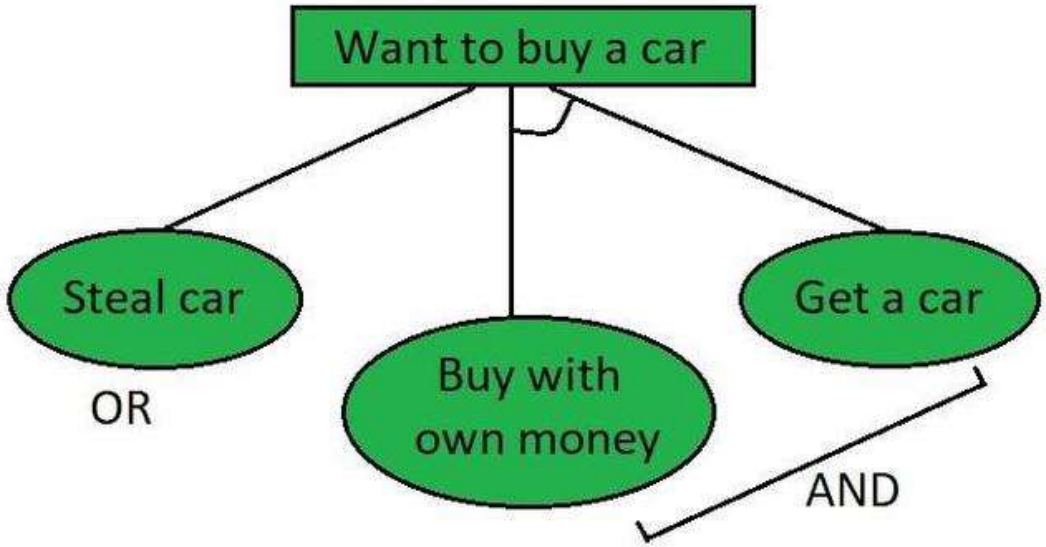
AO* algorithm – Artificial intelligence

Best-first search is what the AO* algorithm does. The AO* method **divides** any given difficult **problem into a smaller group** of problems that are then resolved **using the AND-OR** graph concept. AND OR graphs are specialized graphs that are used in problems that can be divided into smaller problems. The AND side of the graph represents a set of tasks that must be completed to achieve the main goal, while the OR side of the graph represents different methods for accomplishing the same main goal.



AND-OR Graph

In the above figure, the buying of a car may be broken down into smaller problems or tasks that can be accomplished to achieve the main goal in the above figure, which is an example of a simple AND-OR graph. The other task is to either steal a car that will help us accomplish the main goal or use your own money to purchase a car that will accomplish the main goal. The AND symbol is used to indicate the AND part of the graphs, which refers to the need that all sub problems containing the AND to be resolved before the preceding node or issue may be finished.

The start state and the target state are already known in the knowledge-based search strategy known as the AO* algorithm, and the best path is identified by heuristics. The informed search technique considerably reduces the algorithm's time complexity. The AO* algorithm is far more effective in searching AND-OR trees than the A* algorithm.

Working of A* algorithm:

The evaluation function in AO* looks like this:
f(n) = g(n) + h(n)
f(n) = Actual cost + Estimated cost
here,

f(n) = The actual cost of traversal.
g(n) = the cost from the initial node to the current node.
h(n) = estimated cost from the current node to the goal state.

AO* Algorithm:

Our real-life situations can't be exactly decomposed into either AND tree or OR tree but is always a combination of both. So, we need an AO* algorithm where O stands for 'ordered'. The AO* algorithm represents a part of the search graph that has been explicitly generated so far. AO* algorithm is given as follows:

Step-1: Create an initial graph with a single node (start node).

Step-2: Transverse the graph following the current path, accumulating node that has not yet been expanded or solved.

Step-3: Select any of these nodes and explore it. If it has no successors then call this value- FUTILITY else calculate f'(n) for each of the successors.

Step-4: If f'(n)=0, then mark the node as SOLVED.

Step-5: Change the value of f'(n) for the newly created node to reflect its successors by back propagation.

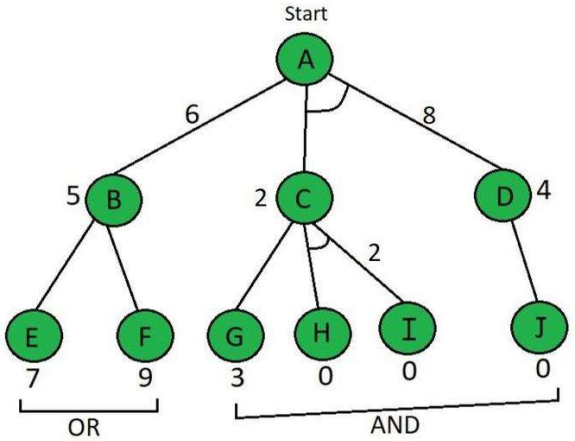
Step-6: Whenever possible use the most promising routes, If a node is marked as SOLVED then mark the parent node as SOLVED.

