get to G .
3. A **fringe,** which is a data structure used to store all the possible states (nodes) that you can go from the current states.
4. A **tree,** that results while traversing to the goal node.
5. A solution **plan,** which the sequence of nodes from S to G.

Depth First Search

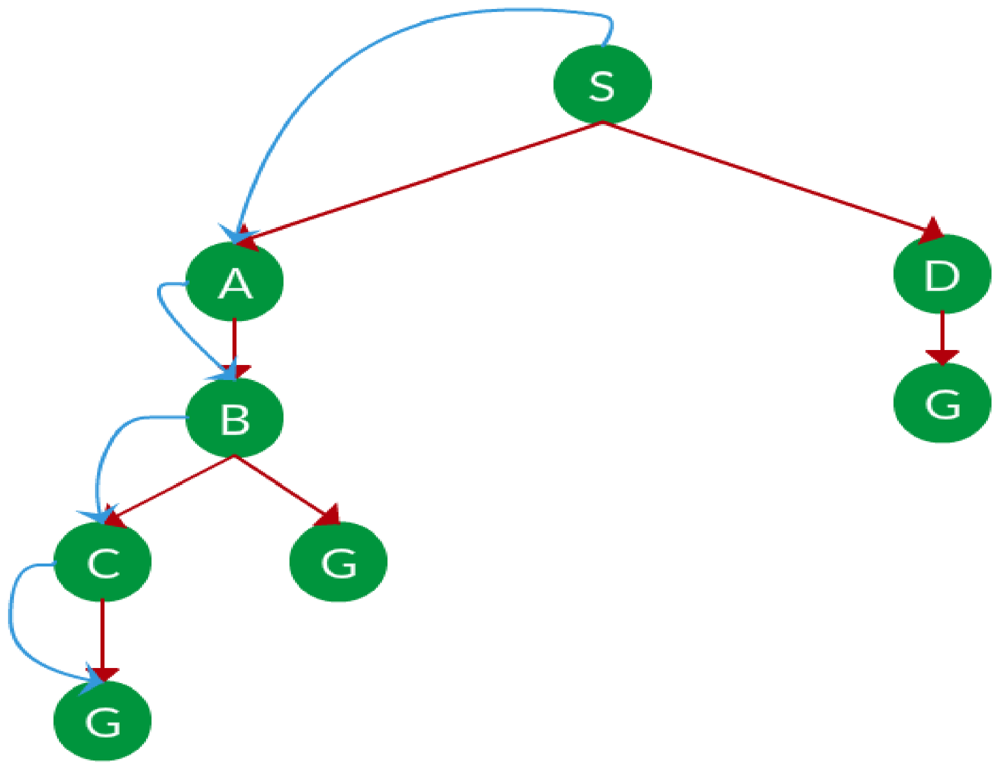
- Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures.
- The algorithm starts at the root node and explores as far as possible along each branch before backtracking.

Example:

- **Question.** Which solution would DFS find to move from node S to node G if run on the graph below?



Solution. The equivalent search tree for the above graph is as follows. As DFS traverses the tree “deepest node first”, it would always pick the deeper branch until it reaches the solution (or it runs out of nodes, and goes to the next branch). The traversal is shown in blue arrows.



Path: S -> A -> B -> C -> G

- Let = the depth of the search tree

  = number of levels of the search tree.
  = number of nodes in level .

Time complexity

- Equivalent to the number of nodes traversed in DFS.

Space complexity

- Equivalent to how large can the fringe get.

Completeness

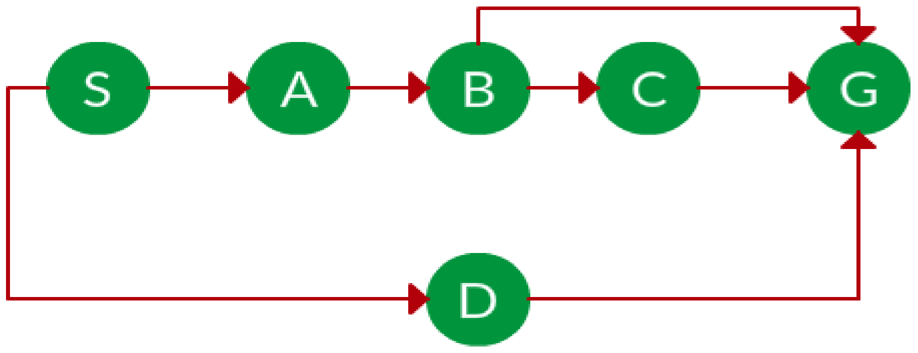
- DFS is complete if the search tree is finite, meaning for a given finite search tree, DFS will come up with a solution if it exists.

