Artificial and Computational Intelligence - Assignment 1

List only the BITS (Name) of active contributors in this assignment:

Ritesh Ranjan

Shashank Singh

Divyanshu Singh

Adapa Hemachand

Problem Statement - 5

1. Explain the PEAS (Performance measure, Environment, Actuator, Sensor.) for your agent

PEAS stands for Performance measure, Environment, Actuator, and Sensor. It is a framework used to specify the characteristics of an intelligent agent in an AI system. Let's define the PEAS components for the rescue robot in the given scenario:

Performance measure:

The performance measure determines how well the rescue robot is performing in its task of helping the trapped rabbit find the optimal path to reach the finish position in the cave. In this scenario, the performance measure could be defined as follows:

The primary goal is to find the shortest and safest path from the rabbit's current position to the finish position while avoiding obstacles like walls, fire, and bushes.

Minimize the total path cost (including penalties for exposure to fire and bushes) to reach the finish position.

The time taken to reach the finish position could also be a factor in the performance measure, encouraging the robot to find the optimal path efficiently.

Environment:

The environment is the space in which the rescue robot operates and where the rabbit is trapped. In this scenario, the environment can be described as follows:

The cave is represented as a grid with cells. Each cell can be either empty, representing open space, or occupied by obstacles like walls, fire, or bushes.

The robot's position and the rabbit's position can be identified within the grid.

The robot can move to the north, south, west, and east directions in the grid, provided the target cell is not blocked by obstacles.

Actuator:

The actuator refers to the mechanisms or devices through which the rescue robot can interact with the environment. In this scenario, the actuator for the robot includes:

Movement control to navigate through the grid in the north, south, west, and east directions.

Path tracking and decision-making to select the optimal path based on the heuristic and path costs.

Sensor:

The sensor represents the perception capabilities of the rescue robot to gather information about the environment. In this scenario, the sensor for the robot includes:

Vision sensors to detect the presence of obstacles like walls, fire, and bushes in adjacent cells.

Distance measuring capabilities to calculate the Manhattan distance as a heuristic whenever necessary.

Sensitivity to fire cells to avoid moving towards them and incurring additional penalties.

Sensitivity to bush plant cells to consider the extra transition cost while planning the path.

<u>In summary, the PEAS components for the rescue robot in this scenario are as follows:</u>

Performance measure: Shortest and safest path to help the rabbit reach the finish position while minimizing the total path cost and considering time efficiency. The performance measure for the rescue robot is to find the optimal path from the start (S) to the goal (G) with the minimum cost. The cost includes the path cost of +3 for each transition, +5 for moving towards the fire cell, and +1 for moving towards the bush plant cell. The robot's objective is to minimize the total cost incurred during the pathfinding process.

Environment: A grid-based cave environment with walls, fire, bushes, and open spaces, where the robot and the trapped rabbit exist. The environment is represented as a 2D grid, where each cell can be either empty ('O'), a wall ('#'), fire ('F'), bush plant ('B'), the start ('S'), or the goal ('G'). The robot has to navigate through this grid while avoiding obstacles like walls, fire, and bushes.

Actuator: Movement control and path planning capabilities to navigate the robot through the cave grid. The actuator for the rescue robot allows it to move in four directions: north, south, west, and east. It can move to neighboring cells in the grid based on its current position.

Sensor: Vision sensors for obstacle detection, distance measurement for the heuristic, and sensitivity to fire and bush cells for penalty considerations during path planning. The robot's sensors provide information about the grid's current state, including the presence of walls, fire, bushes, and the goal. The sensors also detect the type of cells surrounding the robot, helping it make decisions based on the information it receives.

<u>2. Use both the above-mentioned algorithms and implement in PYTHON.</u>

Algorithms to find the optimal path.

a. Iterative Deepening A* Algorithm

b. Hill Climbing Algorithm

a. Iterative Deepening A* Algorithm

Iterative Deepening A* (IDA*) is a search algorithm that combines the concepts of Iterative Deepening Depth-First Search (IDDFS) and A* (A-star) search. It is used to find the optimal path from a start node to a goal node in a weighted graph or search space.

