TOPIC: INFORM SEARCH

Name: Priyanka Kumari
(10800221133)
(L28)
Subject Name: Artificial Intelligence
Paper Code: PEC-IT501B

DEPARTMENT OF INFORMATION TECHNOLOGY
Asansol Engineering College, Asansol
West Bengal – 713305
India

September 2022

Table of Contents

Abstract

These algorithms used to find goal in global state, basically informed
Search type algorithm used where we have idea about goal.

These algorithms are more useful for large search space .and informed
Search algorithm uses the idea of heuristic.

Informed search algorithms have information on the goal state helps in more efficient searching.

Introduction

Informed Search algorithms have information on the goal state which helps in more efficient searching.

The informed search algorithm is more useful for large search space. Informed search algorithm uses the idea of heuristic. In an informed search, a heuristic is a function that estimates how close a state is to the goal state. For example – Manhattan distance, Euclidean distance, etc. (Lesser the distance, closer the goal for more understanding refers above fig1 and table1.

Example: where we use informed search, it is basically greedy type
Algorithm. therefore, it has less chance to give optimal solution.
Now, let see example: suppose we wanted to go Jharkhand here
We know our destination therefore we will use informed search
Real life example: google map

Heuristics function

Here, we learning informed search and we know very well-informed search depend on Heuristic function

Let see, what is Heuristic -space and goal state.

goal state: the destination we want to reach, our "goal". (Notice we can have a couple of goals in a search problem)

State-space: a tuple that contains information about a particular state it is associated with. In general, it can be a tuple like <State, Action, Cost> and we can normally derive if our state is a goal state with this three information.

Now let's talk about heuristics.

A **heuristic** is a function that estimates how close a state to a goal state.
The **heuristics** or **heuristics function** differs from problem to problem and usually need to be designed to fit a certain problem. Most of the artificial intelligence problem can be described as follows: given a huge amount of information, data, and some constraints, tell me how to reach our goal state. In this case, the **heuristics function** tells us how close we are to the goal state. For example, if we need a heuristic function to determine how close a point in a two-dimensional space to our goal point, the distance formula would be a good choice.

In many AI problems, the **heuristics function** is used to "approximate" how close we are to a goal as it's nearly impossible to come up with a function that tells us exactly how close we are. Well, then how do we know if our heuristics is legit for our problem. We need to make sure it's admissible and consistent.

A heuristic is **admissible** if the heuristic value is lower bounds on the actual shortest path cost to our goal state. In short, if our estimated cost by our heuristic function is less than the actual cost, our function is admissible.

A heuristic is **consistent** if it's admissible and it holds that if an action has cost *c*, then taking that action can only cause a drop in heuristic of at most *c*.

Remember that admissibility isn't enough to guarantee correctness in graph search — you need the stronger condition of consistency. However, admissible heuristics are usually also consistent. Therefore, it is usually easiest to start out by brainstorming admissible heuristics and check whether it's consistent

That's it for today. Hope you found what you need to know and message me if you have any questions.

Best-first search Algorithm:

Best first search falls under the category of heuristic search or Informed search.

The idea of **Best First Search** is to use an evaluation function to decide which adjacent is most promising and then explore.

We expand the node that is closest to the goal node in the best first search method, and the closest cost is determined using a heuristic function, i.e.
f(n)= g(n).

Where h(n)= estimated cost from node n to the goal.

The priority queue implements the greedy best first algorithm

Best first search algorithm:

Step 1: Place the initial node in the OPEN list first

Step 2: Stop and return failure if the OPEN list is empty

Step 3: Move node n from the OPEN list to the CLOSED list, as it has the lowest value of h(n). Step 4: Extend node n and construct node n's descendants.

Step 5: Examine each successor of node n to see whether any of them is a goal node. Return success and end the search if any successor node is a goal node; otherwise, proceed to Step 6.

Step 6: The method checks for the evaluation function f(n) for each successor node, then determines if the node has been in the OPEN or CLOSED list. Add the node to the OPEN l if it isn't already on both lists.

Step 7: return to step2

Example:

This below example explained best first algorithm using heuristic Function.