Breadth First Search

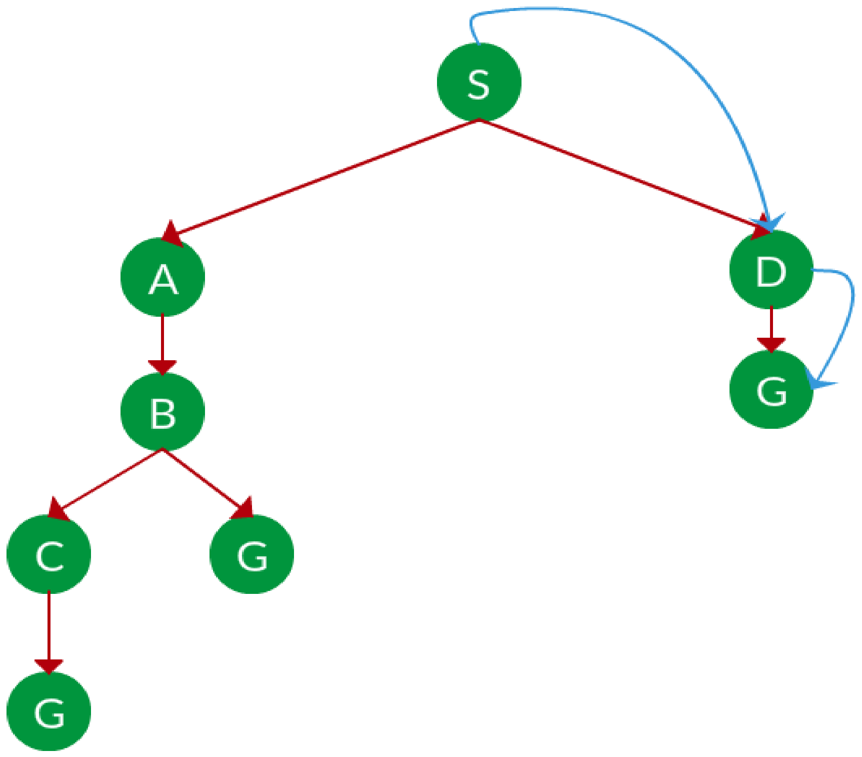
- Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures.
- It starts at the tree root, and explores all of the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level.

Example:

- **Question.** Which solution would BFS find to move from node S to node G if run on the graph below?



Solution. The equivalent search tree for the above graph is as follows. As BFS traverses the tree “shallowest node first”, it would always pick the shallower branch until it reaches the solution (or it runs out of nodes, and goes to the next branch). The traversal is shown in blue arrows.



Path: S -> D -> G

- Let =the depth of the shallowest solution.
  = number of nodes in level .

Time complexity

- Equivalent to the number of nodes traversed in BFS until the shallowest solution.

Space complexity

- Equivalent to how large can the fringe get.

Completeness

- BFS is complete, meaning for a given search tree, BFS will come up with a solution if it exists.

Optimality

- BFS is optimal as long as the costs of all edges are equal.

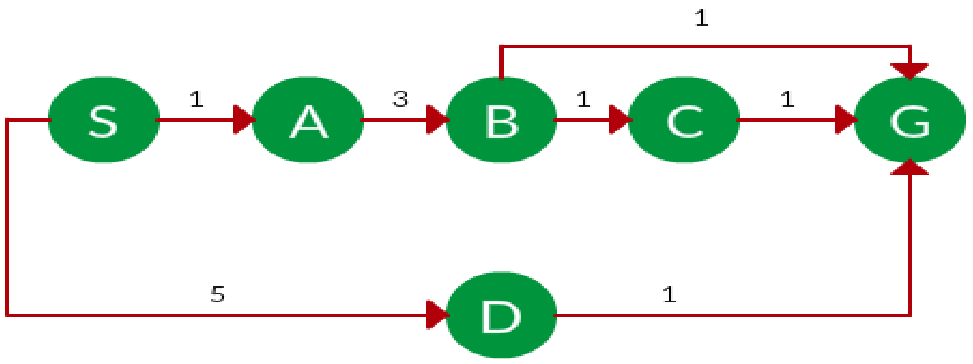
AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

