

## Assignment 4: Implement A\* Algorithm for an Application

Name: Vivek Mule  
Roll: 381072  
Div: A Batch 3  
PRN: 22420145

### Problem Statement

The objective of this assignment is to implement the A\* search algorithm for a pathfinding application. This involves developing a method to find the most efficient path from a starting point to a goal while considering various obstacles and costs associated with movement.

### Objectives

- Understand the principles of heuristic search.
- Implement the A\* algorithm to find the optimal path in a grid or graph-based environment.

### Theory

#### What is the A\* Algorithm?

A\* is a widely used heuristic search algorithm that combines aspects of both Dijkstra's algorithm and Greedy Best-First Search. It efficiently finds the least-cost path from a start node to a goal node by considering both the actual cost to reach a node and an estimated cost to reach the goal.

### Methodology

#### 1. Define a Heuristic Function:

- The heuristic function  $h(n)$  estimates the cost from the current node  $n$  to the goal node. Common heuristics for grid-based pathfinding include:
  - **Manhattan Distance:**  $h(n) = |x_1 - x_2| + |y_1 - y_2|$  (suitable for grid movements where diagonal moves are not allowed).
  - **Euclidean Distance:**  $h(n) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$  (suitable for grid movements that allow diagonal moves).

#### 2. Explore Nodes Based on Cost:

- Each node in the search space maintains two costs:
  - $g(n)$ : The cost to reach the node from the start node.
  - $f(n) = g(n) + h(n)$ : The total estimated cost of the cheapest solution through node  $n$ .

#### 3. Continue Until the Goal is Reached:

- Initialize the open list with the start node and the closed list as empty.
- While there are nodes to explore:
  - Extract the node with the lowest  $f(n)$  from the open list.

- If this node is the goal node, backtrack to find the path.
- Otherwise, generate its successors, calculate their costs, and update their lists accordingly.
- Move the current node to the closed list to prevent re-exploration.

## Working Principle / Algorithm

Here's a simple outline of the A\* algorithm:

1. **Initialize the Open and Closed Lists:**
  - Start with the initial node, add it to the open list.
2. **While the Open List is Not Empty:**
  - Choose the node with the lowest  $f(n)$  from the open list.
  - If the chosen node is the goal, reconstruct the path and terminate.
  - Generate each of its neighboring nodes.
    - Calculate  $g(n)$  for each neighbor.
    - If a neighbor is in the closed list and the new path is better, update its cost.
    - If a neighbor is not in either list, add it to the open list.
3. **Path Reconstruction:**
  - Trace back from the goal node to the start node using parent pointers or a path list.

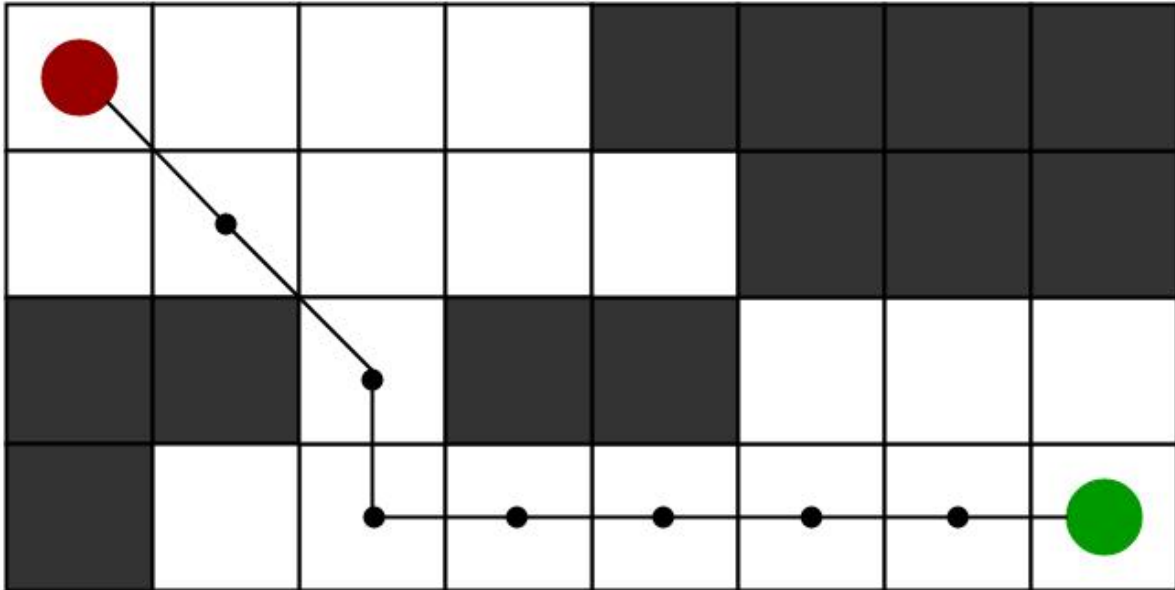
## Advantages

- **Optimality:** A\* guarantees the shortest path if the heuristic used is admissible (i.e., it never overestimates the cost to reach the goal).
- **Flexibility:** A\* can be adapted with different heuristics to fit various types of pathfinding problems.

## Disadvantages / Limitations

- **Memory Usage:** A\* can be memory-intensive for large search spaces, as it stores all generated nodes in the open list.
- **Performance:** The performance can degrade significantly if the heuristic is not well-designed or if the search space is too large.

## Diagram



## Conclusion

The A\* algorithm is an efficient search method that balances exploration of the search space with heuristic estimation to find optimal paths. Its flexibility and optimality make it a popular choice for various applications, including robotics, game development, and navigation systems.