Informed Search

Heuristic Function

A *heuristic function* is the one that guides the decision of selection of a path. A heuristic function $h(n)$ provides an estimate of the cost of the path from the given node to reach to the closest goal node.

GENERATE AND TEST

The most easiest and obvious way to solve any problem is to generate a solution and check if this is the real one. The algorithm for the method is discussed below:

1. Generate a possible solution, which can be a node in the problem space or it can be a path from the initial state.
2. Test if the possible solution is the real one, i.e., compare with the goal states.
3. Check solution, if true, return the solution, else go to step 1.

BEST FIRST SEARCH

The positive aspect of DFS is that the goal can be reached without any need to compute all the states, whereas with BFS, it does not get halted or trapped in dead paths. The best first search allows us to switch between the paths, and thus, gets the benefit of both. To do so, the selection of the most promising node is done. So, it analyses and checks if the selected node is better than the previous one. If not found so, it reverts back to the previous path and proceeds. So, backtracking occurs. In order to achieve this, even though a move is selected, other options are kept so that they can be revisited.

BEST FIRST SEARCH

OR Graph

The notion of OR graphs is required in order to avoid the revisiting of paths and for propagating back to the successor, in case it is required. In this case, for the algorithm to proceed, a node contains the following information:

1. Description of the state it represents.
2. Indication how promising it is.
3. Parent link that points to the best node it has reached from.
4. List of nodes that are generated from it.

BEST FIRST SEARCH

1. ***Open list:*** It consists of list of nodes that have been generated and on whom the heuristic function has already been applied, but yet not examined. We can map them to a priority queue. This queue consists of the nodes or the elements with highest priority. This is dependent on the heuristic function that associates the cost with the node, and hence, prioritises them.

2. ***Closed list:*** It contains the nodes that have already been examined. When a node is generated, then this is required to check that whether it has already been generated.

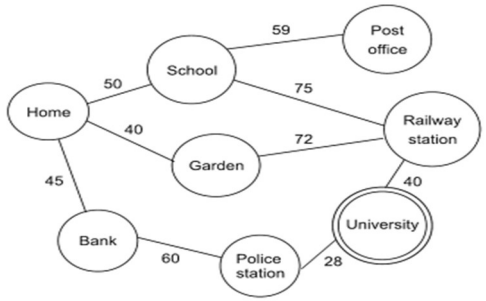
So, what is heuristic and the function evaluation procedure, $f(n)$? $f(n)$ is the function that can be defined as the sum of two elements, i.e., the cost of reaching from the initial node to the current node, $g(n)$ and the additional cost of getting from the current node to the goal state, $h(n)$. So, $g(n) + h(n)$ defines the function.

Best First Search Algorithm

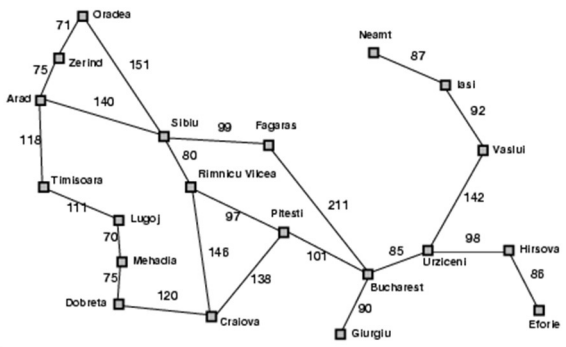
1. Open list ← initial state/node.
2. Do (till goal is found) or (no nodes in the open list).
   (i) Select the best node (since priority queue is used, this is possible) from open list. Add to closed list.
   (ii) Generate the successors.
   (iii) For every successor, do
      (a) If the successor is already generated, change the parent if this new path is better than the earlier one. Accordingly, update the costs to this node as well as to the previous successors, if any.

Best First Search

(b) If the successor is not generated previously, then evaluate it on the basis of heuristic, add it to the open list and make a note of its parent as well.



Romania with step costs in km

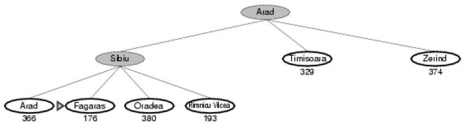


Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

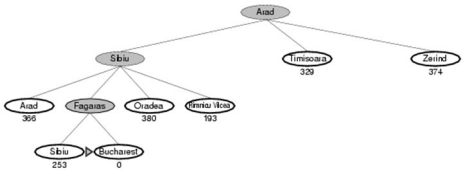
Greedy best-first search example

Greedy best-first search example

Greedy best-first search example



Greedy best-first search example



A* SEARCH

It is the A* approach that now makes use of the function $g(n)$ along with h. Evaluation function $f(n)$ represents the total cost. Here $f(n) = g(n) + h(n)$, where $g(n)$ is the cost so far to reach n, while $h(n)$ is the estimated cost from n to the goal state. The best first search is a special case of A*, where $f(n) = h(n)$. Let us discuss the algorithm in detail.

