6.AO* Algorithm

Introduction

In an AND-OR graph AO* algorithm is an efficient method to explore a solution path. AO* algorithm works mainly based on two phases. First phase will find a heuristic value for nodes and arcs in a particular level. The changes in the values of nodes will be propagated back in the next phase.

Algorithm for AO*: In order to find solution in an AND-OR graph AO* algorithm workswell similar to best first search with an ability to handle the AND arc appropriately. The algorithm finds an optimal path from initial node by propagating the results like solution and change in heuristic value to the ancestors as in algorithm discussed below.

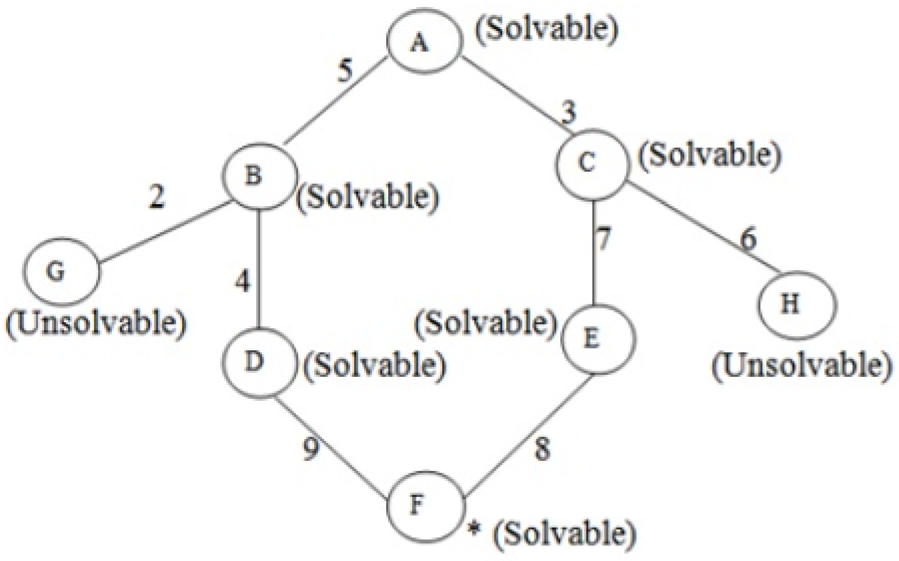
1. *Initialize the graph to start node*
2. *Negotiate the graph following the current path accumulating nodes that have not yet been expanded or solved*
3. *Then choose any of these nodes and expand it and if it has no successors call this value FUTILITY otherwise calculate only f' for each of the successors.*
4. *If f' is 0 then consider the node as SOLVED*
5. *Change the value of f' for the recently created node to display its successors by backpropagation.*
6. *Anywhere possible use the most assuring routes and if a node is marked as SOLVED then mark the parent node as SOLVED.*
7. *If starting node is SOLVED or a value greater than FUTILITY, stop, else repeat from 2.*

Implementation

Let us take an example and understand the AO* algorithm.

Step 1:

Let us take a graph containing 6 nodes A,B,C,D,E,F and the unsolvable nodes are G and H. Take A as starting node and place it into unexplored.



Unexplored=[A]

Explored=[None]

Step 2:

The children of A are B and C which are solvable.

Unexplored= [B, C]

Explored=[A]

Step 3:

The children of B and C are placed into Unexplored and B, C move to explored.

Unexplored= [G, D, E]

Explored= [A, B, C]

Step 4:

As G and H are unsolvable, so we are placing them directly into Explored and keep on exploring D and E.

Step 5:

By proceeding in this way, we reach our goal state which is F. It is success so we can exit.

Advantages:

1. It is an optimal algorithm. It traverses according to the ordering of the nodes.

Disadvantages:

1. Sometimes there are unsolvable nodes. In that case we can’t find the optimal path.

Pythton Code Implementation

```
def aostar(self, depth = 75):

        """Run AO* search."""

        #TODO: Implement ao star search.

        priotity_queue = PriorityQueue()

        h_val = self.start.manhattan_distance() +
self.start.hamming_distance()

        # g_val always is start.step

        f_val = h_val + self.start.step

        priotity_queue.push(self.start, f_val)

        visited = set()

        found = False


        while not priotity_queue.isEmpty():

            state = priotity_queue.pop()


            if state == self.goal:

                found = state

                break


            if state in visited or state.step > depth:

                continue


            visited.add(state)


            for s in state.next():
```

```
        h_val_s = s.manhattan_distance() + s.hamming_distance()
        f_val_s = h_val_s + s.step
        priotity_queue.push(s, f_val_s)

if found:
    self.print_path(found)
    print("Find solution")
else:
    print("No solution found")
```