Birla Institute of Science and Technology

Artificial and Computational Intelligence

Problem solving by informed and uninformed search

Abstract:

In this particular case, we will be dealing with the design of a Problem Solving Agent in the given environment, to reach from office to home.

Problem Statement:

The city Nellore is facing heavy rains due to Cyclone Michaung. Most of the areas are inundated with water.

Design of a navigation agent which you can use to find the safe routes without water. The agent is fed with the map of the city marked with areas of flood. The agent must find the route that is the safest to take you home by choosing the next grid considering different factors. 5 points to be added each time the agent passes adjacent (Up, Down, Left, Right) safe places and 5 points to be deducted while the agent passes near water bodies and 3 points to be deducted if roads in the area are flooded with water.

Initial Board configuration

Task Environment	Grid Navigation (Static / Dynamic)
Algorithm	GBFS, Genetic
Fully vs Partially Observable	Fully
Single vs Multi-Agent	Single
Deterministic vs Stochastic	Deterministic
Episodic vs Sequential	Episodic
Static vs Dynamic	Static
Discrete vs Continuous	Discrete

G. Ankur Vatsa 2023aa05727
Nidasanametla Sree Sitamahalakshmi 2023aa05716
Prasenjit Samantha 2023aa05256
Randhawane Santosh 2023aa05828
Vedagiri Sai Krishna 2023aa05348

PEAS Environment

Performance	Find safest route to home
Environments	Navigation Agent's current position in grid
	City map
	Road
	Location of flood areas
	Goal/home location
	Neighbouring cells: Safe or Open cell and Unsafe or flooded water body cell
Sensors	Detect flood area and safe cell
Actuators	Agent allowed movements.
	Move Left, Move Right, Move Up, Move Down

Task Environment

Sensor based fully observable environment	Single agent navigation
Episodic Action	Static and
Discrete State	Deterministic environment

Task Environment Explained

With the confirmation of a fully observable environment, here's a breakdown of the task environment characteristics:

Deterministic:

- States: All grid positions, obstacles, start, and goal are known upfront, with no hidden elements or uncertainty impacting information accessibility.
- Actions: Movement actions (up, down, left, right) are assumed to be deterministic, always leading to the intended direction without unexpected events.
- Outcomes: Given the same state and action, the next state is always predictable (e.g., moving down from a specific position always leads to the position below).

Static:

- Grid: The 8x8 grid layout and obstacle positions remain constant throughout the task.
- Start and Goal: Both the start and goal nodes are fixed, and their locations never change.

Episodic:

- Independent trials: Each attempt to find the safe path from start to goal is considered an independent episode with a clear beginning (start node) and end (goal node).
- Reset between episodes: After reaching the goal or failing (e.g., hitting an obstacle), the agent is assumed to be reset to the starting position, starting a new episode.

Discrete:

- States: Individual grid positions represent distinct and countable states.
- Actions: The four movement options are distinct and finite.
- Transitions: Movement between states (e.g., from start to next position) happens in discrete steps, not continuously.

G. Ankur Vatsa 2023aa05727
Nidasanametla Sree Sitamahalakshmi 2023aa05716
Prasenjit Samantha 2023aa05256
Randhawane Santosh 2023aa05828
Vedagiri Sai Krishna 2023aa05348

Designing a PSA to reach goal safely using informed search
Method: Greedy best first search

GBFS is an informed search algorithm that always expands the node that is estimated to be closest to the goal based on the heuristic function h(n). It is not optimal because it may get stuck in local optima or loops, but it is often used when optimality is not a strict requirement and computational efficiency is important.

Greedy Best First Algorithm:

1. Initialize a tree with the root node being the start node in the open list.

2. If the open list is empty, return a failure, otherwise, add the current node to the closed list.

3. Remove the node with the lowest h(n) value from the open list for exploration.

4. If a child node is the target, return success. Otherwise, if the node has not been in either the open or closed list, add it to the open list for exploration.

Heuristic function h(n):

A heuristic function – h(n), assesses the proximity of a successive node to the target node, favouring the immediate low-cost option. Consequently, while this approach may prioritize nodes that appear to be closer to the goal, it does not guarantee the discovery of the shortest path to reach the goal node. Given, 5 points to be added each time the agent passes adjacent (Up, Down, Left, Right) safe places and 5 points to be deducted while the agent passes near water bodies and 3 points to be deducted if roads in the area are flooded with water.

Calculating Heuristic – h(n):

h(n) =

if (row, col) == flooded_cell; score -= 3

else if (row, col) == blocked_cell; score -= 5

else if (row, col) == safe_cell; score += 5

Grid Representation

```
Enter the start position (row,col): 0,0
Enter the goal position (row,col): 7,7
Final grid with start and goal positions:
S . . . # # . .
. F F . . . . F
. . . . . . . .
# # . F . . . .
. F . . . . # #
. . . F . . . .
. # # . . F . .
. . . . . F . G
```

G. Ankur Vatsa 2023aa05727
Nidasanametla Sree Sitamahalakshmi 2023aa05716
Prasenjit Samantha 2023aa05256
Randhawane Santosh 2023aa05828
Vedagiri Sai Krishna 2023aa05348