Step-7: If the starting node is SOLVED or value is greater than **FUTILITY** then stop, else repeat from Step-2.

Difference between the A* Algorithm and AO* algorithm

- A* algorithm and AO* algorithm both works on the **best first search**.
- They are both **informed search** and works on given heuristics values.
- **A*** always **gives** the **optimal solution** but AO* doesn't guarantee to give the optimal solution.
- Once AO* got a solution **doesn't explore** all possible paths but A* explores all paths.
- When compared to the A* algorithm, the AO* algorithm uses **less memory.**
- Opposite to the A* algorithm, the AO* algorithm cannot go into an endless **loop.**

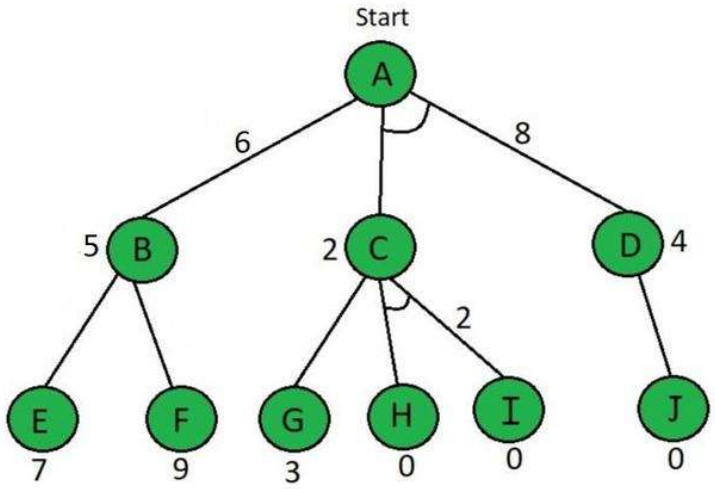
Example:



AO Algorithm – Question tree*

Here in the above example below the Node which is given is the heuristic value i.e h(n). Edge length is considered as 1.

Step 1:

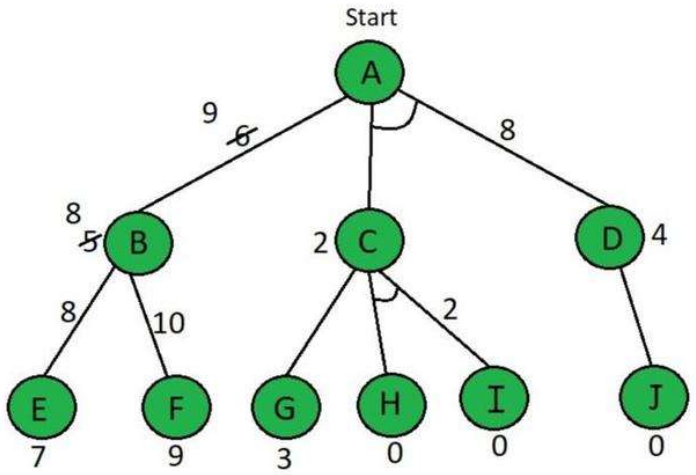


AO Algorithm (Step-1)*

With help of **f(n) = g(n) + h(n)** evaluation function,
Start from node A,
f(A⇢B) = g(B) + h(B)
= 1 + 5 = 6 ……here **g(n)=1** is taken by default for path cost

f(A⇢C+D) = g(c) + h(c) + g(d) + h(d)
= 1 + 2 + 1 + 4 = 8 ……here we have added **C & D** because they are in **AND**
So, by calculation **A⇢B** path is chosen which is the minimum path, i.e **f(A⇢B)**

Step 2:



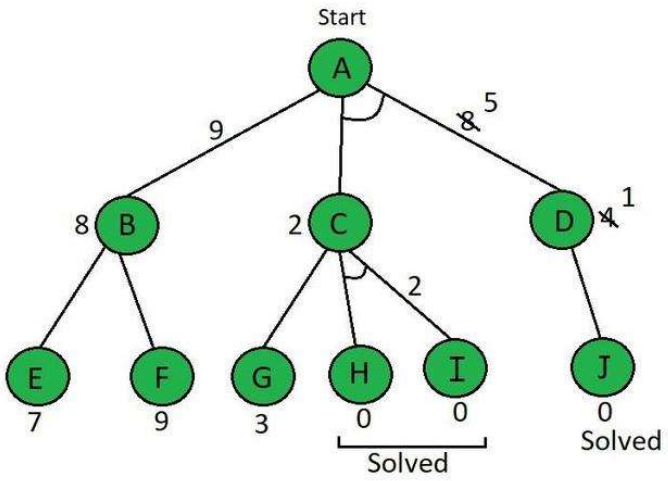
AO Algorithm (Step-2)*

According to the answer of step 1, explore node B.Here the value of E & F are calculated as follows,
f(B⇢E) = g(e) + h(e) = 1 + 7 = 8
f(B⇢F) = g(f) + h(f) = 1 + 9 = 10

