RGPVNOTES.IN

Subject Name: **Artificial Intelligence**

Subject Code: **IT-7005**

Semester: **7th**

Unit I

Meaning and definition of Artificial Intelligence

- "The study of how to make computers do things at which, at the moment, people are better." (Rich and Knight)
- The art of creating machines that perform functions that require intelligence when performed by people ?
- The study of computer system that attempt to model and apply the intelligence of the human mind.
- A branch of computer science dealing with the simulation of intelligent behaviour in computers.

In AI involves borrowing characteristics from human intelligence, and applying them as algorithms in a computer friendly way. From a programming perspective, AI includes the study of symbolic programming, problem solving, and search.

Areas of AI:

- Robotics
- Speech
- Vision
- Natural Language Processing
- ANN
- Expert System
- Understanding

AI vs Conventional Program

Features	Artificial Intelligence	Conventional Programming
Processing Type	**Symbolic**	**Numeric**
Nature of Input	**Can be Incomplete**	**Must be Complete**
Search	**Heuristic (Mostly)**	**Algorithms**

Explanation	**Provided**	**Usually Not Provided**
Major Interest	**Knowledge**	**Data, Information**
Structure	**Separation of Control from Knowledge**	**Control Integrated with Information (Data)**
Maintenance and Update	**Easy Because of Modularity**	**Usually Difficult**
Hardware	**Mainly Workstations and Personal Computers**	**All Types**
Reasoning Capability	**Limited, but Improving**	**None**

The intelligence is intangible (invisible). It is composed of −

- Reasoning
- Learning
- Problem Solving
- Perception
- Linguistic Intelligence (Semantic)

Various types of production systems

A system that uses this form of knowledge representation is called a production system.

Four classes of production systems:-

1. **A monotonic production system**: It is a production system in which the application of a rule never prevents the later application of another rule that could also have been applied at the time first rule was selected.

2. **A non monotonic production system:** A non-monotonic production system is one in which this is not true.

3. **A partially commutative production system**: A partially commutative production system is a production system with the property that if the application of a particular sequence of rules transforms state x into state y then any permutation of those rules that is allowable (i.e. Each

rules preconditions are satisfied when it is applied) also transforms state x into state y. Partially commutative, monotonic production systems are useful for solving ignorable problems.

4. **A commutative production system**: The system is a production system that is both monotonic and partially commutative.

Characteristics of production systems

Production system is a mechanism that describes and performs the search process.

1. A set of rules.

2. One or more knowledge or database.

3. A control strategy that specifies the order of the rules to be applied.

4. A rule applied.

Study and comparison of breadth first search and depth first search technique

Breadth-First Search Strategy (BFS)

This is an exhaustive search technique. The search generates all nodes at a particular level before proceeding to the next level of the tree. The search systematically proceeds testing each node that is reachable from a parent node before it expands to any child of those nodes. Search terminates when a solution is found and the test returns true.

Algorithm:

1. Create a variable called NODE-LIST and set it to initial state.
2. Until a goal state is found or NODE-LIST is empty do:
   i. Remove the first element from NODE-LIST and call it E. If NODE-LIST was empty, quit.
   ii. For each way that each rule can match the state described in E do:
      a. Apply the rule to generate a new state.
      b. If the new state is a goal state, quit and return this state.
      c. Otherwise, add the new state to the end of NODE-LIST.

Depth-First Search Strategy (DFS)

The Depth-First Search Strategy systematically proceeds to some depth d, before another path

is considered. If the maximum depth of search tree is reached and if the solution has not been found, then the search backtracks to the previous level and explores any remaining alternatives at this level, and so on.

Algorithm:

1. If the initial state is a goal state, quit and return success.
2. Otherwise, do the following until success or failure is signaled:

a. Generate a successor, E, of initial state. If there are no more successors, signal failure.

b. Call Depth-First Search, with E as the initial state

c. If success is returned, signal success. Otherwise continue in this loop.