GBFS Pseudo Code

```
-----GBFS pseudo-code----

function greedy_best_first_search(problem):
    Initialize an empty priority queue frontier
    Add the initial state of the problem to the frontier with priority
    f(initial_state) = h(initial_state)
    Initialize an empty set explored to keep track of explored states

    while frontier is not empty:
        current_node = pop the node with the highest priority from the
                                                                frontier
        Add current_node.state to explored
        if current_node.state is the goal state:
            return current_node.path  # Return the path to the goal
        for each neighbor of current_node.state:
            if neighbor is not in explored and neighbor is not in frontier:
                Add neighbor to frontier with priority f(neighbor) =
                                                              h(neighbor)
            else if neighbor is in frontier with a lower priority:
                Update neighbor's priority to f(neighbor) = h(neighbor)
                Update neighbor's parent to current_node

    return failure  # No solution found
```

G. Ankur Vatsa 2023aa05727
Nidasanametla Sree Sitamahalakshmi 2023aa05716
Prasenjit Samantha 2023aa05256
Randhawane Santosh 2023aa05828
Vedagiri Sai Krishna 2023aa05348

Designing a PSA to reach goal safely using uninformed search
Method: Genetic Algorithm

Uninformed search algorithms explore the search space without any additional information about the state space or the problem's constraints. They make decisions based solely on the available state transitions and do not use problem-specific knowledge beyond the definition of the problem itself. Examples of some other uninformed search algorithms include depth-first search, breadth-first search, and uniform-cost search.

Genetic Algorithm:

A genetic algorithm (GA) is a type of metaheuristic optimization algorithm inspired by the process of natural selection and evolution. It is used to find approximate solutions to optimization and search problems by mimicking the process of natural selection in biological organisms. A simplified explanation of how a genetic algorithm works is given below:

1. **Initialization**: A population of potential solutions (individuals) to the optimization problem is randomly generated.
2. **Evaluation**: Every individual in the population is evaluated and assigned a fitness score, which indicates how well it solves the optimization problem.
3. **Selection**: Individuals are selected from the population to become parents for the next generation based on their fitness scores. Individuals with higher fitness scores are more likely to be selected.
4. **Crossover**: Pairs of selected parents are combined to produce offspring (new individuals) through crossover or recombination. This process involves exchanging genetic information (e.g., parts of a solution representation) between parents to create new candidate solutions.
5. **Mutation**: Occasionally, random changes (mutations) are introduced to the offspring's genetic information to add diversity to the population.
6. **Replacement**: The offspring replaces some individuals in the current population, creating a new generation of candidate solutions.
7. **Termination**: The process iterates through the selection, crossover, mutation, and replacement steps for a predetermined number of generations or until a satisfactory solution is found.

By repeating these steps over multiple generations, the population evolves towards better solutions to the optimization problem.

Pseudo code for Genetic Algorithm

```
initialize_population()
evaluate_fitness(population)
best_solution = find_best_solution(population)

while termination_criteria_not_met():
    next_generation = []

    while len(next_generation) < population_size:
        parent1 = select_parent(population)
        parent2 = select_parent(population)
        child = crossover(parent1, parent2)
        mutate(child)
        next_generation.append(child)

    population = next_generation
    evaluate_fitness(population)
    best_solution = find_best_solution(population)

return best_solution
```

G. Ankur Vatsa 2023aa05727
Nidasanametla Sree Sitamahalakshmi 2023aa05716
Prasenjit Samantha 2023aa05256
Randhawane Santosh 2023aa05828
Vedagiri Sai Krishna 2023aa05348