The basic idea behind IDA* is to perform a series of depth-limited searches, increasing the depth limit with each iteration until the goal node is reached. At each depth limit, the algorithm uses A* search to explore nodes within the given depth bound, effectively minimizing the total cost of the path.

Here's the general outline of the Iterative Deepening A* algorithm:

1. Initialize depth limit d to 0.
2. Perform A* search with a depth limit of d.
3. If a goal node is found, return the optimal path and terminate.
4. If no goal node is found within the depth limit d, increase d to the minimum f-cost (the sum of the path cost and the heuristic cost) among all nodes that exceeded the depth limit in the previous iteration.
5. Repeat steps 2-4 until a goal node is found.

The main advantage of IDA* over regular A* is that it doesn't require a lot of memory to store the search tree, as it only stores the path from the root node to the current node. However, IDA* may explore some nodes multiple times, which could make it less efficient compared to other informed search algorithms like A* with enough memory.

In the context of the rescue robot scenario, we have applied the IDA* algorithm to find the optimal path for the robot to reach the trapped rabbit while considering the path costs, penalties for fire and bush exposure, and using the Manhattan distance as the heuristic. The algorithm will iteratively increase the depth limit and perform A* search at each iteration until it finds the optimal path or determines that there is no valid path within the given depth bounds.

b. Hill Climbing Algorithm

The Hill Climbing algorithm is a local search algorithm used for optimization problems. It starts with an initial solution and iteratively makes small incremental changes to improve the solution until a local optimum is reached. It is called "hill climbing" because it metaphorically climbs a hill, trying to find the highest point (maximum) in the solution space.

Here's the general outline of the Hill Climbing algorithm:

1. Initialize the current solution with an initial state or a random solution.
2. Evaluate the current solution using an objective function, which measures the quality or fitness of the solution. The objective function is typically the one to be maximized or minimized.

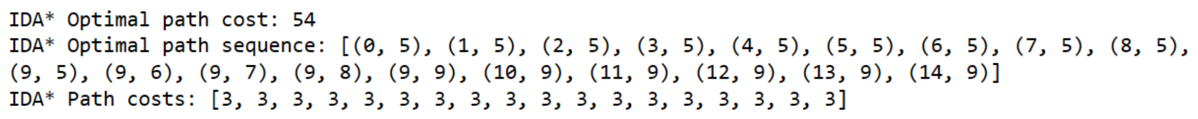
3. Generate neighboring solutions by making small modifications to the current solution. These modifications depend on the problem domain and can be random or systematic changes.
4. Evaluate the neighboring solutions using the objective function.
5. Select the best neighboring solution based on the evaluation and compare it with the current solution.
6. If the best neighboring solution is better (e.g., has a higher fitness value) than the current solution, move to the neighboring solution and repeat steps 3 to 6.
7. If no neighboring solution is better than the current solution, terminate the algorithm, and the current solution is considered a local optimum.

It is important to note that the Hill Climbing algorithm does not guarantee finding the global optimum. It might get stuck in a local optimum and not explore other parts of the solution space. To overcome this limitation, variants of Hill Climbing, such as Simulated Annealing and Genetic Algorithms, have been developed.

In the context of the rescue robot scenario, we have used the Hill Climbing algorithm to find a path from the robot's current position to the trapped rabbit. The objective function could be a combination of the path cost, penalties for fire and bush exposure, and the heuristic based on the Manhattan distance. The algorithm will try to make incremental moves towards a better path, but it may converge to a local optimal path, which may not be the globally optimal solution.

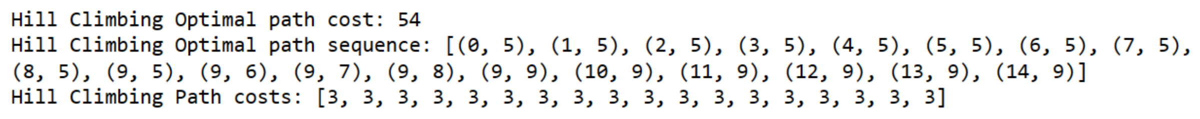
3. Print the optimal path sequence with costs.

