Unit - 3

Search Techniques in A.I.

Data Driven & Goal Driven Search

Data driven search is also called forward chaining Data driven search/reasoning takes the facts of the problem and applies the rules and legal moves to produce new facts that lead to a goal; Goal driven reasoning issues on the goal, finds the rules that could produce the goal, and chains through successive, rules and subgoals to the given fact of the problem.

Data driven search uses the knowledge and Constraints found in the given data of a problem to guide search along the lines to be true.

Goal driven search uses knowledge of the desired goal to guide the search through relevant rules and eliminate branches of the space.

Data driven search is appropriate to the problem in which:

* All or most of the data are given in the initial problem statement.
* There are a large number of potential goals

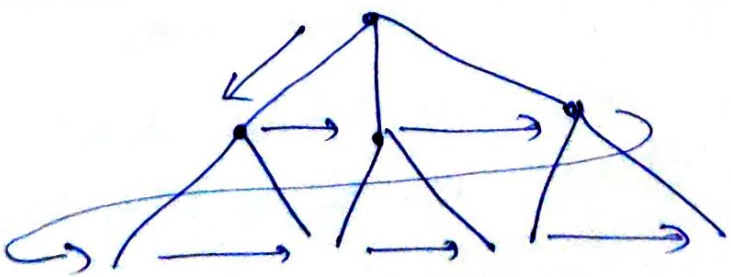
but there are only a few ways to use the facts and given information of a particular problem instance.

Uninformed Search

When no information is known a priori, a search program must perform a blind or uninformed search. A blind or uninformed search algorithm is one that uses no information other than the initial state, the search operators, and a test for solution. Search program may be required to return only a solution value when a goal is found or to record and return the solution. Two types of blind search techniques are given below: Blind search.

- Breadth first Search
- Depth first Search

Breadth first Search



Algo for breadth-first algorithm is as follows:-

1. Place the starting node s on queue.
2. If the queue is empty, return fail and stop.

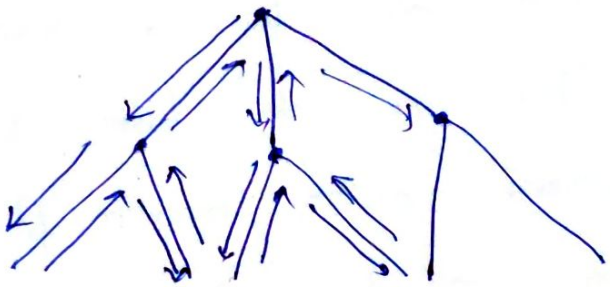
(3) If first element on the queue is a goal node g, return success and stop otherwise. (3)

(4) Remove and expand the first element from the queue and place all the children at the end of the queue in any order.

(5) Return to step (2).

Time complexity of breadth-first search is $O(b^d)$.

Depth - First Search



Algorithm for Depth-first search

(1) Place the starting node s on the queue.

(2) If the queue is empty, return failure and stop.

(3) If the first element on the queue is a goal node g, return success and stop otherwise,

(4) Remove and expand the first element, and place the children at the front of the queue.

(5) Return to step (2).

The time complexity of the depth-first tree search is $O(b^d)$ the same as that for breadth-first

Depth Search with Iterative Deepening

Depth first iterative deepening searches are performed as a form of repetitive depth first search moving to a successively deeper depth with each iteration. It begins by performing a depth-first search to a depth of one. It then discards all the nodes generated and starts over doing a search to a depth of two. If no goal has been found, it discards all the nodes generated and does a depth-first search to a depth of three. This process continues until a goal node is found or some maximum depth is reached.

Depth first iterative deepening search expands all nodes at a given depth before expanding nodes at a greater depth, it is guaranteed to find a shortest-path solution. The main disadvantages of this method is that is performs wasted computations before reaching to a goal depth. The time and space complexities of this search are $O(b^d)$ and $O(d)$ respectively.

Informed Search

When more information than the initial state the operators and goal test is available, the size of search can usually be constrained. When this is case, the better the information available, the more efficient the search process will be. Such methods are known as informed search methods.

Hill Climbing Search

Hill Climbing is often used when a good heuristic function is available for evaluating states but when no other useful knowledge is available.

Simple Hill Climbing:

The simplest way to implement hill climbing is as follows:-

Algorithm: Simple Hill Climbing.

1. Evaluate the initial state. If it is a goal state, then return it and quit. Otherwise continue with the intial state as the current state.
2. Loop until a solution is found or until there are no new operators left to be applied in the current state:

a) Select an operator that has not yet been applied to the current state and apply it to produce a new state.

b) Evaluate the new state.

i) If it is a goal state, then return it and quit.

ii) If it is not a goal state but it is better than the current state, then make it the current state.

iii) If it is not better than the current state, then continue in the loop.