GBFS Algorithm Execution

```
Start (0, 0), Goal (7, 7):
Open List (Priority Queue): [(10, (0, 0))]
Closed List (Visited Set): set()
Priority of next cell 3
Priority of next cell 3
Open List (Priority Queue): [(3, (0, 1)), (3, (1, 0))]
Closed List (Visited Set): {(0, 0)}
Priority of next cell 9
Open List (Priority Queue): [(3, (1, 0)), (9, (0, 2))]
Closed List (Visited Set): {(0, 1), (0, 0)}
Priority of next cell 7
Open List (Priority Queue): [(7, (2, 0)), (9, (0, 2))]
Closed List (Visited Set): {(0, 1), (1, 0), (0, 0)}
Priority of next cell 5
Open List (Priority Queue): [(5, (2, 1)), (9, (0, 2))]
Closed List (Visited Set): {(0, 1), (1, 0), (2, 0), (0, 0)}
Priority of next cell 16
Open List (Priority Queue): [(9, (0, 2)), (16, (2, 2))]
Closed List (Visited Set): {(0, 1), (2, 1), (0, 0), (2, 0), (1, 0)}
Priority of next cell 8
Open List (Priority Queue): [(8, (0, 3)), (16, (2, 2))]
Closed List (Visited Set): {(0, 1), (2, 1), (0, 0), (2, 0), (0, 2), (1, 0)}
Priority of next cell 16
Open List (Priority Queue): [(16, (1, 3)), (16, (2, 2))]
Closed List (Visited Set): {(0, 1), (2, 1), (0, 0), (0, 3), (2, 0), (0, 2), (1, 0)}
Priority of next cell 17
Priority of next cell 15
Open List (Priority Queue): [(15, (1, 4)), (17, (2, 3)), (16, (2, 2))]
Closed List (Visited Set): {(0, 1), (2, 1), (0, 0), (0, 3), (2, 0), (0, 2), (1, 0), (1, 3)}
Priority of next cell 26
Priority of next cell 16
Open List (Priority Queue): [(16, (1, 5)), (16, (2, 2)), (26, (2, 4)), (17, (2, 3))]
Closed List (Visited Set): {(0, 1), (2, 1), (0, 0), (0, 3), (2, 0), (1, 4), (0, 2), (1, 0),
(1, 3)}
Priority of next cell 27
Priority of next cell 19
Open List (Priority Queue): [(16, (2, 2)), (17, (2, 3)), (26, (2, 4)), (27, (2, 5)), (19, (1,
6))]
Closed List (Visited Set): {(0, 1), (2, 1), (0, 0), (1, 5), (0, 3), (2, 0), (1, 4), (0, 2),
(1, 0), (1, 3)}
Priority of next cell 7
Open List (Priority Queue): [(7, (3, 2)), (17, (2, 3)), (26, (2, 4)), (27, (2, 5)), (19, (1,
6))]
Closed List (Visited Set): {(0, 1), (2, 1), (0, 0), (1, 5), (0, 3), (2, 0), (1, 4), (0, 2),
(2, 2), (1, 0), (1, 3)}
Priority of next cell 18
Open List (Priority Queue): [(17, (2, 3)), (18, (4, 2)), (26, (2, 4)), (27, (2, 5)), (19, (1,
6))]
Closed List (Visited Set): {(0, 1), (2, 1), (0, 0), (1, 5), (0, 3), (2, 0), (1, 4), (0, 2),
(2, 2), (1, 0), (3, 2), (1, 3)}
Open List (Priority Queue): [(18, (4, 2)), (19, (1, 6)), (26, (2, 4)), (27, (2, 5))]
Closed List (Visited Set): {(0, 1), (2, 1), (0, 0), (1, 5), (0, 3), (2, 0), (1, 4), (2, 3),
(0, 2), (2, 2), (1, 0), (3, 2), (1, 3)}
Priority of next cell 9
Priority of next cell 11
Open List (Priority Queue): [(9, (5, 2)), (11, (4, 3)), (26, (2, 4)), (27, (2, 5)), (19, (1,
6))]
Closed List (Visited Set): {(0, 1), (2, 1), (0, 0), (1, 5), (0, 3), (2, 0), (1, 4), (4, 2),
(2, 3), (0, 2), (2, 2), (1, 0), (3, 2), (1, 3)}
Priority of next cell 10
Open List (Priority Queue): [(10, (5, 1)), (11, (4, 3)), (26, (2, 4)), (27, (2, 5)), (19, (1,
6))]
Closed List (Visited Set): {(0, 1), (2, 1), (0, 0), (1, 5), (0, 3), (2, 0), (1, 4), (4, 2),
(2, 3), (0, 2), (2, 2), (1, 0), (3, 2), (1, 3), (5, 2)}
Priority of next cell 24
Open List (Priority Queue): [(11, (4, 3)), (19, (1, 6)), (26, (2, 4)), (27, (2, 5)), (24, (5,
0))]
```

G. Ankur Vatsa 2023aa05727
Nidasanametla Sree Sitamahalakshmi 2023aa05716
Prasenjit Samantha 2023aa05256
Randhawane Santosh 2023aa05828
Vedagiri Sai Krishna 2023aa05348