Comparison: DFS & BFS

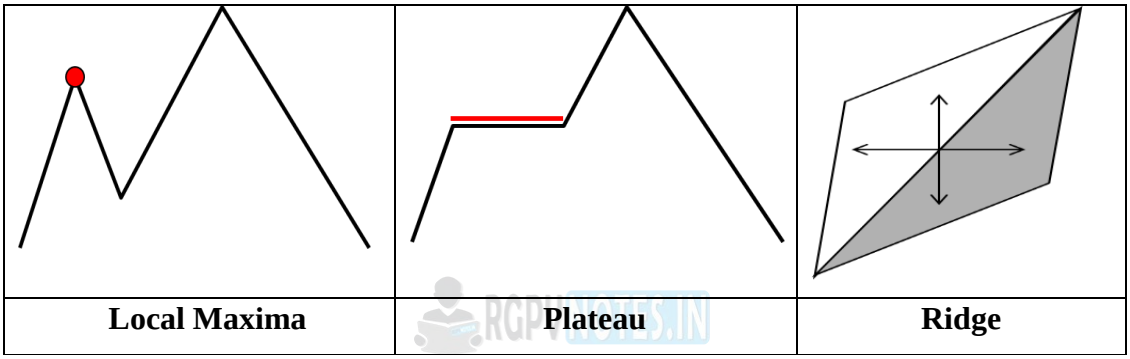
Depth First Search	Breath First Search
DFS requires less memory since only the nodes on the current path are stored.	BFS guarantees that the space of possible moves is systematically examined. This search requires Considerable memory resources.
By chance, DFS may find a solution without examining much of the search space at all. Then it finds Solution faster.	The search systematically proceeds testing each node that is reachable from a parent node before it expands to any Child of those nodes.
If the selected path does not reach to the solution node, DFS gets stuck into a blind alley.	BFS will not get trapped exploring a blind alley.
Does not guarantee to find solution. Backtracking is required if wrong path is selected.	If there is a solution, BFS is guaranteed to find it.

Hill Climbing

Hill Climbing has three well-known drawbacks shown in the following figure.

Local Maxima: a local maximum is a state that is better than all its neighbors but is not better than some other states further away.

i. **Plateau**: a plateau is a flat area of the search space in which, a whole set of neighboring states have the same values.
ii. **Ridge**: is a special kind of local maximum. It is an area of the search space that is higher than surrounding areas and that itself has slop.



Local Maxima	Plateau	Ridge

Hill Climbing Mechanism

- In each of the previous cases (local maxima, plateaus & ridge), the algorithm reaches a point at which no progress is being made.
- A solution is,
  i. Backtrack to some earlier node and try going in a different direction.
  ii. Make a big jump to try to get in a new section.
  iii. Moving in several directions at once.

The Hill-climbing variations are:-

- Stochastic hill-climbing
  - Random selection among the uphill moves.
  - The selection probability can vary with the steepness of the uphill move.
- First-choice hill-climbing

- Stochastic hill climbing by generating successors randomly until a better one is found.

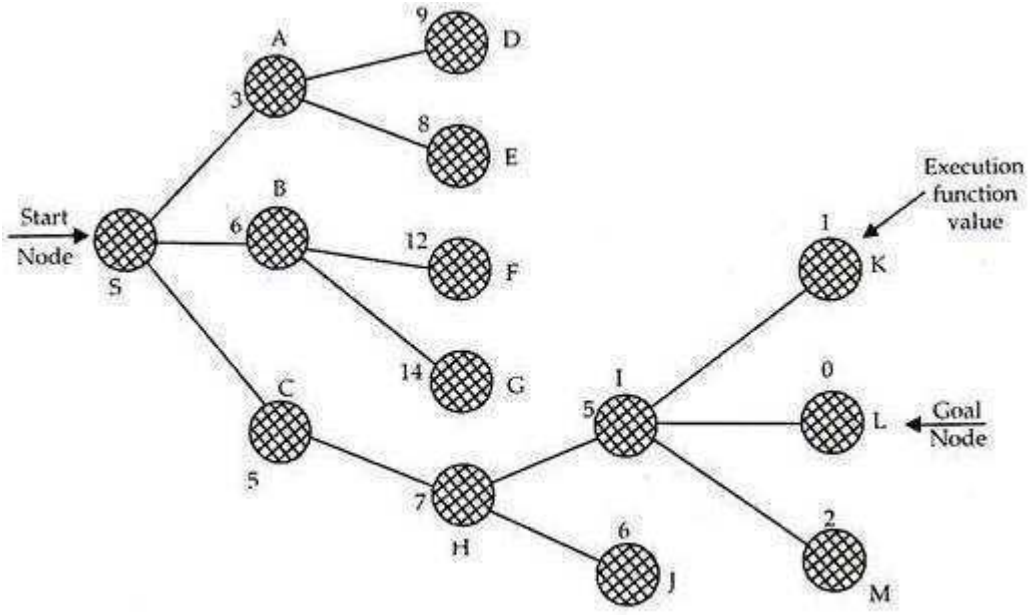
➢ Random-restart hill-climbing

- Tries to avoid getting stuck in local maxima.