Steepest-Ascent Hill Climbing.

A useful variation on simple Hill Climbing considers all the moves from the current state and selects the best one as the next state. This method is called steepest-ascent hill climbing or gradient search.

Algorithm:-

1. Evaluate the initial state. If it is goal state, return it and quit. Otherwise continue with the initial state as the current state.

2. Loop until a solution is found or until a complete iteration produces no change to the current state

a) Let SUCC be a state such that any possible successor of the current state will be better

than SUCC.

(b) For each operator that applies to the current state do:

i) Apply the operator and generate a new state.

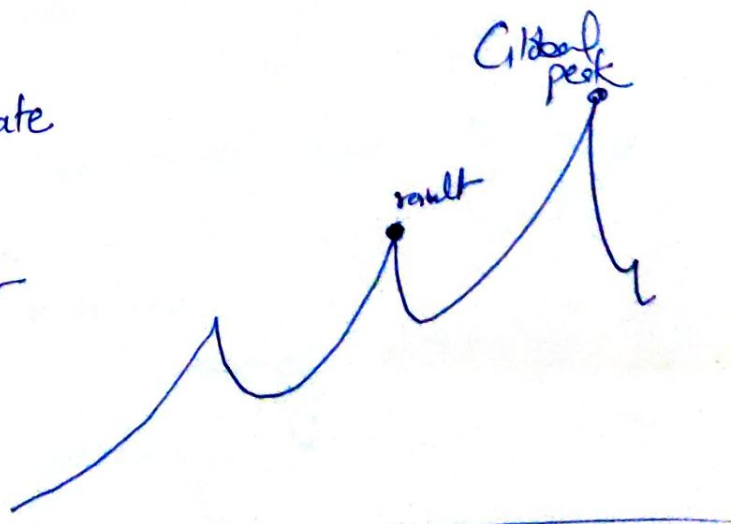
ii) Evaluate the new state. If it is a goal then return it and quit. If not, compare it to SUCC. If it is better then set SUCC to this state. If it is not better, leave SUCC alone.

iii) If the SUCC is better than current state, then set current to SUCC.

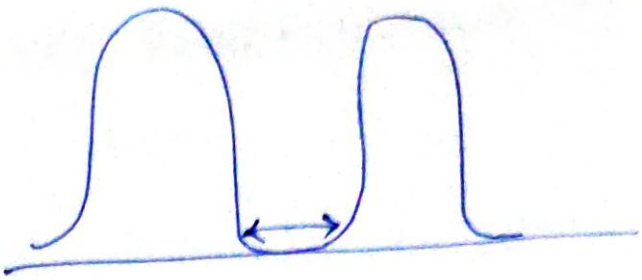
Both basic and steepest-ascent hill climbing may fail to find a solution. Either algorithm may terminate not by finding but getting to a state from which no better states can be generated.

Local maxima :-

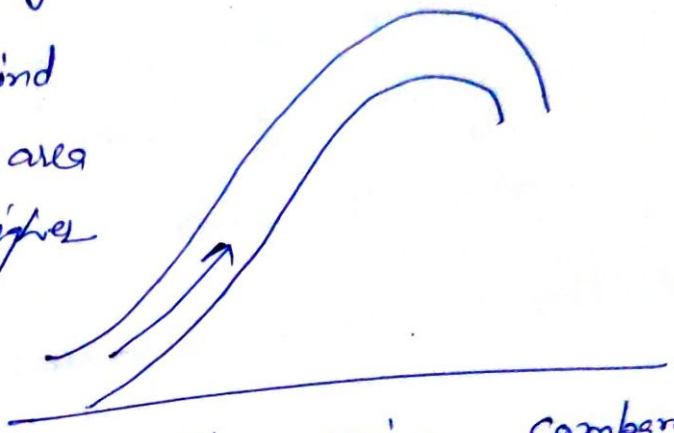
A local maxima is a state that is better than all its neighbours but is not better than some other states farther away.



Plateau – is a flat area of search space in which a whole set of neighboring states have the same value. On a plateau, it is not possible to determine the best direction in which to move by making local comparisons.



Ridge:- is a special kind of local maximum. It is an area of search space that is higher than surrounding areas and that itself has a slope. But the orientation of high region, compared to the set of available moves and the directions in which they move, makes it impossible to traverse a ridge by single moves.



These are some ways of dealing with these problems, although these methods are by no mean guaranteed!

* Backtrack to some earlier node and try going in different direction. This is a fairly good way of dealing with local maxima.
* Make a big jump in some direction to try to get to a new section of search space. This is particularly good way of dealing with plateaus.