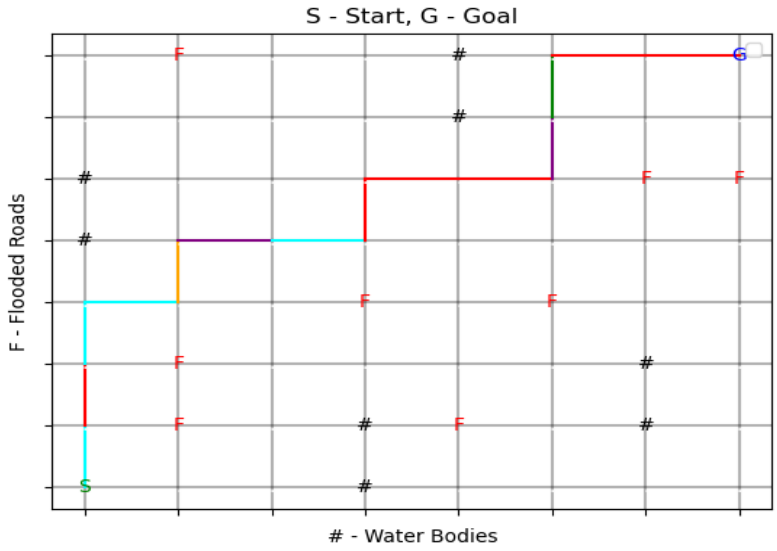
```
Closed List (Visited Set): {(0, 1), (2, 1), (0, 0), (1, 5), (0, 3), (2, 0), (5, 1), (1, 4),
(4, 2), (2, 3), (0, 2), (2, 2), (1, 0), (3, 2), (1, 3), (5, 2)}
Priority of next cell 28
Open List (Priority Queue): [(19, (1, 6)), (24, (5, 0)), (26, (2, 4)), (27, (2, 5)), (28, (4,
4))]
Closed List (Visited Set): {(0, 1), (2, 1), (0, 0), (1, 5), (4, 3), (0, 3), (2, 0), (5, 1),
(1, 4), (4, 2), (2, 3), (0, 2), (2, 2), (1, 0), (3, 2), (1, 3), (5, 2)}
Priority of next cell 28
Priority of next cell 13
Open List (Priority Queue): [(13, (0, 6)), (27, (2, 5)), (24, (5, 0)), (28, (4, 4)), (28, (2,
6)), (26, (2, 4))]
Closed List (Visited Set): {(0, 1), (2, 1), (0, 0), (1, 5), (4, 3), (0, 3), (2, 0), (5, 1),
(1, 4), (4, 2), (2, 3), (0, 2), (2, 2), (1, 0), (1, 6), (3, 2), (1, 3), (5, 2)}
Priority of next cell 11
Open List (Priority Queue): [(11, (0, 7)), (27, (2, 5)), (24, (5, 0)), (28, (4, 4)), (28, (2,
6)), (26, (2, 4))]
Closed List (Visited Set): {(4, 3), (5, 1), (0, 2), (2, 2), (1, 0), (1, 6), (1, 3), (4, 2),
(0, 1), (2, 1), (1, 5), (3, 2), (5, 2), (0, 0), (0, 3), (2, 0), (1, 4), (0, 6), (2, 3)}
Open List (Priority Queue): [(24, (5, 0)), (27, (2, 5)), (26, (2, 4)), (28, (4, 4)), (28, (2,
6))]
Closed List (Visited Set): {(4, 3), (5, 1), (0, 2), (2, 2), (1, 0), (1, 6), (1, 3), (4, 2),
(0, 1), (0, 7), (2, 1), (1, 5), (3, 2), (5, 2), (0, 0), (0, 3), (2, 0), (1, 4), (0, 6), (2,
3)}
Priority of next cell 15
Priority of next cell 7
Open List (Priority Queue): [(7, (4, 0)), (26, (2, 4)), (15, (6, 0)), (28, (4, 4)), (27, (2,
5)), (28, (2, 6))]
Closed List (Visited Set): {(4, 3), (5, 1), (0, 2), (2, 2), (1, 0), (1, 6), (1, 3), (4, 2),
(5, 0), (0, 1), (0, 7), (2, 1), (1, 5), (3, 2), (5, 2), (0, 0), (0, 3), (2, 0), (1, 4), (0,
6), (2, 3)}
Open List (Priority Queue): [(15, (6, 0)), (26, (2, 4)), (28, (2, 6)), (28, (4, 4)), (27, (2,
5))]
Closed List (Visited Set): {(4, 0), (4, 3), (5, 1), (0, 2), (2, 2), (1, 0), (1, 6), (1, 3),
(4, 2), (5, 0), (0, 1), (0, 7), (2, 1), (1, 5), (3, 2), (5, 2), (0, 0), (0, 3), (2, 0), (1,
4), (0, 6), (2, 3)}
Priority of next cell 21
Open List (Priority Queue): [(21, (7, 0)), (26, (2, 4)), (28, (2, 6)), (28, (4, 4)), (27, (2,
5))]
Closed List (Visited Set): {(4, 0), (4, 3), (5, 1), (0, 2), (2, 2), (1, 0), (1, 6), (1, 3),
(4, 2), (5, 0), (0, 1), (0, 7), (2, 1), (1, 5), (3, 2), (5, 2), (0, 0), (0, 3), (2, 0), (1,
4), (0, 6), (2, 3), (6, 0)}
Priority of next cell 17
Open List (Priority Queue): [(17, (7, 1)), (26, (2, 4)), (28, (2, 6)), (28, (4, 4)), (27, (2,
5))]
Closed List (Visited Set): {(4, 0), (4, 3), (5, 1), (0, 2), (2, 2), (1, 0), (1, 6), (1, 3),
(4, 2), (5, 0), (0, 1), (0, 7), (2, 1), (1, 5), (7, 0), (3, 2), (5, 2), (0, 0), (0, 3), (2,
0), (1, 4), (0, 6), (2, 3), (6, 0)}
Priority of next cell 18
Open List (Priority Queue): [(18, (7, 2)), (26, (2, 4)), (28, (2, 6)), (28, (4, 4)), (27, (2,
5))]
Closed List (Visited Set): {(4, 0), (4, 3), (5, 1), (0, 2), (2, 2), (1, 0), (1, 6), (1, 3),