Iterative Deepening A* Output:



```
IDA* Optimal path cost: 54
IDA* Optimal path sequence: [(0, 5), (1, 5), (2, 5), (3, 5), (4, 5), (5, 5), (6, 5), (7, 5), (8, 5),
(9, 5), (9, 6), (9, 7), (9, 8), (9, 9), (10, 9), (11, 9), (12, 9), (13, 9), (14, 9)]
IDA* Path costs: [3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3]
```

Hill Climbing Algorithm Output:



```
Hill Climbing Optimal path cost: 54
Hill Climbing Optimal path sequence: [(0, 5), (1, 5), (2, 5), (3, 5), (4, 5), (5, 5), (6, 5), (7, 5),
(8, 5), (9, 5), (9, 6), (9, 7), (9, 8), (9, 9), (10, 9), (11, 9), (12, 9), (13, 9), (14, 9)]
Hill Climbing Path costs: [3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3]
```

4. Include code in your implementation to calculate the space complexity and time complexity for the informed search and print the same. For local search interpret the significance of the hyperparameters if any applicable.

A* Algorithm (Informed Search):

Space Complexity for A* Algorithm:

The space complexity of the A* algorithm primarily depends on the size of the search space and the data structures used to store information during the search.

In the worst case, the algorithm may need to store information for all the nodes in the search space until the goal is found or until the entire search space is explored. This would result in a space complexity of O(V), where V is the number of nodes in the search space (grid cells in the cave).

Additionally, the algorithm uses data structures like the open set (priority queue) and the came_from dictionary to store the visited nodes and their associated costs. The size of these data structures can grow up to O(V).

Time Complexity for A* Algorithm:

The time complexity of the A* algorithm is influenced by the efficiency of the heuristic function and the structure of the search space.

In the worst case, where the heuristic function is not informative and the search space is large, the algorithm may explore a significant portion of the search space, leading to a time complexity of O(b^d), where b is the branching factor (average number of successors per node) and d is the depth of the goal node.

However, with an admissible and consistent heuristic (like the Manhattan distance), A* usually performs efficiently, and its time complexity is closer to the best-case time complexity of O(d).

Hill Climbing Algorithm (Local Search):

Space Complexity for Hill Climbing Algorithm:

The space complexity of the Hill Climbing algorithm mainly depends on the size of the neighborhood explored during the search.

In the worst case, if the search explores all possible neighboring states from the initial state, the space complexity would be O(V), where V is the number of states (grid cells in the cave).

Time Complexity for Hill Climbing Algorithm:

The time complexity of the Hill Climbing algorithm is influenced by the size of the search space, the number of iterations, and the quality of the chosen neighbors in each iteration.

In the worst case, where the search gets stuck in a local optimum and has to repeatedly explore the same neighborhood, the time complexity can be high.

The time complexity is not easily quantifiable and depends on the specifics of the problem and the chosen neighborhood search strategy.

Significance of Hyperparameters for Hill Climbing:

The Hill Climbing algorithm has some hyperparameters that can affect its performance, including the initial solution, neighborhood selection strategy, and termination condition.

The initial solution determines where the search starts, and it can significantly influence whether the algorithm finds a good solution or gets stuck in a local optimum.

The neighborhood selection strategy determines how the algorithm explores neighboring states. Choosing a good neighborhood strategy can help escape local optima and explore the search space more efficiently.

The termination condition determines when the algorithm stops. If the termination condition is too strict, the search may terminate prematurely without finding a good solution. If it is too lenient, the search may continue indefinitely without converging to a solution.

In summary, the A* algorithm is a more systematic and informed search approach that guarantees finding the optimal solution if an admissible and consistent heuristic is used.

However, its space and time complexity can grow significantly with the size of the search space.

On the other hand, the Hill Climbing algorithm is a local search approach that is simple and memory-efficient but may get stuck in local optima and is less predictable in terms of time complexity. Its performance heavily depends on the chosen hyperparameters and the specific problem's characteristics.