*) Apply two or more rules before doing the test. This corresponds to moving in several directions at once. This is particularly good strategy for dealing with ridge.

Best First Search :-

At each step of the best-first search process, select most promising of the nodes have generated so far. This is done by applying an appropriate heuristic function to each of them.

To implement a graph-search procedure, we will need to use two list of nodes:

* OPEN - nodes that have been generated and have had the heuristic function applied to them but which have not yet been examined. OPEN is actually a priority queue in which the elements with highest priority are those with most promising value of the heuristic function.
* CLOSED - nodes that have already been examined.

The function g is a measure of the cost of getting from the initial state to the current node. The function h' is an estimate of the additional cost of getting from the current node to a goal state.

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

A* Algorithm

1. Start with OPEN containing only the initial node. Set that node's g value to 0, its h' value to whatever it is. Set CLOSED to the empty list.
2. Until a goal node is found, repeat the following procedure: If there are no nodes on OPEN, report failure. Otherwise, pick the node on OPEN with the lowest f' value. Call it BESTNODE. Remove it from OPEN. Place it on CLOSED. Generate the successors of BESTNODE but do not set BESTNODE to point to them yet.

For each such SUCCESSOR, do the following:

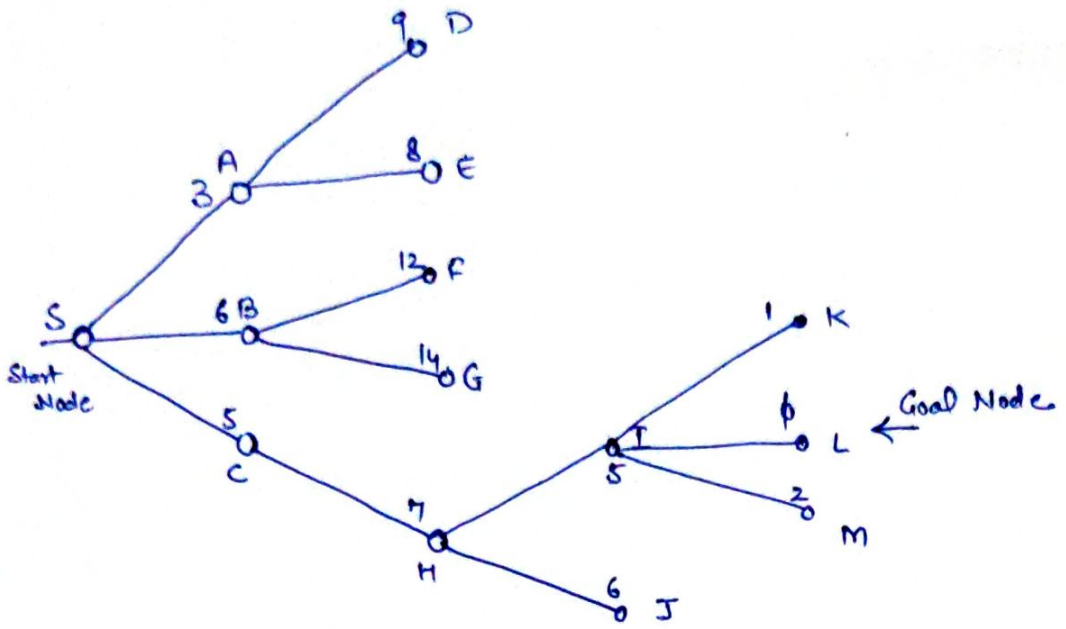
(a) Set SUCCESSOR to point back to BESTNODE. These backwards links will make it possible to recover the path once a solution is found.

(b) Compute g(SUCCESSOR) = g(BESTNODE) + cost of getting from BESTNODE to SUCCESSOR.

(c) See if SUCCESSOR is same as any node on OPEN. If so call that node OLD. Since this node already exists in graph, we can throw SUCCESSOR away and add Old to the list of BESTNODE'S successor.

d) If SUCCESSOR was not on OPEN, see if it on CLOSED. If so, call the node on CLOSED OLD and add OLD to the list of BESTNODE's successors Check to see if the new path or the old path is better just as in step 2(c), and set the parent link and g and f' values appropriately. If the path we are propagating through is now better, reset the parent and continue propagation.

(e) If SUCCESSOR was not already on either OPEN or CLOSED, then put it on OPEN, and add it to the list of BESTNODE's successors. Compute f'(SUCCESSOR) = g(SUCCESSOR) + h'(SUCCESSOR).



step	Node being expanded	Children	Available Node	Node Chosen
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)	(	