(7, 1), (4, 2), (5, 0), (0, 1), (0, 7), (2, 1), (1, 5), (7, 0), (3, 2), (5, 2), (0, 0), (0,
3), (2, 0), (1, 4), (0, 6), (2, 3), (6, 0)}
Priority of next cell 29
Open List (Priority Queue): [(26, (2, 4)), (27, (2, 5)), (28, (2, 6)), (28, (4, 4)), (29, (7,
3))]
Closed List (Visited Set): {(4, 0), (4, 3), (5, 1), (0, 2), (2, 2), (1, 0), (1, 6), (1, 3),
(7, 1), (4, 2), (5, 0), (0, 1), (0, 7), (2, 1), (1, 5), (7, 0), (3, 2), (5, 2), (0, 0), (0,
3), (2, 0), (1, 4), (0, 6), (2, 3), (7, 2), (6, 0)}
Priority of next cell 19
Open List (Priority Queue): [(19, (3, 4)), (27, (2, 5)), (28, (2, 6)), (29, (7, 3)), (28, (4,
4))]
Closed List (Visited Set): {(4, 0), (4, 3), (5, 1), (0, 2), (2, 2), (1, 0), (1, 6), (1, 3),
(7, 1), (4, 2), (5, 0), (0, 1), (0, 7), (2, 4), (2, 1), (1, 5), (7, 0), (3, 2), (5, 2), (0,
0), (0, 3), (2, 0), (1, 4), (0, 6), (2, 3), (7, 2), (6, 0)}
Priority of next cell 28
Open List (Priority Queue): [(27, (2, 5)), (28, (3, 5)), (28, (2, 6)), (29, (7, 3)), (28, (4,
4))]
Closed List (Visited Set): {(4, 0), (3, 4), (4, 3), (5, 1), (0, 2), (2, 2), (1, 0), (1, 6),
(1, 3), (7, 1), (4, 2), (5, 0), (0, 1), (0, 7), (2, 4), (2, 1), (1, 5), (7, 0), (3, 2), (5,
2), (0, 0), (0, 3), (2, 0), (1, 4), (0, 6), (2, 3), (7, 2), (6, 0)}
Open List (Priority Queue): [(28, (2, 6)), (28, (3, 5)), (28, (4, 4)), (29, (7, 3))]
```

```
Closed List (Visited Set): {(4, 0), (3, 4), (4, 3), (5, 1), (0, 2), (2, 2), (1, 0), (1, 6),
(2, 5), (1, 3), (7, 1), (4, 2), (5, 0), (0, 1), (0, 7), (2, 4), (2, 1), (1, 5), (7, 0), (3,
2), (5, 2), (0, 0), (0, 3), (2, 0), (1, 4), (0, 6), (2, 3), (7, 2), (6, 0)}
Priority of next cell 19
Priority of next cell 16
Open List (Priority Queue): [(16, (2, 7)), (19, (3, 6)), (28, (4, 4)), (29, (7, 3)), (28, (3,
5))]
Closed List (Visited Set): {(4, 0), (3, 4), (4, 3), (5, 1), (0, 2), (2, 2), (1, 0), (1, 6),
(2, 5), (1, 3), (7, 1), (4, 2), (5, 0), (0, 1), (0, 7), (2, 4), (2, 1), (1, 5), (7, 0), (3,
2), (5, 2), (0, 0), (0, 3), (2, 0), (1, 4), (0, 6), (2, 3), (2, 6), (7, 2), (6, 0)}
Priority of next cell 15
Open List (Priority Queue): [(15, (3, 7)), (19, (3, 6)), (28, (4, 4)), (29, (7, 3)), (28, (3,
5))]
Closed List (Visited Set): {(4, 0), (3, 4), (4, 3), (5, 1), (0, 2), (2, 2), (1, 0), (1, 6),
(2, 5), (1, 3), (7, 1), (4, 2), (5, 0), (0, 1), (0, 7), (2, 4), (2, 1), (2, 7), (1, 5), (7,
0), (3, 2), (5, 2), (0, 0), (0, 3), (2, 0), (1, 4), (0, 6), (2, 3), (2, 6), (7, 2), (6, 0)}
Open List (Priority Queue): [(19, (3, 6)), (28, (3, 5)), (28, (4, 4)), (29, (7, 3))]
Closed List (Visited Set): {(4, 0), (3, 4), (4, 3), (3, 7), (5, 1), (0, 2), (2, 2), (1, 0),
(1, 6), (2, 5), (1, 3), (7, 1), (4, 2), (5, 0), (0, 1), (0, 7), (2, 4), (2, 1), (2, 7), (1,
5), (7, 0), (3, 2), (5, 2), (0, 0), (0, 3), (2, 0), (1, 4), (0, 6), (2, 3), (2, 6), (7, 2),
(6, 0)}
Open List (Priority Queue): [(28, (3, 5)), (29, (7, 3)), (28, (4, 4))]
Closed List (Visited Set): {(4, 0), (3, 4), (4, 3), (3, 7), (5, 1), (0, 2), (2, 2), (1, 0),
(1, 6), (2, 5), (1, 3), (7, 1), (4, 2), (5, 0), (3, 6), (0, 1), (0, 7), (2, 4), (2, 1), (2,