Uniform Cost Search

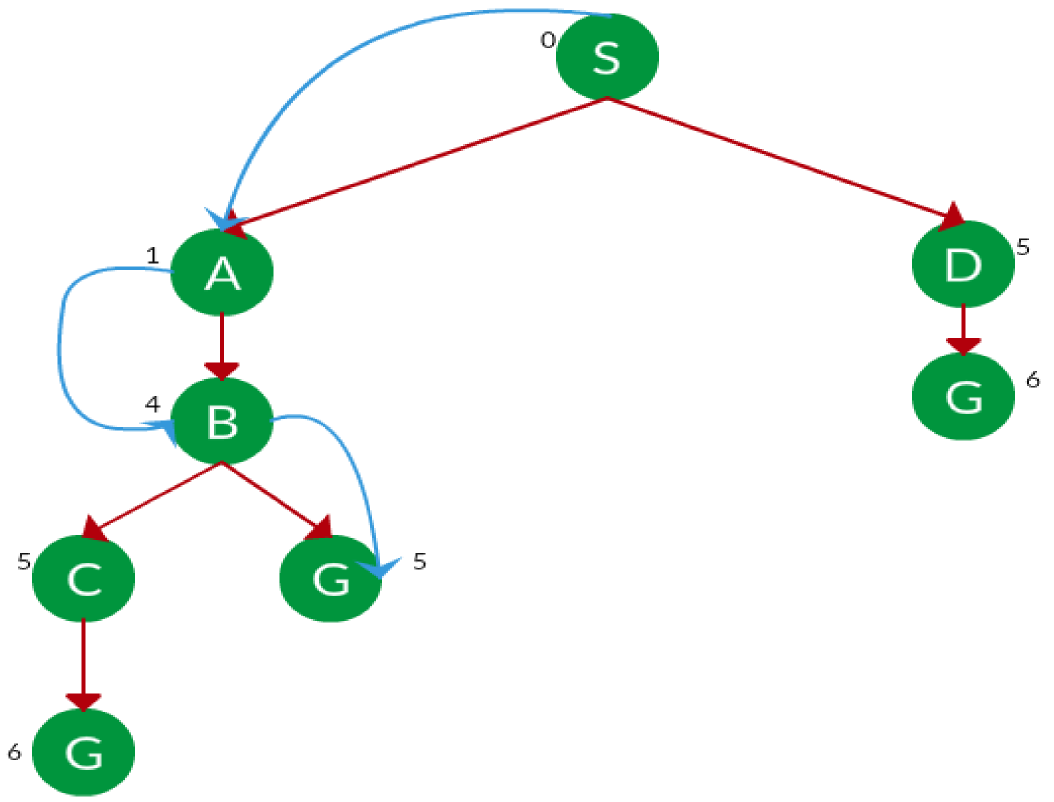
- UCS is different from BFS and DFS because here the costs come into play. In other words, traversing via different edges might not have the same cost. The goal is to find a path where the cumulative sum of costs is least.
- cost(node) = cumulative cost of all nodes from root cost(root) = 0

Example:

- **Question.** Which solution would UCS find to move from node S to node G if run on the graph below?



Solution. The equivalent search tree for the above graph is as follows. Cost of each node is the cumulative cost of reaching that node from the root.



Path: S -> A -> B -> G
Cost: 5

Let = cost of solution.
= arcs cost.

Advantages

- UCS is complete.
- UCS is optimal.

Disadvantages

- Explores options in every “direction”.
- No information on goal location.

Informed Search Algorithms

- Here, the algorithms have information on the goal state, which helps in more efficient searching. This information is obtained by something called a *heuristic*.

- Greedy Search
- A* Tree Search
- A* Graph Search

- **Search Heuristics:** In an informed search, a heuristic is a *function* that estimates how close a state is to the goal state.

  For examples – Euclidean distance, etc.

AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

Problem Characteristics

1. Is the problem decomposable ?
2. Can solution steps be ignored or undone?
3. Is the Universal Predictable?
4. Is good solution absolute or relative ?(Is the solution a state or a path ?)
5. The knowledge base consistent ?
6. What is the role of Knowledge?
7. Does the task requires interaction with the person.
8. Problem Classification

Heuristic Search

- A *heuristic* is a method that
- might not always find the best solution
- *but* is guaranteed to find a good solution in reasonable time.
- By sacrificing completeness it increases efficiency.
- Useful in solving tough problems which
  - ➢ could not be solved any other way.
  - ➢ solutions take an infinite time or very long time to compute.

Generate and Test Algorithm

- Generate a possible solution which can either be a point in the problem space or a path from the initial state.

- Test to see if this possible solution is a real solution by comparing the state reached with the set of goal states.

- If it is a real solution, return. Otherwise repeat from 1.

Hill Climbing Algorithm

- Hill climbing algorithm is a technique which is used for optimizing the mathematical problems.
- It is also called greedy local search as it only looks to its good immediate neighbor state and not beyond that.
- A node of hill climbing algorithm has two components which are state and value.
- Hill Climbing is mostly used when a good heuristic is available.
- In this algorithm, we don't need to maintain and handle the search tree or graph as it only keeps a single current state.

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

Features of Hill Climbing

1. **Generate and Test variant**

- Hill Climbing is the variant of Generate and Test method. The Generate and Test method produce feedback which helps to decide which direction to move in the search space.

2. **Greedy approach**

- Hill-climbing algorithm search moves in the direction which optimizes the cost.

3. **No backtracking**

- It does not backtrack the search space, as it does not remember the previous states.