Best First (Greedy) Search

Best-first search is a search algorithm which explores a graph by expanding the most promising node chosen according to a specified rule. This specific type of search is called greedy best-first search or pure heuristic search. BFS is good because it does not get trapped on dead end paths.

Best first search combines the advantages of both DFS and BFS into a single method. One way of combining BFS and DFS is to follow a single path at a time, but switch paths whenever some competing path looks more promising than the current one does. At each step of the Best First Search process; we select the most promising of the nodes we have generated so far. This is done by applying an appropriate heuristic function to each of them. We then expand the chosen node by using the rules to generate its successors. If one of them is a solution, we can quit. If not, all those new nodes are added to the set of nodes generated so far.



Example : First the start node S is expanded. It has three children A, B and C with heuristic function values 3, 6 and 5 respectively. These values approximately indicate how far they are from the goal node. The child with minimum value namely A is chosen. The children of A are generated.

They are D and E with values 9 and 8. The search process has now four nodes to search for i.e., node D with value 9, node E with value 8, node B with value 6 and node C with value 5. Of them, node C has got the minimal value which is expanded to give node H with value 7. At this point, the nodes available for search are (D: 9), (E: 8), (B: 6) and (H: 7). Of these, B is minimal and hence B is expanded to give (F: 12), (G: 14).

At this juncture, the node available for search are (D: 9), (E: 8), (H: 7), (F: 12), and (G: 14) out of which (H: 7) is minimal and is expanded to give (I: 5), (J: 6). Nodes now available for expansion are (D: 9), (E: 8), (F: 12), (G: 14), (1:5), (J: 6). Of these, the node with minimal value is (I: 5) which is expanded to give the goal node. The various steps are shown in the table, (queue is not followed strictly as was done in Table)

As we can see, best-first search is "jump all around" in the search graph to identify the node with minimal evaluation function value.

Step #	Node being expanded	Children	Available nodes (open)	Node chosen Best node
1	S	(A: 3), (B: 6), (C : 5)	(A: 3), (B : 6), (C : 5)	(A: 3)
2	A	(D : 9), (E : 8)	(B : 6), (C : 5), (D : 9), (E: 8)	(C: 5)
3	C	(H: 7)	(B : 6), (D : 9), (E : 8), (H: 7)	(B: 6)
4	B	(F : 12), (G : 14)	(D : 9), (E : 8), (H : 7), (F : 12), (G : 14)	(H: 7)
5	H	(I : 5), (J : 6)	(D : 9), (E : 8), (F : 12), (G : 14), (I : 5), (J : 6),	(I: 5)
6	I	(K : 1), (L : 0), (M : 2)	(D: 9), (E : 8), (H : 7), (F : 12), (G : 14), (J : 6) (K: 1), (L : 0), (M : 2)	Search stops as goal is L reached

There is only a minor variation between hill climbing and best-first search. In the former, we sorted the children of the first node being generated, and in the latter we have to sort the entire list to identify the next node to be expanded.

Algorithm: Best First Search

1. Start with OPEN containing just the initial state

2. Until a goal is found or there are no nodes left on OPEN do:

a. Pick the best node on OPEN

b. Generate its successors

c. For each successor do:

i. If it has not been generated before, evaluate it, add it to OPEN, and record its parent.

If it has been generated before, change the parent if this new path is better than the previous one. In that case, update the cost of getting to this node and to any successors that this node may already have.

A* algorithm

It is best-known form of Best First search. It avoids expanding paths that are already expensive, but expands most promising paths first.

f(n) = g(n) + h(n), where

- g(n) the cost (so far) to reach the node
- h(n) estimated cost to get from the node to the goal
- f(n) estimated total cost of path through n to goal. It is implemented using priority queue by increasing f(n).