7), (1, 5), (7, 0), (3, 2), (5, 2), (0, 0), (0, 3), (2, 0), (1, 4), (0, 6), (2, 3), (2, 6),
(7, 2), (6, 0)}
Priority of next cell 19
Open List (Priority Queue): [(19, (4, 5)), (29, (7, 3)), (28, (4, 4))]
Closed List (Visited Set): {(4, 0), (3, 4), (4, 3), (3, 7), (5, 1), (0, 2), (2, 2), (1, 0),
(1, 6), (2, 5), (1, 3), (7, 1), (4, 2), (5, 0), (3, 6), (0, 1), (0, 7), (2, 4), (2, 1), (2,
7), (1, 5), (7, 0), (3, 2), (3, 5), (5, 2), (0, 0), (0, 3), (2, 0), (1, 4), (0, 6), (2, 3),
(2, 6), (7, 2), (6, 0)}
Priority of next cell 22
Open List (Priority Queue): [(22, (5, 5)), (29, (7, 3)), (28, (4, 4))]
Closed List (Visited Set): {(4, 0), (3, 4), (4, 3), (3, 7), (5, 1), (0, 2), (2, 2), (1, 0),
(1, 6), (2, 5), (1, 3), (7, 1), (4, 2), (4, 5), (5, 0), (3, 6), (0, 1), (0, 7), (2, 4), (2,
1), (2, 7), (1, 5), (7, 0), (3, 2), (3, 5), (5, 2), (0, 0), (0, 3), (2, 0), (1, 4), (0, 6),
(2, 3), (2, 6), (7, 2), (6, 0)}
Priority of next cell 21
Priority of next cell 23
Open List (Priority Queue): [(21, (5, 6)), (23, (5, 4)), (28, (4, 4)), (29, (7, 3))]
Closed List (Visited Set): {(4, 0), (3, 4), (4, 3), (3, 7), (5, 1), (0, 2), (2, 2), (1, 0),
(1, 6), (2, 5), (1, 3), (7, 1), (4, 2), (4, 5), (5, 0), (3, 6), (0, 1), (0, 7), (2, 4), (2,
1), (2, 7), (1, 5), (7, 0), (3, 2), (3, 5), (5, 2), (5, 5), (0, 0), (0, 3), (2, 0), (1, 4),
(0, 6), (2, 3), (2, 6), (7, 2), (6, 0)}
Priority of next cell 24
Priority of next cell 17
Open List (Priority Queue): [(17, (5, 7)), (23, (5, 4)), (28, (4, 4)), (29, (7, 3)), (24, (6,
6))]
Closed List (Visited Set): {(4, 0), (3, 4), (4, 3), (3, 7), (5, 1), (0, 2), (2, 2), (1, 0),
(1, 6), (2, 5), (1, 3), (7, 1), (4, 2), (4, 5), (5, 0), (5, 6), (3, 6), (0, 1), (0, 7), (2,
4), (2, 1), (2, 7), (1, 5), (7, 0), (3, 2), (3, 5), (5, 2), (5, 5), (0, 0), (0, 3), (2, 0),
(1, 4), (0, 6), (2, 3), (2, 6), (7, 2), (6, 0)}
Priority of next cell 23
Open List (Priority Queue): [(23, (5, 4)), (23, (6, 7)), (28, (4, 4)), (29, (7, 3)), (24, (6,
6))]
Closed List (Visited Set): {(4, 0), (3, 4), (4, 3), (3, 7), (5, 1), (5, 7), (0, 2), (2, 2),
(1, 0), (1, 6), (2, 5), (1, 3), (7, 1), (4, 2), (4, 5), (5, 0), (5, 6), (3, 6), (0, 1), (0,
7), (2, 4), (2, 1), (2, 7), (1, 5), (7, 0), (3, 2), (3, 5), (5, 2), (5, 5), (0, 0), (0, 3),
(2, 0), (1, 4), (0, 6), (2, 3), (2, 6), (7, 2), (6, 0)}
Priority of next cell 24
Open List (Priority Queue): [(23, (6, 7)), (24, (6, 4)), (28, (4, 4)), (29, (7, 3)), (24, (6,
6))]
Closed List (Visited Set): {(4, 0), (3, 4), (4, 3), (3, 7), (5, 4), (5, 1), (5, 7), (0, 2),
(2, 2), (1, 0), (1, 6), (2, 5), (1, 3), (7, 1), (4, 2), (4, 5), (5, 0), (5, 6), (3, 6), (0,
1), (0, 7), (2, 4), (2, 1), (2, 7), (1, 5), (7, 0), (3, 2), (3, 5), (5, 2), (5, 5), (0, 0),
(0, 3), (2, 0), (1, 4), (0, 6), (2, 3), (2, 6), (7, 2), (6, 0)}
Priority of next cell 24
Open List (Priority Queue): [(24, (6, 4)), (24, (6, 6)), (28, (4, 4)), (29, (7, 3)), (24, (7,
7))]
Closed List (Visited Set): {(4, 0), (3, 4), (4, 3), (3, 7), (5, 4), (5, 1), (5, 7), (0, 2),
(2, 2), (1, 0), (1, 6), (2, 5), (1, 3), (7, 1), (4, 2), (4, 5), (5, 0), (5, 6), (3, 6), (0,
