**Name: Aditya Parade  
Roll No: 381047  
PRN: 22311577**

**Assignment 4**

**Problem Statement:** To Implement A\* Algorithm for an application.

**1. Introduction**

The **A\*** (**A-star**) algorithm is one of the most widely used and efficient **pathfinding** and **graph traversal** algorithms in Artificial Intelligence. It is designed to find the **shortest path** between two nodes in a graph or grid, considering both the **cost of reaching a node** and an **estimated cost to the goal**.

The A\* algorithm is often applied in **navigation systems, robotics, games (like maze or map navigation), and route planning applications**. It combines the strengths of **Dijkstra’s Algorithm** (optimal pathfinding) and **Greedy Best-First Search** (fast goal-oriented searching), achieving a balance between speed and accuracy.

For example, in a map or maze-solving application, A\* efficiently finds the shortest path from a start point to a destination while avoiding obstacles.

**2. Objective**

The main objectives of this experiment are:

* To understand and implement the **A\*** algorithm for solving real-world pathfinding problems.
* To apply A\* for finding the **shortest route** between a start and goal node in a grid or map.
* To explore how the **heuristic function** affects the performance and accuracy of the algorithm.
* To compare A\* with other uninformed search techniques (like BFS and Dijkstra’s) in terms of efficiency and optimality.

**3. Theory**

**3.1 Overview of A\* Algorithm**

The **A\*** algorithm is an **informed search algorithm** that uses both actual cost and heuristic information to find an optimal path efficiently.  
It evaluates each node using the function:

Where:

* **g(n)** → The actual cost from the start node to the current node *n*.
* **h(n)** → The heuristic (estimated) cost from *n* to the goal node.
* **f(n)** → The total estimated cost of the cheapest solution through *n*.

The algorithm selects the node with the **lowest f(n)** value for expansion at each step.

**3.2 Working Steps**

1. Initialize an **open list** (nodes to be evaluated) and a **closed list** (nodes already evaluated).
2. Add the start node to the open list with **f(start) = h(start)**.
3. While the open list is not empty:
   * Select the node *n* with the lowest *f(n)*.
   * If *n* is the goal node, stop — the path has been found.
   * Otherwise, generate all possible neighbors of *n*.
   * For each neighbor:
     + Calculate *g*, *h*, and *f* values.
     + If the neighbor is not in the open or closed list, add it to the open list.
   * Move *n* from the open list to the closed list.
4. Repeat until the goal is reached or the open list becomes empty.

**3.3 Heuristic Function (h(n))**

The heuristic function provides an estimate of the cost to reach the goal from the current node.  
Common heuristic functions include:

* **Manhattan Distance:**

(Used in grid-based maps with 4-direction movement)

* **Euclidean Distance:**

(Used when diagonal movement is allowed)

* **Chebyshev Distance:**

(Used in 8-direction movement)

The heuristic must be **admissible**, meaning it never overestimates the true cost.

**3.4 Pseudocode for A\***

function A\_Star(start, goal):

open\_list ← {start}

closed\_list ← {}

g(start) ← 0

f(start) ← h(start)

while open\_list is not empty:

current ← node in open\_list with lowest f

if current = goal:

return reconstruct\_path(current)

remove current from open\_list

add current to closed\_list

for each neighbor of current:

tentative\_g ← g(current) + distance(current, neighbor)

if neighbor in closed\_list and tentative\_g ≥ g(neighbor):

continue

if neighbor not in open\_list or tentative\_g < g(neighbor):

parent(neighbor) ← current

g(neighbor) ← tentative\_g

f(neighbor) ← g(neighbor) + h(neighbor)

if neighbor not in open\_list:

add neighbor to open\_list

return failure

**3.5 Applications of A\***

* **Pathfinding in Maps and Games** – used in Google Maps, GPS navigation, and AI game characters.
* **Robot Motion Planning** – helps robots find optimal paths avoiding obstacles.
* **Network Routing** – finding efficient data paths in network graphs.
* **Puzzle Solving** – used in 8-puzzle or 15-puzzle to find the shortest sequence of moves.

**3.6 Example – Grid Pathfinding**

In a 5×5 grid:

* Start = (0, 0)
* Goal = (4, 4)
* Obstacles in some cells  
  The A\* algorithm finds the shortest path while avoiding obstacles by evaluating both distance traveled (g) and estimated distance to goal (h).

**4. Conclusion**

In this experiment, we successfully implemented the **A\*** algorithm for a pathfinding application.  
The algorithm efficiently found the **shortest and optimal path** by combining **actual cost (g)** and **heuristic estimation (h)**.

Key takeaways:

* A\* is **complete**, **optimal**, and **efficient** when the heuristic is admissible.
* The choice of heuristic greatly affects performance — a good heuristic reduces computation time.
* A\* outperforms uninformed search algorithms like BFS and DFS in complex pathfinding problems.

This experiment demonstrates the practical power of **heuristic-based search** in AI and its wide range of applications in navigation, robotics, and decision-making systems.