AI & NN Assignment

Praneesh R V
CB.SC.U4CYS23036

A* Pathfinding Algorithm

Aim

To develop a Python program that implements the A* (A-Star) search algorithm to determine the most efficient path between a starting point and a destination within a given graph structure.

Description

The A* search algorithm is a widely used and highly effective pathfinding method that combines the strengths of Dijkstra's algorithm and greedy best-first search. It is particularly useful for finding the shortest path in weighted graphs by considering both the actual cost of reaching a node and a heuristic estimate of the cost from that node to the goal.

The algorithm evaluates each node 'n' using the following cost function:

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

Where:

$g(n)$ represents the actual cost from the start node to the current node n.

$h(n)$ represents the estimated cost (heuristic) from node n to the goal node.

By balancing these two factors, A* efficiently prioritizes nodes that are likely to lead to the optimal path while avoiding unnecessary exploration of less promising paths.

Algorithm

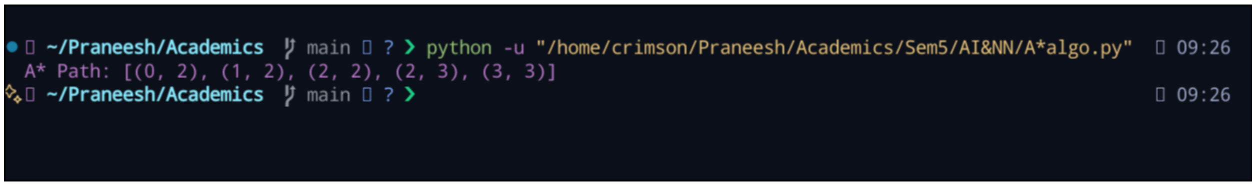
1. **Initialization**
    - Insert the starting node into the open list (a priority queue based on the lowest `f(n)` value).
2. **Loop Until Open List is Empty**
    - Select the node from the open list with the lowest `f(n)` value.
    - If the selected node is the destination, reconstruct the path and return it.
    - Move the selected node to the closed list (indicating it has been evaluated).
3. **Process Each Neighbor of the Current Node**
    - If the neighbor has already been evaluated (i.e., present in the closed list), skip it.
    - Calculate the tentative g(n) for the neighbor.
    - If this path to the neighbor is better (i.e., has a lower `g(n)`):
        - Update the neighbor's `g(n)` and `f(n)` values.
        - Record the current node as the neighbor's parent (for path reconstruction).
        - If the neighbor is not in the open list, add it.
4. **Failure Condition**

- If the open list becomes empty and the destination has not been reached, conclude that no valid path exists.

Python code

```python
from queue import PriorityQueue

def heuristic(pos1, pos2):
   return abs(pos1[0] - pos2[0]) + abs(pos1[1] - pos2[1])

def a_star_search(grid, start, goal):
   rows, cols = len(grid), len(grid[0])

   frontier = PriorityQueue()
   frontier.put((0, start))

   came_from = {}
   g_score = {start: 0}

   while not frontier.empty():
       _, current = frontier.get()

       if current == goal:
           path = []
           while current in came_from:
               path.append(current)
               current = came_from[current]
           path.append(start)
           path.reverse()
           return path

       r, c = current
       for dr, dc in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
           neighbor = (r + dr, c + dc)
           if (
               0 <= neighbor[0] < rows and
               0 <= neighbor[1] < cols and
               grid[neighbor[0]][neighbor[1]] == 0
           ):
               new_g = g_score[current] + 1

               if neighbor not in g_score or new_g <
g_score[neighbor]:
                   g_score[neighbor] = new_g
                   f_score = new_g + heuristic(neighbor, goal)
                   frontier.put((f_score, neighbor))
                   came_from[neighbor] = current

   return None
grid = [
   [0, 1, 0, 0],
   [0, 1, 0, 1],
   [0, 0, 0, 0],
   [1, 1, 0, 0]
]

start = (0, 2)
end = (3, 3)

path = a_star_search(grid, start, end)
print("A* Path:", path)
```

Output



```
● ~/Praneesh/Academics  main ? > python -u "/home/crimson/Praneesh/Academics/Sem5/AI&NN/A*algo.py"   09:26
  A* Path: [(0, 2), (1, 2), (2, 2), (2, 3), (3, 3)]
 ~/Praneesh/Academics  main ? >                                                                          09:26
```

Hill Climbing Algorithm

Aim

To implement the Hill Climbing algorithm in Python for locating a local optimum of a given mathematical function.

Description

Hill Climbing is a local search optimization algorithm widely used in the field of Artificial Intelligence. It iteratively evaluates neighboring solutions and moves toward the direction of improvement. The algorithm terminates when no better neighboring solution is available, thus reaching a local optimum.

This technique is especially suitable for problems where the search space is large, and an optimal or near-optimal solution is sufficient. It is easy to implement and computationally efficient for simple optimization tasks.

Algorithm

1. **Initialization**
   Select a random starting point within the domain of the function.

2. **Fitness Evaluation**
   Evaluate the fitness or quality of the current solution using the given function.

3. **Iterative Improvement**
   Repeat the following until convergence or maximum steps:

   - Generate neighboring solutions.

    - Select the best neighbor.
    - If the best neighbor has a higher fitness than the current solution, move to it.
    - If no improvement is found, terminate the process.

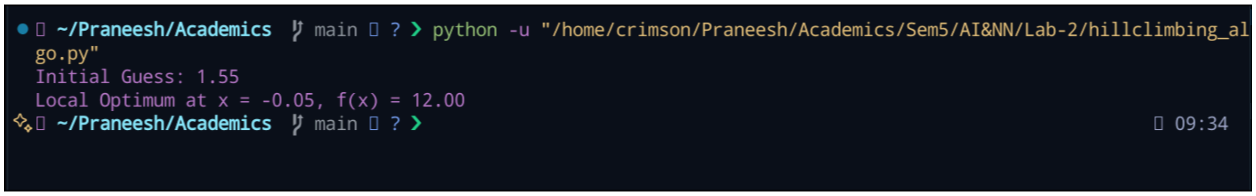
4. **Output**
   Return the best solution found along with its fitness value.

Python Code

```python
import random

def evaluate(x):
   return -1 * (x ** 2) + 12

def simple_hill_climb(initial_guess):
   current = initial_guess
   delta = 0.2
   max_steps = 100

   for _ in range(max_steps):
       neighbors = [current + delta, current - delta]

       best_neighbor = max(neighbors, key=evaluate)

       if evaluate(best_neighbor) > evaluate(current):
           current = best_neighbor
       else:
           break
   return current, evaluate(current)


initial_value = random.uniform(-8, 8)

optimum = simple_hill_climb(initial_value)
```

```python
print(f"Initial Guess: {initial_value:.2f}")
print(f"Local Optimum at x = {optimum[0]:.2f}, f(x) = {optimum[1]:.2f}")
```

Output



```
● ~/Praneesh/Academics  main ? > python -u "/home/crimson/Praneesh/Academics/Sem5/AI&NN/Lab-2/hillclimbing_al
  go.py"
  Initial Guess: 1.55
  Local Optimum at x = -0.05, f(x) = 12.00
  ~/Praneesh/Academics  main ? >                                                                      09:34
```