```

```
1), (0, 7), (2, 4), (2, 1), (2, 7), (1, 5), (7, 0), (6, 7), (3, 2), (3, 5), (5, 2), (5, 5),
(0, 0), (0, 3), (2, 0), (1, 4), (0, 6), (2, 3), (2, 6), (7, 2), (6, 0)}
Priority of next cell 20
Priority of next cell 15
Open List (Priority Queue): [(15, (6, 3)), (24, (6, 6)), (20, (7, 4)), (29, (7, 3)), (24, (7,
7)), (28, (4, 4))]
Closed List (Visited Set): {(4, 0), (3, 4), (4, 3), (3, 7), (5, 4), (5, 1), (5, 7), (0, 2),
(2, 2), (1, 0), (1, 6), (2, 5), (1, 3), (7, 1), (4, 2), (4, 5), (5, 0), (5, 6), (3, 6), (0,
1), (0, 7), (2, 4), (2, 1), (2, 7), (1, 5), (7, 0), (6, 4), (6, 7), (3, 2), (3, 5), (5, 2),
(5, 5), (0, 0), (0, 3), (2, 0), (1, 4), (0, 6), (2, 3), (2, 6), (7, 2), (6, 0)}
Open List (Priority Queue): [(20, (7, 4)), (24, (6, 6)), (28, (4, 4)), (29, (7, 3)), (24, (7,
7))]
Closed List (Visited Set): {(4, 0), (3, 4), (4, 3), (3, 7), (5, 4), (5, 1), (5, 7), (0, 2),
(2, 2), (1, 0), (1, 6), (2, 5), (1, 3), (7, 1), (4, 2), (4, 5), (5, 0), (5, 6), (3, 6), (0,
1), (0, 7), (2, 4), (2, 1), (2, 7), (1, 5), (7, 0), (6, 4), (6, 7), (3, 2), (3, 5), (5, 2),
(5, 5), (0, 0), (0, 3), (2, 0), (1, 4), (0, 6), (2, 3), (2, 6), (7, 2), (6, 0), (6, 3)}
Open List (Priority Queue): [(24, (6, 6)), (24, (7, 7)), (28, (4, 4)), (29, (7, 3))]
Closed List (Visited Set): {(4, 0), (3, 4), (4, 3), (3, 7), (5, 4), (5, 1), (5, 7), (0, 2),
(2, 2), (1, 0), (1, 6), (2, 5), (1, 3), (7, 4), (7, 1), (4, 2), (4, 5), (5, 0), (5, 6), (3,
6), (0, 1), (0, 7), (2, 4), (2, 1), (2, 7), (1, 5), (7, 0), (6, 4), (6, 7), (3, 2), (3, 5),
(5, 2), (5, 5), (0, 0), (0, 3), (2, 0), (1, 4), (0, 6), (2, 3), (2, 6), (7, 2), (6, 0), (6,
3)}
Priority of next cell 15
Open List (Priority Queue): [(15, (7, 6)), (24, (7, 7)), (28, (4, 4)), (29, (7, 3))]
Closed List (Visited Set): {(4, 0), (3, 4), (4, 3), (3, 7), (5, 4), (5, 1), (5, 7), (0, 2),
(2, 2), (1, 0), (1, 6), (2, 5), (1, 3), (7, 4), (7, 1), (4, 2), (4, 5), (5, 0), (5, 6), (3,
6), (0, 1), (0, 7), (2, 4), (2, 1), (2, 7), (1, 5), (7, 0), (6, 4), (6, 7), (3, 2), (3, 5),
(5, 2), (5, 5), (0, 0), (0, 3), (2, 0), (1, 4), (0, 6), (2, 3), (2, 6), (7, 2), (6, 0), (6,
6), (6, 3)}
Open List (Priority Queue): [(24, (7, 7)), (29, (7, 3)), (28, (4, 4))]
Closed List (Visited Set): {(4, 0), (3, 4), (4, 3), (3, 7), (5, 4), (5, 1), (5, 7), (0, 2),
(2, 2), (1, 0), (1, 6), (2, 5), (1, 3), (7, 4), (7, 1), (4, 2), (4, 5), (5, 0), (5, 6), (3,
6), (0, 1), (0, 7), (2, 4), (2, 1), (2, 7), (1, 5), (7, 0), (6, 4), (6, 7), (7, 6), (3, 2),
(3, 5), (5, 2), (5, 5), (0, 0), (0, 3), (2, 0), (1, 4), (0, 6), (2, 3), (2, 6), (7, 2), (6,
0), (6, 6), (6, 3)}
Average Branching Factor: 3
Depth of the graph search tree is: 14
Total nodes expanded: 45
Time Complexity of GBFS: 4782969
Worst case Space Complexity of GBFS: 6
Windows 105705472 105705472
Execution time: 0.005991458892822266 seconds
Memory usage: 0.0 KB
Path taken by the agent using Greedy Best First Search: [(0, 0), (0, 1), (0, 2), (0, 3), (1,
3), (1, 4), (2, 4), (3, 4), (3, 5), (4, 5), (5, 5), (5, 6), (5, 7), (6, 7), (7, 7)]
```