So, by above calculation B⇢E path is chosen which is minimum path, i.e **f(B⇢E)** because **B's** heuristic value is different from its actual value. The heuristic is updated and the minimum cost path is selected. The minimum value in our situation is **8**. Therefore, the heuristic for **A** must be updated due to the change in **B's** heuristic. So we need to calculate it again.

f(A⇢B) = g(B) + updated h(B) = 1 + 8 = 9
We have Updated all values in the above tree.

Step 3:



AO Algorithm (Step-3)*

By comparing **f(A⇢B)** & **f(A⇢C+D)**
f(A⇢C+D) is shown to be **smaller**. i.e 8 < 9
Now explore f(A⇢C+D)
So, the current node is **C**

f(C⇢G) = g(g) + h(g) = 1 + 3 = 4

f(C⇢H+I) = g(h) + h(h) + g(i) + h(i) = 1 + 0 + 1 + 0 ……here we have added **H & I** because they are in **AND**
= 2

f(C⇢H+I) is selected as the path with the lowest cost and the heuristic is also left unchanged because it matches the actual cost. Paths H & I are solved because the heuristic for those paths is **0,** but Path **A⇢D** needs to be calculated because it has an **AND**.

f(D⇢J) = g(j) + h(j) = 1 + 0 = 1
Thus, the heuristic of node D needs to be updated to 1.

f(A⇢C+D) = g(c) + h(c) + g(d) + h(d) = 1 + 2 + 1 + 1 = 5

As we can see that path **f(A⇢C+D)** is get solved and this **tree has become a solved tree** now.
In simple words, the main flow of this algorithm is that we have to find **firstly level 1st** heuristic value and **then level 2nd** and after that **update the values** with going **upward** means towards the root node.
In the above tree diagram, we have updated all the values.

AO* Performance:

AO* is complete, meaning it finds a solution, if there is any, and does not fall into an infinite loop. Moreover, the AND feature in this algorithm reduces the demand for memory.
The space complexity comes in polynomial order, while the time complexity is $O(b^d)$, where b stands for branching and d is the maximum depth or number of levels in the search tree.

Real-Life Applications of AO* algorithm:

Vehicle Routing Problem:
The vehicle routing problem is determining the shortest routes for a fleet of vehicles to visit a set of customers and return to the depot, while minimizing the total distance traveled and the total time taken. The AO* algorithm can be used to find the optimal routes that satisfy both objectives.
Portfolio Optimization:
Portfolio optimization is choosing a set of investments that maximize returns while minimizing risks. The AO* algorithm can be used to find the optimal portfolio that satisfies both objectives, such as maximizing the expected return and minimizing the standard deviation.

Pseudocode for AO* Algorithm:

As we have seen in the above example, the process goes into a cycle of forward and backward propagation until there is no more change in the minimum cost path. To simplify, we did not go into details of the backpropagation since it follows the same rules of forward propagation but in the opposite direction. Let's see the pseudocode below:

Algorithm 1: Pseudocode of AO* Algorithm

```
Data: Graph, StartNode
Result: The minimum cost path from StartNode to GoalNode
CurrentNode ← StartNode
while There is a new path with lower cost from StartNode to the
 GoalNode do
    calculate the cost of path from the current node to the goal
     node through each of its successor nodes;
    if the successor node is connected to other successor nodes by
     AND-ARCS then
        sum up the cost of all paths in the AND-ARC;
        return the total cost;
    else
        calculate the cost of the single path in the OR side;
        return the single cost;
    end
    find the minimum cost path
    CurrentNode ← SuccessorNodeOfMinimumCostPath
    if CurrentNode has no successor node then
        do the backpropagation and correct the estimated costs;
        CurrentNode ← StartNode
        return CurrentNode, New estimated costs;
    else
        return null;
    end
    return The minimum cost path;
end
```

Conclusion:

AO* as a sample of best-first search algorithms. We noticed that the strength of this algorithm comes from a divide and conquer strategy. The AND feature brings all the tasks of the same goal under one umbrella and reduces the complexities. However, such a benefit comes at the cost of losing optimality.

Bibliography:

https://www.geeksforgeeks.org/ao-algorithm-artificial-intelligence/
http://www.facweb.iitkgp.ac.in/~pallab/ai.slides/lec4.ppt
https://sites.astro.caltech.edu/~george/aybi199/AMooreTutorials/astar.ppt
https://www.baeldung.com/cs/ao-star-algorithm
https://webeduclick.com/ao-algorithm-in-artificial-intelligence/