	Marathwada Mitra Mandal's	
	Institute of Technology, Lohgaon Pune - 47	
	Department of Artificial Intelligence and Data Science	

Semester -I A.Y.2025-26 Sub.: - Artificial Intelligence Lab Class: SE

---

1. Aim

To implement the **A*** (A-star) algorithm for solving AI search problems using the Graph Search method.

2. Objectives

- To understand the working of heuristic search in AI.
- To explore the use of A* algorithm in pathfinding and graph traversal problems.
- To analyze the efficiency of informed search strategies compared to uninformed search.

3. Theory

The **A*** algorithm is an **informed search strategy** that combines the strengths of **Uniform Cost Search (UCS)** and **Greedy Best-First Search**.

It uses both the actual cost to reach a node (**g(n)**) and the estimated cost from that node to the goal (**h(n)**).

Evaluation Function

$f(n)=g(n)+h(n)$

- **g(n):** Cost from the start node to current node *n*.
- **h(n):** Heuristic estimate of the cost from *n* to goal.
- **f(n):** Estimated total cost of the path through *n*.

Applications

- Pathfinding in maps (GPS navigation).
- Game AI (shortest pathfinding for NPCs).
- Robot navigation.

4. Algorithm (Steps of A*)

1. Initialize the **open list** with the start node.
2. Initialize the **closed list** as empty.
3. Repeat until the goal is found or open list is empty:

○ Select the node with the **lowest f(n)** from the open list.

○ If this node is the goal, return success (trace back the path).

○ Otherwise, expand the node:

■ For each successor, calculate g(n), h(n), and f(n).

■ If the successor is not in open/closed lists, add it to the open list.

■ If it is already present with a higher cost, update its values.

○ Move the expanded node to the closed list.

4. If the open list becomes empty and the goal is not found → return failure.

5. Python Implementation

```
# Example graph
graph = {
    'S': {'A': 1, 'B': 4},
    'A': {'B': 2, 'C': 5, 'D': 12},
    'B': {'C': 2},
    'C': {'D': 3, 'G': 7},
    'D': {'G': 2},
    'G': {}
}
```

```
# Heuristic values
heuristics = {
    'S': 7, 'A': 6, 'B': 4,
    'C': 2, 'D': 1, 'G': 0
}
# Run A*
start, goal = 'S', 'G'
```

6. Sample Output

7. Observations

- A* algorithm expands fewer nodes than BFS/DFS because it uses heuristics.
- Optimality depends on correctness of the heuristic function.
- In the given example, the path found is the shortest with minimum cost.

8. Conclusion

The **A*** algorithm was successfully implemented for graph-based search problems. It demonstrates how heuristic guidance improves search efficiency and guarantees optimal solutions if heuristics are admissible.

Example code :

```
def a_star(graph, heuristics, start, goal):
    open_list = set([start])
    closed_list = set()

    # Stores g(n) values (cost from start to node)
    g = {node: float('inf') for node in graph}
    g[start] = 0

    # Stores f(n) = g(n) + h(n)
    f = {node: float('inf') for node in graph}
    f[start] = heuristics[start]

    # To reconstruct the path
    parents = {start: None}

    while open_list:
        # Select node with lowest f(n)
        current = min(open_list, key=lambda node: f[node])

        if current == goal:
            # Reconstruct path
            path = []
            while current:
                path.append(current)
                current = parents[current]
            path.reverse()
            return path, g[goal]
```

```
        open_list.remove(current)
        closed_list.add(current)

        for neighbor, cost in graph[current].items():
            if neighbor in closed_list:
                continue

            tentative_g = g[current] + cost

            if neighbor not in open_list:
                open_list.add(neighbor)
            elif tentative_g >= g[neighbor]:
                continue

            # Update parent, g, and f values
            parents[neighbor] = current
            g[neighbor] = tentative_g
            f[neighbor] = g[neighbor] + heuristics[neighbor]

    return None, float('inf')


# Example graph
graph = {
    'Home': {'School': 50, 'Garden': 40 , 'Bank': 45},
    'School': {'Post office': 59, 'Railway station': 75,},
    'Garden': {'Railway station': 72},
    'Bank': {'Police station': 60, },
    'Police station': {'University': 28},
    'Railway station': {'University': 40},
    'University':{}
}

# Heuristic values
```

```python
heuristics = {
    'Home': 120, 'Bank': 80, 'Garden': 100,
    'School': 70, 'Railway station': 20, 'Police station': 26 , 'Post office': 110 , 'University': 0
}


start, goal = 'Home', 'University'
path, cost = a_star(graph, heuristics, start, goal)

if path:
    print(f"Path found: {' -> '.join(path)} with total cost: {cost}")
else:
    print("No path found.")
```

Output :