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

State Space Diagram for Hill Climbing

- The state-space landscape is a graphical representation of the hill-climbing algorithm which is showing a graph between various states of algorithm and Objective function/Cost.

- On Y-axis we have taken the function which can be an objective function or cost function, and state-space on the x-axis. If the function on Y-axis is cost then, the goal of search is to find the global minimum and local minimum.

- If the function of Y-axis is Objective function, then the goal of the search is to find the global maximum and local maximum.

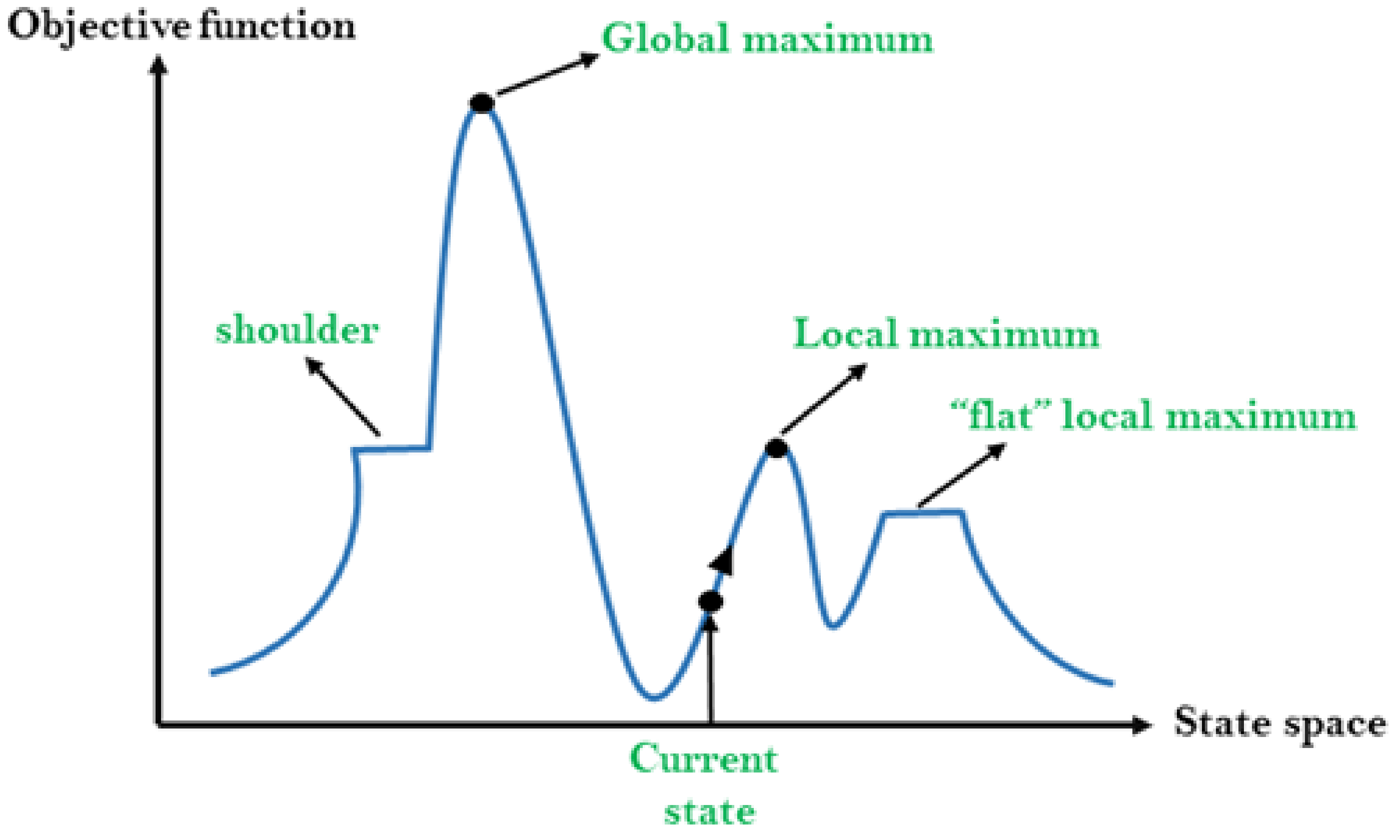
1. **Local Maximum:** Local maximum is a state which is better than its neighbor states, but there is also another state which is higher than it.

2. **Global Maximum:** Global maximum is the best possible state of state space landscape. It has the highest value of objective function.

3. **Current state:** It is a state in a landscape diagram where an agent is currently present.

4. **Flat local maximum:** It is a flat space in the landscape where all the neighbor states of current states have the same value.

AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY


Objective function
Global maximum
shoulder
Local maximum
"flat" local maximum
State space
Current state

Types of Hill Climbing Algorithm

1. Simple hill Climbing
2. Steepest-Ascent hill-climbing

1. Simple Hill Climbing

- Simple hill climbing is the simplest way to implement a hill climbing algorithm. It only evaluates the neighbor node state at a time and selects the first one which optimizes current cost and set it as a current state.
- It only checks it's one successor state, and if it finds better than the current state, then move else be in the same state. This algorithm has the following features:
- Less optimal solution and the solution is not guaranteed
- Less time consuming

Algorithm for Simple Hill Climbing

Step 1: Evaluate the initial state, if it is goal state then return success and Stop.

Step 2: Loop Until a solution is found or there is no new operator left to apply.

Step 3: Select and apply an operator to the current state.

Step 4: Check new state:

- If it is goal state, then return success and quit.
- Else if it is better than the current state then assign new state as a current state.
- Else if not better than the current state, then return to step2.

Step 5: Exit.

AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

2. Steepest-Ascent hill climbing

- The steepest-Ascent algorithm is a variation of simple hill climbing algorithm.
- This algorithm examines all the neighboring nodes of the current state and selects one neighbor node which is closest to the goal state.
- This algorithm consumes more time as it searches for multiple neighbors

Algorithm for Steepest-Ascent hill climbing

1. Evaluate the initial state, if it is goal state then return success and stop, else make current state as initial state.
2. Loop until a solution is found or the current state does not change.

- Let SUCC be a state such that any successor of the current state will be better than it.
- For each operator that applies to the current state:

a. Apply the new operator and generate a new state.
b. Evaluate the new state.
c. If it is goal state, then return it and quit, else compare it to the SUCC.
d. If it is better than SUCC, then set new state as SUCC.
e. If the SUCC is better than the current state, then set current state to SUCC.

3. Exit.

Simulated Annealing

- A hill-climbing algorithm which never makes a move towards a lower value guaranteed to be incomplete because it can get stuck on a local maximum.

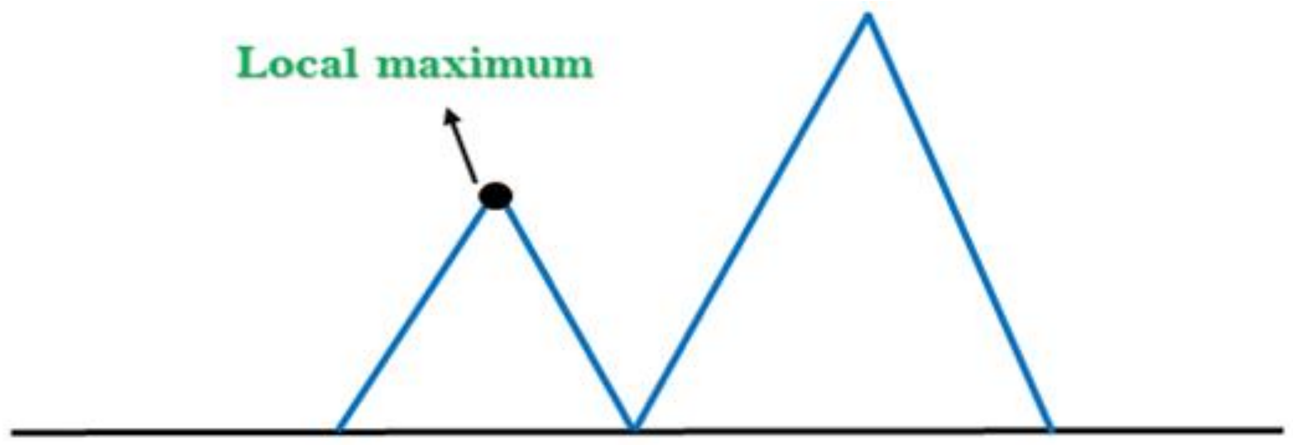
- And if algorithm applies a random walk, by moving a successor, then it may complete but not efficient.

- **Simulated Annealing** is an algorithm which yields both efficiency and completeness.

