



Fr. Conceicao Rodrigues College of Engineering
Fr. Agnel Ashram, Bandstand, Bandra (W), Mumbai - 400050

Department of Computer Engineering
Academic Term II: 23-24

Class: B.E (Computer), Sem – VI

Subject Name: Artificial Intelligence

Student Name: Siddhesh Pradhan

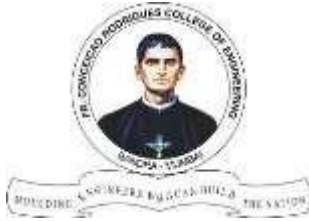
Roll No: 9632

Practical No:	6
Title:	Implementation of AO* algorithm
Date of Performance:	
Date of Submission:	

Rubrics for Evaluation:

Sr. No	Performance Indicator	Excellent	Good	Below Average	Marks
1	On time Completion & Submission (01)	01 (On Time)	NA	00 (Not on Time)	
2	Logic/Algorithm Complexity analysis (03)	03(Correct)	02(Partial)	01 (Tried)	
3	Coding Standards (03): Comments/indentation/Naming conventions Test Cases /Output	03(All used)	02 (Partial)	01 (rarely followed)	
4	Post Lab Assignment (03)	03(done well)	2 (Partially Correct)	1(submitted)	
Total					

Signature of the Teacher:



Fr. Conceicao Rodrigues College of Engineering
Fr. Agnel Ashram, Bandstand, Bandra (W), Mumbai - 400050

Experiment No: 6

Title: Implementation of AO* algorithm

Objective: To study AO* algorithm and implement it in an efficient manner

Theory:

AO* Algorithm basically based on problem decomposition (Breakdown problem into small pieces). Basically, we will calculate the **cost function** here ($F(n) = G(n) + H(n)$)

H: heuristic/ estimated value of the nodes. and **G:** actual cost or edge value (here unit value).

Here we have taken the **edges value 1**, meaning we have to focus solely on the **heuristic value**.

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 backpropagation.

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.

CODE:

```
import heapq
```

```
class Graph:
```

```
    def __init__(self, vertices):
```

```
        self.vertices = vertices
```

```
        self.adj_list = {vertex: [] for vertex in range(vertices)}
```



Fr. Conceicao Rodrigues College of Engineering
Fr. Agnel Ashram, Bandstand, Bandra (W), Mumbai - 400050

```
def add_edge(self, u, v, w):

    self.adj_list[u].append((v, w))


def a_star_search(self, start):

    # Initialize f' value for each vertex

    f_values = [float('inf')] * self.vertices

    f_values[start] = 0


    # Initialize heap for open list

    open_list = [(0, start)]


    # Iterate until open list is empty

    while open_list:

        # Pop node with minimum f' value

        f_prime, current = heapq.heappop(open_list)


        # Mark node as SOLVED if f' value is 0

        if f_prime == 0:

            print(f"Node {current} is SOLVED.")

            continue


        # Explore each neighbor of current node

        for neighbor, edge_weight in self.adj_list[current]:

            # Calculate f' value for the neighbor
```



```
f_prime_neighbor = max(f_values[current], edge_weight)
```

```
# Update f' value if it's improved
```

```
if f_prime_neighbor < f_values[neighbor]:
```

```
    f_values[neighbor] = f_prime_neighbor
```

```
    heapq.heappush(open_list, (f_prime_neighbor, neighbor))
```

```
print("Algorithm execution complete.")
```

```
def draw_graph(self):
```

```
    # Code to visualize the graph
```

```
    pass
```

```
def main():
```

```
    # Example usage
```

```
    graph = Graph(5)
```

```
    graph.add_edge(0, 1, 2)
```

```
    graph.add_edge(0, 2, 1)
```

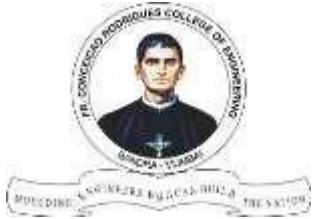
```
    graph.add_edge(1, 3, 3)
```

```
    graph.add_edge(2, 4, 4)
```

```
    graph.a_star_search(0)
```

```
if __name__ == "__main__":
```

```
    main()
```



OUTPUT:

```
Node 0 is SOLVED.  
Algorithm execution complete.
```

Post Lab Assignment:

1. What is the difference between A* and AO* algorithm?
2. Why AO* algorithm only works when heuristic values are underestimated?

^ AI Expt 6 Postlab

1. What is difference between A^* and AO^*

Ans:-

- | A^* | AO^* |
|--|---|
| ① Not designed for handling changes. | ① Specially designed to adapt changes. |
| ② Primarily uses AND operation. | ② Uses both AND & OR. |
| ③ Consumes Less Memory. | ③ Consumes More Memory. |
| ④ Well-suited for static environments. | ④ Well-suited for dynamic environments. |
| ⑤ Explores Less no. of nodes. | ⑤ Explores More no. of nodes. |

2. Why AO^* algorithm only works when heuristic values are underestimated?

Ans:- The admissibility property of the heuristic function in AO^* is essential for ensuring the algorithm's correctness and efficiency. It guarantees optimality by preventing the algorithm from prematurely discarding potentially optimal paths. Additionally, it ensures completeness by guaranteeing that AO^* will find a solution if one exists in the search space. Finally, an underestimated heuristic helps AO^* avoid misleading information, leading to more efficient exploration of the search space.

