KIET GROUP OF INSTITUTIONS

CREDIT SCORE PREDICTION USING MACHINE LEARNING

Submitted by:
Tushar Sharma
Roll No: 46
Section: D
B.Tech CSE-AI

Under the guidance of:
Abhishek Shukla
Department of Computer Science & Engineering (AI)

Introduction

The *A (A-star) algorithm** is a widely used pathfinding and graph traversal algorithm, known for its efficiency in finding the shortest path between two points. It combines the advantages of **Dijkstra's Algorithm** (which ensures the shortest path) and **Greedy Best-First Search** (which improves speed using heuristics).

A* uses a cost function:

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

where:

- g(n)g(n)g(n) is the actual cost from the start node to node nnn.
- h(n)h(n)h(n) is the estimated cost (heuristic) from node nnn to the goal.

By intelligently choosing which node to explore next, A* ensures optimal and efficient pathfinding, making it widely used in **robotics, game development, GPS navigation, and AI applications**.

Methodology

1. **Initialize** an open list that keeps track of nodes to be explored, starting with the initial node.
2. **Expand nodes**: Select the node with the lowest cost function , where:
    - is the actual cost from the start node to node .
    - is the estimated heuristic cost from node to the goal.
3. **Generate successors**: Compute their -scores and add them to the open list.
4. **Update paths**: If a shorter path to a node is found, update the parent reference and recalculate .
5. **Repeat** the process until the goal node is reached or the open list is empty.
6. **Trace back the path** from the goal node to reconstruct the solution.

Code

```
import heapq


class Node:
    def __init__(self, position, parent=None):
        self.position = position
        self.parent = parent
        self.g = 0  # Cost from start to this node
        self.h = 0  # Heuristic (estimated cost to goal)
```

```python
        self.f = 0  # Total cost

    def __lt__(self, other):
        return self.f < other.f

def heuristic(a, b):
    return abs(a[0] - b[0]) + abs(a[1] - b[1])  # Manhattan distance

def a_star(grid, start, goal):
    open_list = []
    closed_set = set()
    start_node = Node(start)
    goal_node = Node(goal)
    heapq.heappush(open_list, start_node)

    while open_list:
        current_node = heapq.heappop(open_list)

        if current_node.position == goal_node.position:
            path = []
            while current_node:
                path.append(current_node.position)
                current_node = current_node.parent
            return path[::-1]  # Return reversed path

        closed_set.add(current_node.position)

        for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1)]:  # Move in
4 directions
            neighbor_pos = (current_node.position[0] + dx,
current_node.position[1] + dy)
            if (neighbor_pos[0] < 0 or neighbor_pos[1] < 0 or
                neighbor_pos[0] >= len(grid) or neighbor_pos[1] >=
len(grid[0]) or
                grid[neighbor_pos[0]][neighbor_pos[1]] == 1 or  #
Obstacle check
                neighbor_pos in closed_set):
                continue

            neighbor_node = Node(neighbor_pos, current_node)
            neighbor_node.g = current_node.g + 1
            neighbor_node.h = heuristic(neighbor_pos,
goal_node.position)
            neighbor_node.f = neighbor_node.g + neighbor_node.h
```

```
            if any(open_node.position == neighbor_node.position and
open_node.f <= neighbor_node.f for open_node in open_list):
                continue

            heapq.heappush(open_list, neighbor_node)

    return None  # No path found

# Example usage
grid = [
    [0, 1, 0, 0, 0],
    [0, 1, 0, 1, 0],
    [0, 0, 0, 1, 0],
    [0, 1, 1, 1, 0],
    [0, 0, 0, 0, 0]
]
start = (0, 0)
goal = (0,4)
path = a_star(grid, start, goal)
print("Path:", path)
```

Output/Result

Below is the screenshot of the topic

```
# Example usage
grid = [
    [0, 1, 0, 0, 0],
    [0, 1, 0, 1, 0],
    [0, 0, 0, 1, 0],
    [0, 1, 1, 1, 0],
    [0, 0, 0, 0, 0]
]
start = (0, 0)
goal = (0,4)
path = a_star(grid, start, goal)
print("Path:", path)

Path: [(0, 0), (1, 0), (2, 0), (2, 1), (2, 2), (1, 2), (0, 2), (0, 3), (0, 4)]
```

References/Credits

- A* Algorithm Explanation: Wikipedia
- Python Implementation: Open-source projects

Developed by: Tushar sharma(Roll No: 46, Section D)