Problems in Hill Climbing Algorithm

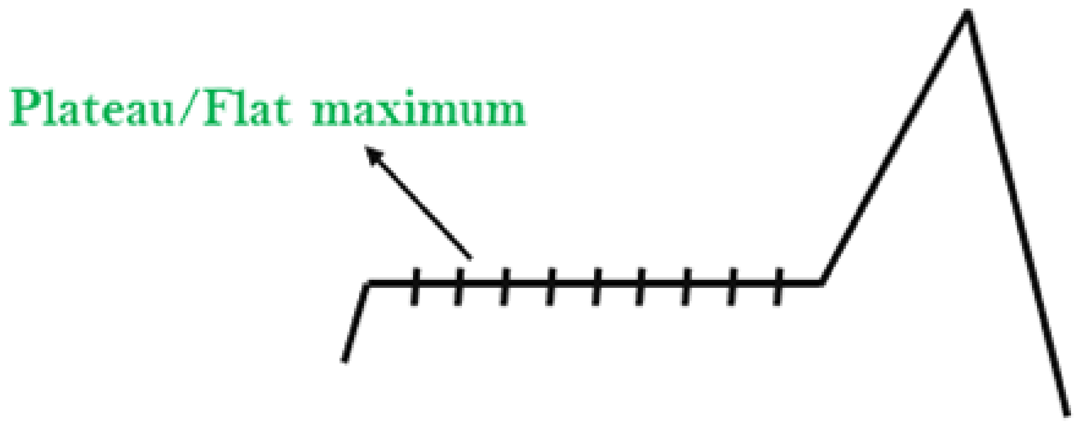
1. Local Maximum

- A local maximum is a peak state in the landscape which is better than each of its neighboring states, but there is another state also present which is higher than the local maximum.



2. Plateau

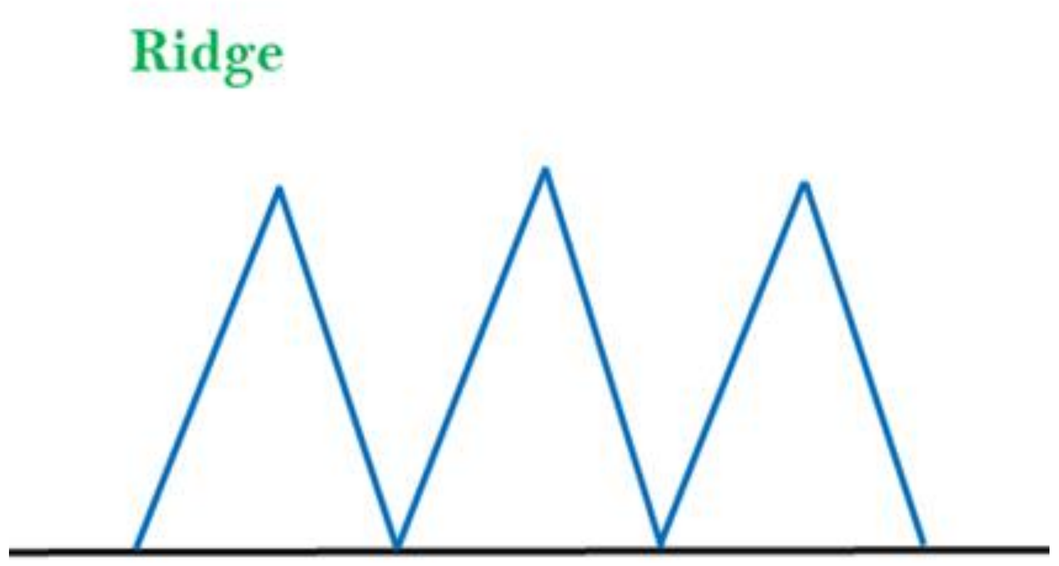
- A plateau is the flat area of the search space in which all the neighbor states of the current state contains the same value, because of this algorithm does not find any best direction to move. A hill-climbing search might be lost in the plateau area.



AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

3. Ridges

- A ridge is a special form of the local maximum.
- It has an area which is higher than its surrounding areas, but itself has a slope, and cannot be reached in a single move.



Best-first Search Algorithm (Greedy Search)

Step 1: Place the starting node into the OPEN list.

Step 2: If the OPEN list is empty, Stop and return failure.

Step 3: Remove the node n, from the OPEN list which has the lowest value of h(n), and places it in the CLOSED list.

Step 4: Expand the node n, and generate the successors of node n.

Step 5: Check each successor of node n, and find whether any node is a goal node or not. If any successor node is goal node, then return success and terminate the search, else proceed to Step 6.

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

Step 6: For each successor node, algorithm checks for evaluation function f(n), and then check if the node has been in either OPEN or CLOSED list. If the node has not been in both list, then add it to the OPEN list.

Step 7: Return to Step 2.

Advantages

- Best first search can switch between BFS and DFS by gaining the advantages of both the algorithms.

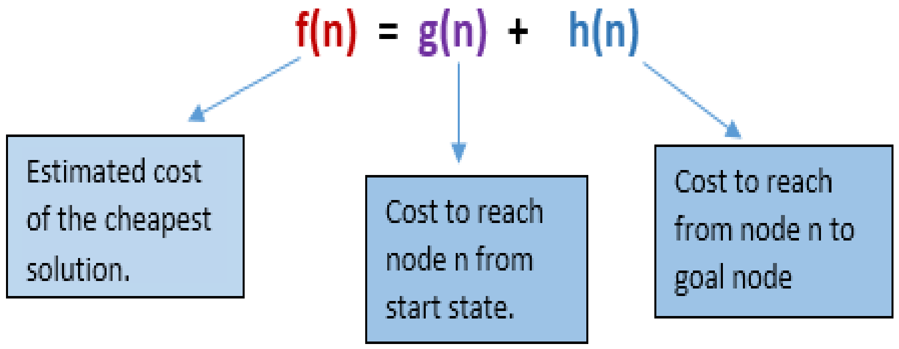
Disadvantages

- It can behave as an unguided depth-first search in the worst case scenario.
- This algorithm is not optimal.