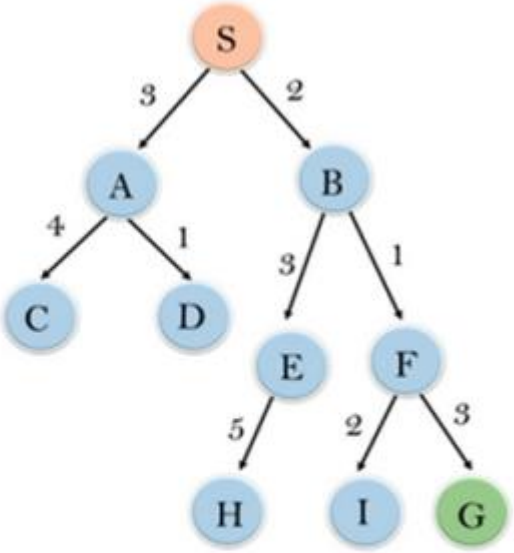
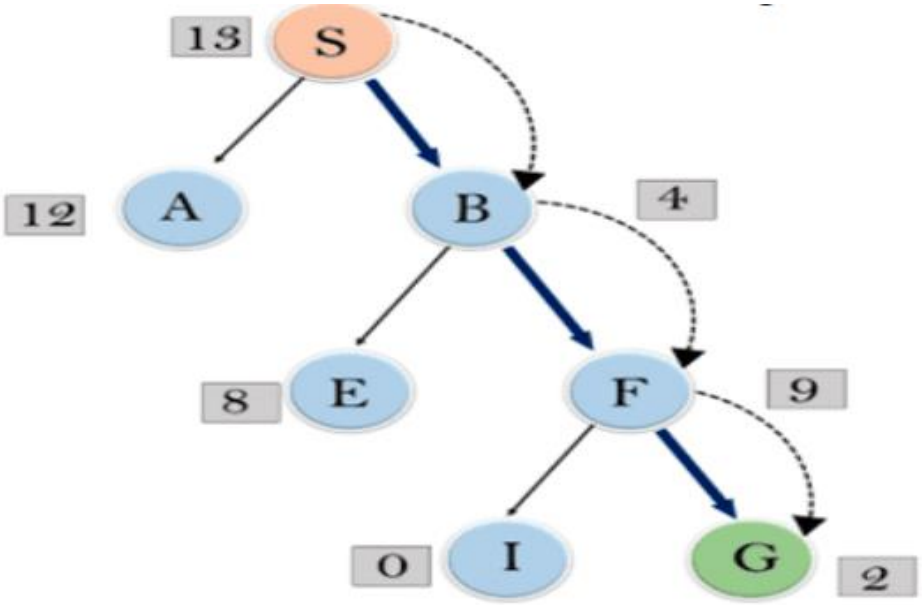


Fig-1

node	H (n)
A	12
B	4
C	7
D	3
E	8
F	2
H	4
I	9
S	13
G	0

Table no.-1

In this search example, we are using two lists which are OPEN and CLOSED Lists. Following are the iteration for traversing the above example.



Expand the nodes of S and put them in the CLOSED list

Initialization: Open [A, B], Closed [S]

Iteration 1: Open [A], Closed [S, B]

Iteration 2: Open [E, F, A], Closed [S, B] : Open [E, A], Closed [S, B, F]

Iteration 3: Open [I, G, E, A], Closed [S, B, F] : Open [I, E, A], Closed [S, B, F, G]

Therefore, the final solution path will be: S----> B----->F----> G

Analysis of algorithm

Time Complexity: The Greedy best first search's worst case time complexity is O(bm).

Space Complexity: The Greedy best first search's worst-case space complexity is O(bm).

A* search Algorithm

A* Search algorithm is one of the best and popular technique used in path-finding and graph traversals.

It using heuristic function h(n) for finding path.

It searches for shorter paths first, thus making it an optimal and complete algorithm. An optimal algorithm will find the least cost outcome for a problem, while a complete algorithm finds all the possible outcomes of a problem.

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).

Algorithm of A* search:

Step 1: Place the beginning node in the OPEN list as the first step.

Step 2: Determine whether or not the OPEN list is empty; if it is, return failure and stop.

Step 3: Choose the node with the shortest value of the evaluation function (g+h) from the OPEN list; if node n is the goal node, return success and stop; otherwise, return failure and continue.

Step 4: Expand node n and create all of its descendants, then place n in the closed list. Check whether n' is already in the OPEN or CLOSED list for each successor n'; if not, compute the evaluation function for n' and insert it in the Open list. Step 5: If node n' is already in the OPEN and CLOSED states, it should be attached to the back pointer, which represents the lowest g(n') value

Example:

The below example explains A*search with heuristic function.