1. We begin with the open list containing only the initial state.
   (a) Let $g(n) = 0$ and $h(n)$ = value whatever it is.
   (b) So, $f(n) = h(n)$ as $g(n) = 0$.

A* Algorithm

2. Repeat: Do till we get the goal.
   If no nodes are existing in the open list, return failure to reach goal.
   Else
   1. Select and remove node from the open list, with the lowest f value.
   2. Let us call this node the current node or C-node.
   3. Put C-node on the closed list.
   4. Check if C-node = goal state, return solution.
   5. If not, then generate the successor for C-node and for every successor,
      (i) Save the path from the successor to C-node. (Required to retrieve the solution path)
      (ii) Calculate g(successor) ← g(C-node) + Cost of reaching to successor from C-node.

A* Algorithm

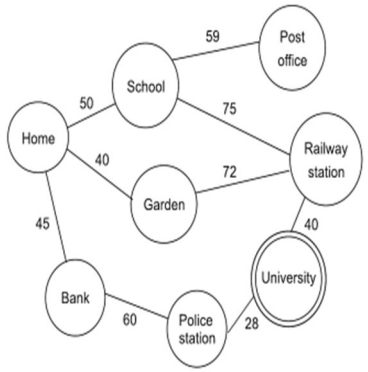
Now, three cases are to be handled.

1. Check if the successor is on open list.
   This means that the node has been generated previously, but yet not considered in the path.
   (a) If so, throw this node (let us call this as Pnode) and add it to C-node's successor list.
   (b) Now, a decision is to be made whether the connect path between the Pnode's parent is set to C-node. This is dependent on the cost. If this cost is cheaper, then it should be. (This is true as successor and the P-node are same). This is done by comparing the g values. If g(successor) is less, reset P-node's link and update f(P-node) considering the value of g(P-node).
2. Check if the successor is on only closed list.
   If the node is present on the closed list, let us call this node as P-node.
   Add this node to the C-node's successors. Again check the costs as we have done in the previous step. Accordingly, update the g and f values.

A* Algorithm

3. Check if the successor is neither on the open list nor on the closed list.
   The most simple case, simply add it to the open list and the list of C-node's successors. Compute f(successor) = g(successor) + h(successor) and proceed.

A* Algorithm



Home 120
Bank 80
Garden 100
School 70
Railway Station 20
Post office 110
Police station 26

A* Algorithm

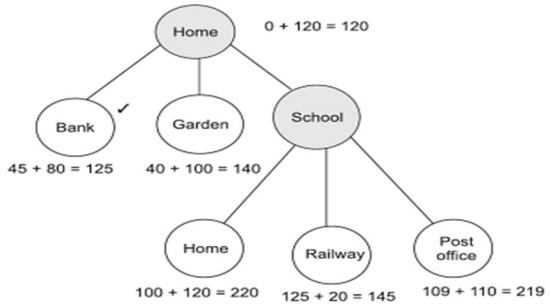
Step 1:

0 + 120 = 120

Step 2:

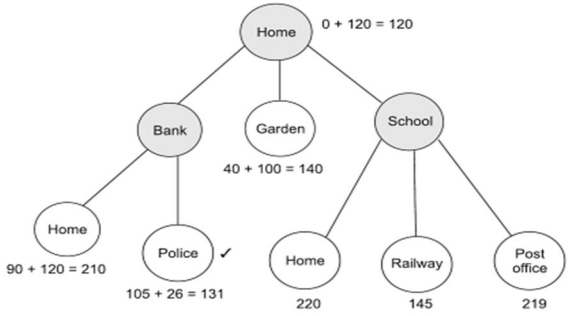
A* Algorithm

Step 3:



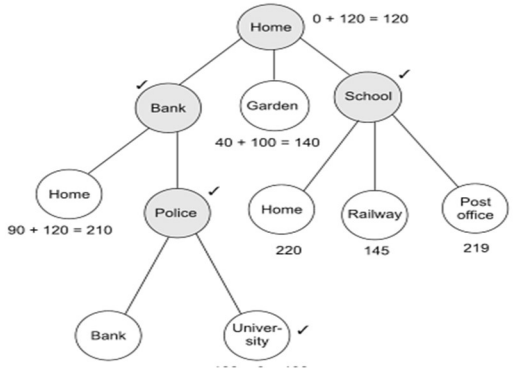
A* Algorithm

Step 4:



A* Algorithm

Step 5:

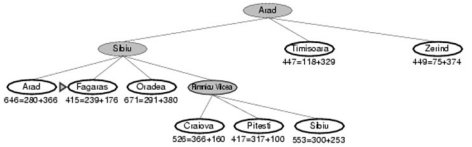


A* search example

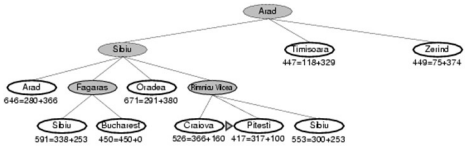
A* search example

A* search example

A* search example



A* search example



A* search example