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

A* Search Algorithm

- A* search is the most commonly known form of best-first search. It uses heuristic function h(n), and cost to reach the node n from the start state g(n).
- A* search algorithm finds the shortest path through the search space using the heuristic function. This search algorithm
- expands less search tree and provides optimal result faster. A* algorithm is similar to UCS except that it uses g(n)+h(n) instead of g(n).


f(n) = g(n) + h(n)
Estimated cost of the cheapest solution.
Cost to reach node n from start state.
Cost to reach from node n to goal node

Algorithm of A* search

Step1: Place the starting node in the OPEN list.

Step 2: Check if the OPEN list is empty or not, if the list is empty then return failure and stops.

Step 3: Select the node from the OPEN list which has the smallest value of evaluation function (g+h), if node n is goal node then return success and stop, otherwise

Step 4: Expand node n and generate all of its successors, and put n into the closed list. For each successor n', check whether n' is already in the OPEN or CLOSED list, if not then compute evaluation function for n' and place into Open list.

Step 5: Else if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest g(n') value.

Step 6: Return to **Step 2**.

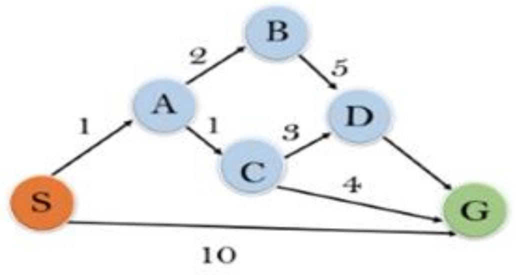
Advantages

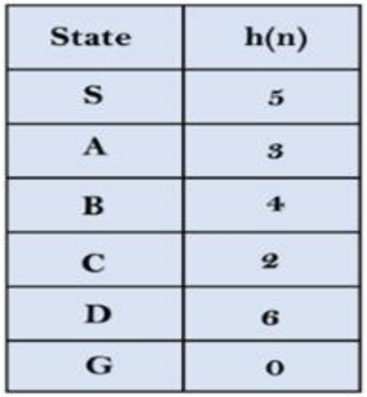
- A* search algorithm is the best algorithm than other search algorithms.
- A* search algorithm is optimal and complete.
- This algorithm can solve very complex problems.

Disadvantages

- It does not always produce the shortest path as it mostly based on heuristics and approximation.
- A* search algorithm has some complexity issues.
- The main drawback of A* is memory requirement as it keeps all generated nodes in the memory, so it is not practical for various large-scale problems.

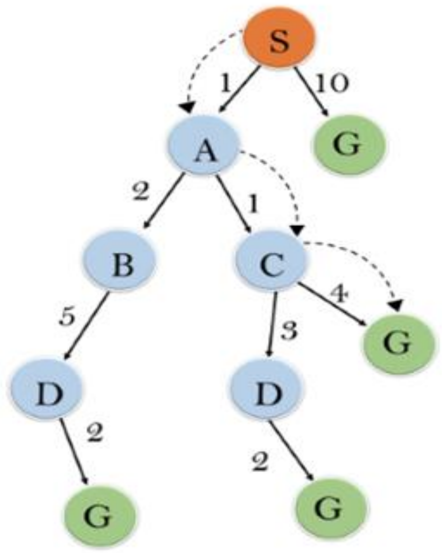
Example





State	h(n)
S	5
A	3
B	4
C	2
D	6
G	0

Solution



- **Initialization:** {(S, 5)}
- **Iteration1:** {(S--> A, 4), (S-->G, 10)}
- **Iteration2:** {(S--> A-->C, 4), (S--> A-->B, 7), (S-->G, 10)}
- **Iteration3:** {(S--> A-->C--->G, 6), (S--> A-->C--->D, 11), (S--> A-->B, 7), (S-->G, 10)}
- **Iteration 4** will give the final result, as **S--->A--->C--->G** it provides the optimal path with cost 6.

Problem Reduction (AND - OR graphs - AO * Algorithm)

- When a problem can be divided into a set of sub problems, where each sub problem can be solved separately and a combination of these will be a solution, AND-OR graphs or AND - OR trees are used for representing the solution.

- The decomposition of the problem or problem reduction generates AND arcs. One AND are may point to any number of successor nodes.

- An algorithm to find a solution in an AND - OR graph must handle AND area appropriately. A* algorithm can not search AND - OR graphs efficiently. This can be understand from the give figure.