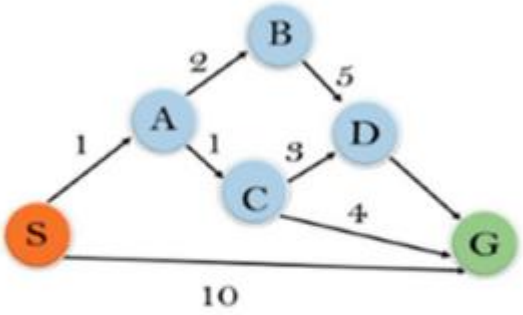
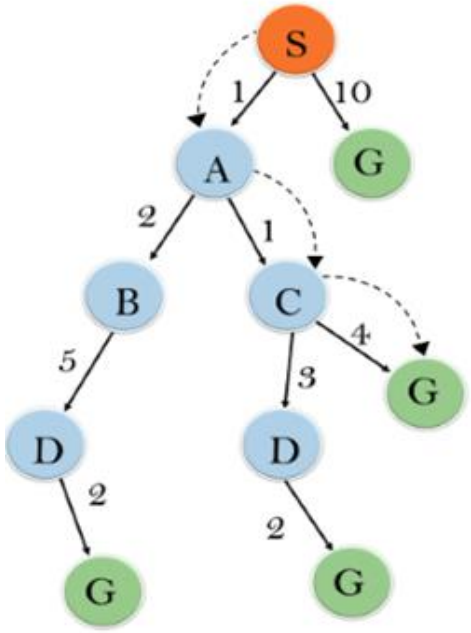


Fig-2

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

Table no-2



Solution:

Expand the nodes of S and put them in the CLOSED list

Initialization: Open [A, B], Closed [S] Iteration 1: Open [A], Closed [S, B]

Iteration 2: Open [E, F, A], Closed [S, B] : Open [E, A], Closed [S, B, F]

Iteration 3: Open [I, G, E, A], Closed [S, B, F] : Open [I, E, A], Closed [S, B, F, G]

Therefore, the final solution path will be: S----> B----->F---->

Analysis

Time Complexity: The Greedy best first search's worst case time complexity is O(bm).

Space Complexity: The Greedy best first search's worst-case space complexity is O(bm). Where m is the search space's maximum depth

Hill climbing

Hill climbing algorithm is a local search algorithm which continuously moves in the direction of increasing elevation/value to find the peak of the mountain or best solution to the problem.

And it terminates when it reaches a peak value where no neighbour has a higher value.

It is also called greedy local search as it only looks to its good immediate neighbour state and not beyond that.

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.

Algorithm

- **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:
    a. If it is goal state, then return success and quit.
    b. Else if it is better than the current state then assign new state as a current state.
    c. Else if not better than the current state, then return to step2.
- **Step 5:** Exit.

Drawback in hill climbing algorithm

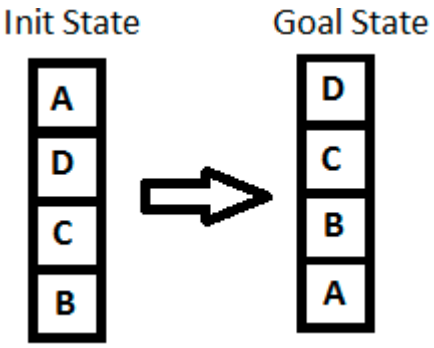
1. Local maximum

2. Plateau

3. Ridges

Example

Hill Climbing Algorithm can be categorized as an informed search. So we can implement any node-based search or problems like the n-queens problem using it. To understand the concept easily, we will take up a very simple example.

