5. AO* Heuristic Search

22/08/2019

INTRODUCTION

The Depth first search and Breadth first search given earlier for OR trees or graphs can be easily adopted by AND-OR graph. The main difference lies in the way termination conditions are determined, since all goals following an AND nodes must be realized; where as a single goal node following an OR node will do. For this purpose, we are using AO* algorithm.

Like A* algorithm here we will use two arrays and one heuristic function.

ALGORITHM OF AO* HEURISTIC

First, we define 2 arrays **CLOSE** and **OPEN**. “Open” contains those nodes which has already been traversed but are not marked as solved. “Close" contains all the nodes which has already been processed.

1. Place the starting node into unexplored array.
2. Next, we have to compute the most promising solution tree, say T0 .
3. Select a node ‘n’ which is member of both T 0 and unexplored array.
4. Remove ‘n’ from unexplored and place it in explored.
5. If ‘n’ is the terminal goal node then mark ‘n’ as solved and mark all its ancestors as solved.
6. If the starting node is marked as solved then it is ‘success’ and exit.
7. If it is ‘n’ is not a solvable node then it is a failure. If start node is not in its ancestor then also it is failure.
8. Expand ‘n’ and find all its successors and calculate the value of h(n) and push those nodes into unexplored.
9. Return to step 2.

IMPLEMENTATION USING DATA STRUCTURE

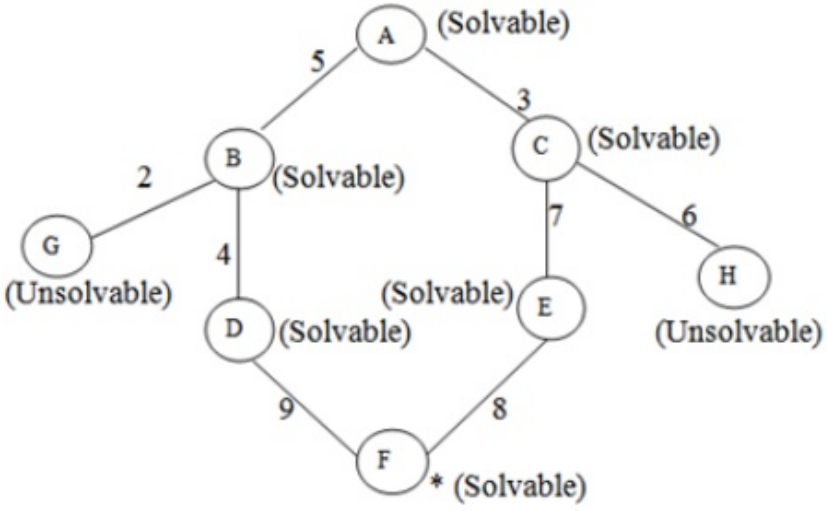
Let us take an example and understand the AO* algorithm.

Step 1: In the above graph, the solvable nodes are A, B, C, D, E, F and the unsolvable nodes are G, H. Take A as the starting node. So place A into OPEN.

OPEN = [A] and **CLOSE** = [None]

Step 2: The children of A are B and C which are solvable. So place them into OPEN and place A in CLOSE.

OPEN = [B, C] and **CLOSE** = [A]



Step 3: The children of B and C are to be placed into OPEN and also B and C are removed from OPEN and placed into CLOSE.

OPEN = [G, D, E] and **CLOSE** = [A, B, C]

Step 4: As G and H are unsolvable, so we are placing them directly into CLOSE and keep on exploring D and E.

OPEN = [F] and **CLOSE** = [D, E, A, B, C, G, H]

Step 5: By proceeding in this way, we reach our goal state which is F. It is success, so we can exit.

PYTHON IMPLEMENTATION OF BFS

```
def and_or_graph_search(problem):
    """Used when the environment is nondeterministic and completely observable.
    Contains OR nodes where the agent is free to choose any action.
    After every action there is an AND node which contains all possible states
    the agent may reach due to stochastic nature of environment.
    Returns a conditional plan to reach goal state,
    or failure if the former is not possible."""

    # functions used by and_or_search
    def or_search(state, problem, path):
        """returns a plan as a list of actions"""
        if problem.goal_test(state):
            return []
        if state in path:
            return None
        for action in problem.actions(state):
            plan = and_search(problem.result(state, action),
                              problem, path + [state, ])
            if plan is not None:
                return [action, plan]
```

```
    def and_search(states, problem, path):
        """Returns plan in form of dictionary where we
        take action plan[s] if we reach state s."""
        plan = {}
        for s in states:
            plan[s] = or_search(s, problem, path)
            if plan[s] is None:
                return None
        return plan

    # body of and or search
    return or_search(problem.initial, problem, [])
```

ADVANTAGES AND DISADVANTAGES OF A * HEURISTICS

- It is an optimal algorithm.
- If traverse according to the ordering of nodes. It can be used for both OR and AND graph.
- Sometimes for unsolvable nodes, it can’t find the optimal path. Its complexity is than other algorithms.