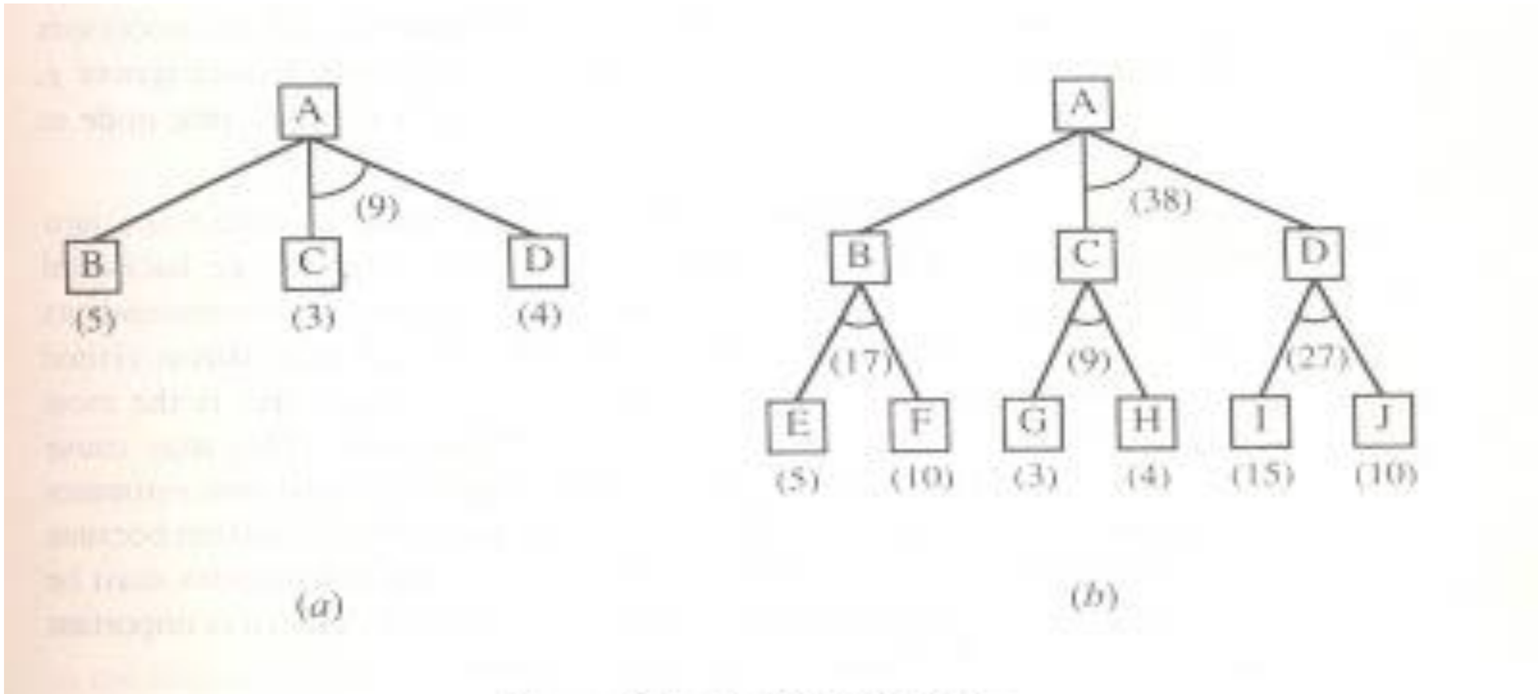


Figure 3.7: AND-OR Graphs

AO* Algorithm

1. Let G consists only to the node representing the initial state call this node INTT. Compute h' (INIT).
2. Until INIT is labeled SOLVED or hi (INIT) becomes greater than FUTILITY, repeat the following procedure.

- Trace the marked arcs from INIT and select an unbounded node NODE.
- Generate the successors of NODE . if there are no successors then assign FUTILITY as h' (NODE). This means that NODE is not solvable. If there are successors then for each one called SUCCESSOR, that is not also an ancester of NODE do the following

a. Add SUCCESSOR to graph G
b. If successor is not a terminal node, mark it solved and assign zero to its h ' value.
c. If successor is not a terminal node, compute it h' value.

- Propagate the newly discovered information up the graph by doing the following .let S be a set of nodes that have been marked SOLVED. Initialize S to NODE. Until S is empty repeat the following procedure;

a. Select a node from S call if CURRENT and remove it from S.

b. Compute h' of each of the arcs emerging from CURRENT , Assign minimum h' to CURRENT.
c. Mark the minimum cost path a s the best out of CURRENT.
d. Mark CURRENT SOLVED if all of the nodes connected to it through the new marked are have been labeled SOLVED
e. If CURRENT has been marked SOLVED or its h ' has just changed, its new status must be propagate backwards up the graph . hence all the ancestors of CURRENT are added to S.

AMIRAJ

COLLEGE OF ENGINEERING & TECHNOLOGY