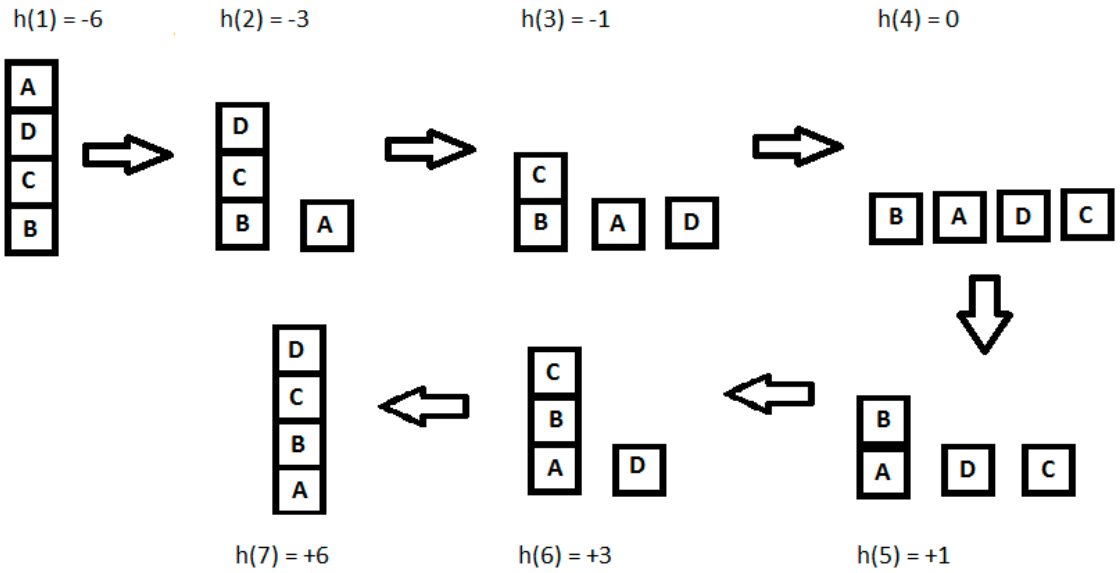


key point while solving any hill-climbing problem is to choose an appropriate heuristic function.

Let's define such function *h:*

***h(x)* = +1 for all the blocks in the support structure if the block is correctly positioned otherwise -1 for all the blocks in the support structure.**

Here, we will call any block correctly positioned if it has the same support structure as the goal state. As per the hill climbing procedure discussed earlier let's look at all the iterations and their heuristics to reach the target state:


h(1) = -6
h(2) = -3
h(3) = -1
h(4) = 0
A
D
C
B
D
C
B
A
C
B
A
D
B
A
D
C
B
A
D
C
D
C
B
A
C
B
A
D
B
A
D
C
h(7) = +6
h(6) = +3
h(5) = +1

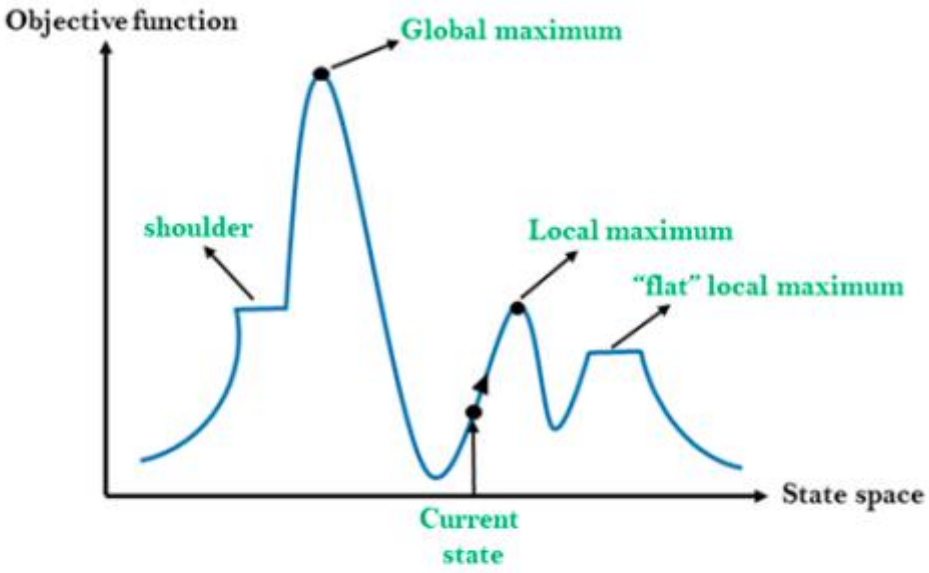

Objective function
Global maximum
shoulder
Local maximum
"flat" local maximum
State space
Current
state

Fig:-3

Conclusion

These algorithms always working in current local best solution. therefore, there is no guarantee of optimal solution and. all use where you have knowledge of you destination for example traveling, maybe it is not good in searching. And time complexity is not good so think before use

Reference

https://www.geeksforgeeks.org/introduction-hill-climbing-artificial-intelligence/

Hill Climbing Algorithm in AI - Javatpoint

www.googel.com

book reference: Tata McGraw-Hill publishing company limited