A* Search also makes use of a priority queue just like Uniform Cost Search with the element stored being the path from the start state to a particular node, but the priority of an element is not the same. In Uniform Cost Search we used the actual cost of getting to a particular node

from the start state as the priority. For A*, we use the cost of getting to a node plus the heuristic at that point as the priority. Let n be a particular node, then we define g(n) as the cost of getting to the node from the start state and h(n) as the heuristic at that node. The priority thus is f(n) = g(n) + h(n). The priority is maximum when the f(n) value is least. We use this priority queue in the following algorithm, which is quite similar to the Uniform Cost Search algorithm:

Insert the root node into the queue

While the queue is not empty

Dequeue the element with the highest priority

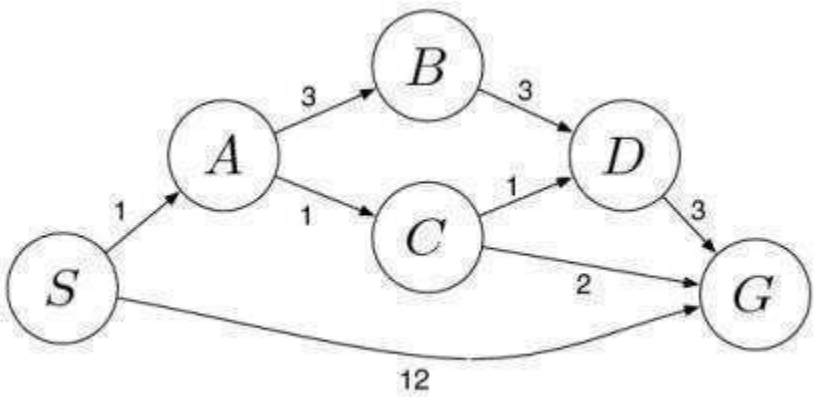
(If priorities are same, alphabetically smaller path is chosen)

If the path is ending in the goal state, print the path and exit

Else

Insert all the children of the dequeued element, with f(n) as the priority

Now let us apply the algorithm on the above search tree and see what it gives us. We will go through each iteration and look at the final output. Each element of the priority queue is written as [path,f(n)]. We will use h1 as the heuristic, given in the diagram below.



State	h1	h2
S	4	3
A	2	2
B	6	5
C	2	1
D	3	2
G	0	0

Initialization: { [S , 4] }

Iteration1: { [S->A , 3] , [S->G , 12] }

Iteration2: { [S->A->C , 4] , [S->A->B , 10] , [S->G , 12] }

Iteration3: { [S->A->C->G , 4] , [S->A->C->D , 6] , [S->A->B , 10] , [S->G , 12] }

Iteration4 gives the final output as S->A->C->G.

AO* algorithms

The AO*

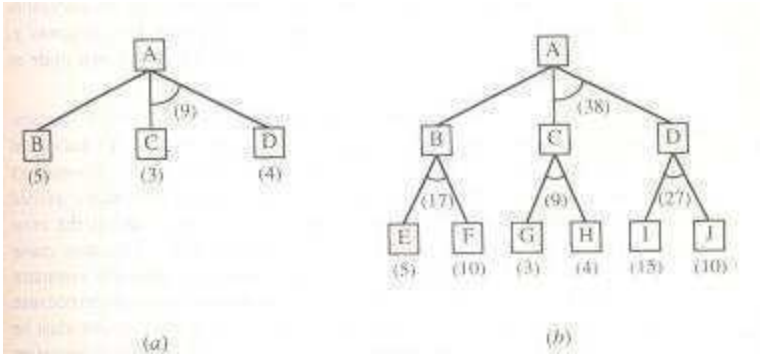
The AO* algorithm will use a single structure GRAPH, representing the part of the search graph that has been explicitly generated so far. Each node in the graph will point both down to its immediate successors and up to its immediate predecessors.

•Each node in the graph will also have associated with it an h' value, an estimate of the cost of a path from itself to a set of solution nodes.

•We will not store g (the cost of getting from the start node to the current node) as we did in the A* algorithm.

•And such a value is not necessary because of the top-down traversing of the edge which guarantees that only nodes that are on the best path will ever be considered for expansion.

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.



Algorithm: AO*

1. Let GRAPH consist only of the node representing the initial state. Call this node INIT, Compute INIT.
2. Until INIT is labeled SOLVED or until INIT's h' value becomes greater than FUTILITY, repeat the following procedure:
   a. Trace the labeled arcs from INIT and select for expansion one of the as yet Unexpanded nodes that occurs on this path. Call the selected node NODE.
   b. Generate the successors of NODE. If there are none, then assign FUTILITY as the h' value of NODE. This is equivalent to saying that NODE is not solvable. If there are successors, then for each one (called SUCCESSOR) that is not also an ancestor of NODE do the following:
      i. Add SUCCESSOR to GRAPH
      ii. If SUCCESSOR is a terminal node, label it SOLVED and assign it an h' value of 0
      iii. If SUCCESSOR is not a terminal node, compute its h' value
   c. Propagate the newly discovered information up the graph by doing the following: Let S be a set of nodes that have been labeled SOLVED or whose h' values have been changed and so need to have values propagated back to their parents. Initialize 5 to NODE.

      Until S is empty, repeat the, following procedure:
      i. If possible, select from S a node none of whose descendants in GRAPH occurs in S. If there is no such node, select any node from S. Call this node CURRENT, and remove it from S.

ii. Compute the cost of each of the arcs emerging from CURRENT. The cost of each arc is equal to the sum of the h' values of each of the nodes at the end of the arc plus whatever the cost of the arc itself is. Assign as CURRENT'S new h' value the minimum of the costs just computed for the arcs emerging from it.
iii. Mark the best path out of CURRENT by marking the arc that had the minimum cost as computed in the previous step.
iv. Mark CURRENT SOLVED if all of the nodes connected to it through the new labeled arc have been labeled SOLVED.
v. If CURRENT has been labeled SOLVED or if the cost of CURRENT was just changed, then its new status must be propagated back up the graph. So add all of the ancestors of CURRENT to S.

Types of control strategies

Control strategies help us decide which rule to apply next during the process of searching for a solution to a problem. What to do when there are more than 1 matching rules?

Good control strategy should:

- It should cause motion
- It should be Systematic

The first requirement of a control strategy is that it must cause motion. The second requirement of a control strategy is that issue must be systematic. We will explain these two with respect to water jug problem. If we have implemented choosing the first operator and then the one which matches the first one, then we would not have solved the problem. If we follow any strategy which can cause some motion then will lead to a solution. But if it is not followed systematically, and then got the solution. One day to follow a systematic control strategy is to construct a tree with the initial state as its root. By applying all possible combinations from the first level leaf nodes. Continue the process until some rule produces a goal state.

Control strategies are classified as:

Uninformed/blind search control strategy:

- Do not have additional information about states beyond problem definition.

- Total search space is looked for the solution.
- Example: Breadth First Search (BFS), Depth First Search (DFS), Depth-Limited Search (DLS).

Informed/Directed Search Control Strategy:

- Some information about problem space is used to compute preference among the various possibilities for exploration and expansion.
- Examples: Best First Search, Problem Decomposition, A*, Mean end Analysis

The control strategy for the search process is called breadth first search. Other systematical control strategies are also available for example, we can select one single branch of a tree until it yields a solution or until some pre specified depth has been reached. If not we go back and explore to other branches this is called depth – first – search. The water jug problems will lead to an answer by adoption any control strategy because the problem is simple. This is not always the case.

- Selecting rules; keeping track of those sequences of rules that have already been tried and the states produced by them.
- Goal state provides a basis for the termination of the problem solving task.

1- PATTERN MATCHING STAGE
Execution of a rule requires a match.

preconditions match content of
of a rule <=====> the working memory.

when match is found => rule is applicable several rules may be applicable

2- CONFLICT RESOLUTION (SELECTION STRATEGY) STAGE
Selecting one rule to execute;

3- ACTION STAGE
Applying the action part of the rule => changing the content of the workspace

=>new patterns, new matches => new set of rules eligible for execution

Recognize -act control cycle

RGPVNOTES.IN

We hope you find these notes useful.

You can get previous year question papers at https://qp.rgpvnotes.in .

If you have any queries or you want to submit your study notes please write us at rgpvnotes.in@gmail.com

LIKE & FOLLOW US ON FACEBOOK
facebook.com/rgpvnotes.in