Total path cost using Greedy Best First Search: 14

Total Memory Usage Greedy Best First Search: 45

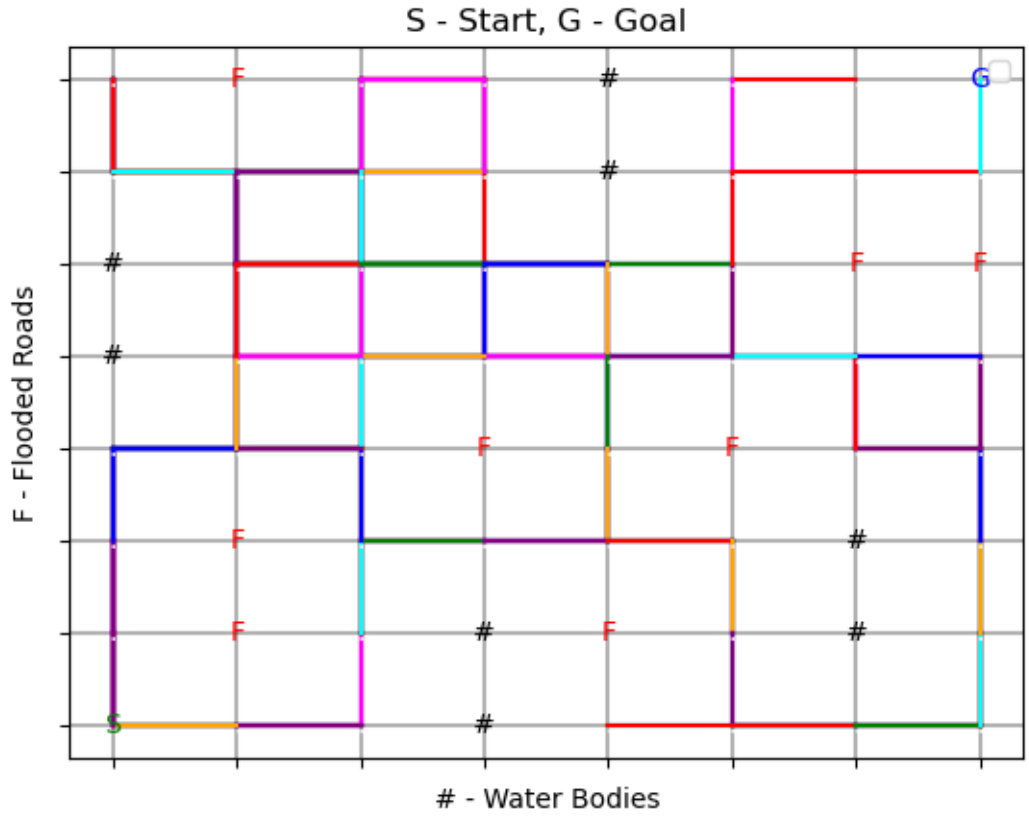


Genetic Algorithm Execution

```
Start (0, 0), Goal (7, 7):
Memory Computation for Genetic Search is : 1000
Time Complexity of Genetic Search Algorithm: 626200.0
Space Complexity of Genetic Search Algorithm: 7210
Windows 114753536 115077120
Execution time: 3.9377124309539795 seconds
Memory usage: 316.0 KB
Path taken by the agent using Genetic Search Algorithm: [(0, 0), (0, 1), (0, 0), (1, 0), (2,
0), (2, 1), (2, 0), (2, 1), (2, 0), (1, 0), (2, 0), (1, 0), (0, 0), (0, 1), (0, 2), (0, 1),
(0, 2), (0, 1), (0, 2), (0, 3), (1, 3), (0, 3), (1, 3), (1, 4), (1, 3), (2, 3), (1, 3), (0,
3), (0, 2), (0, 1), (0, 2), (0, 3), (1, 3), (0, 3), (1, 3), (2, 3), (2, 4), (3, 4), (2, 4),
(2, 5), (2, 4), (3, 4), (2, 4), (3, 4), (3, 5), (2, 5), (1, 5), (1, 6), (2, 6), (1, 6), (0,
6), (0, 7), (0, 6), (1, 6), (2, 6), (1, 6), (1, 5), (2, 5), (1, 5), (1, 6), (0, 6), (0, 7),
(0, 6), (0, 7), (0, 6), (1, 6), (0, 6), (0, 7), (0, 6), (1, 6), (0, 6), (1, 6), (2, 6), (2,
5), (1, 5), (1, 4), (1, 5), (1, 4), (2, 4), (2, 3), (1, 3), (1, 4), (1, 3), (2, 3), (1, 3),
(0, 3), (0, 2), (0, 3), (1, 3), (0, 3), (0, 2), (0, 1), (0, 2), (0, 1), (0, 2), (0, 3), (0,
2), (0, 3), (1, 3), (1, 4), (1, 5), (1, 6), (2, 6), (2, 7), (2, 6), (1, 6), (2, 6), (2, 7),
(2, 6), (2, 5), (2, 6), (2, 7), (3, 7), (2, 7), (3, 7), (3, 6), (3, 7), (3, 6), (3, 5), (3,
4), (2, 4), (2, 5), (3, 5), (2, 5), (1, 5), (1, 6), (2, 6), (3, 6), (3, 7), (3, 6), (3, 5),
(3, 6), (2, 6), (1, 6), (1, 5), (1, 6), (0, 6), (1, 6), (1, 5), (1, 4), (1, 5), (2, 5), (2,
4), (2, 5), (2, 4), (2, 3), (1, 3), (1, 4), (1, 3), (2, 3), (2, 4), (3, 4), (3, 5), (2, 5),
(1, 5), (1, 4), (2, 4), (2, 3), (1, 3), (2, 3), (1, 3), (1, 4), (2, 4), (2, 3), (2, 2), (2,
1), (2, 2), (2, 3), (2, 4), (3, 4), (4, 4), (5, 4), (4, 4), (4, 5), (5, 5), (4, 5), (3, 5),
(3, 6), (2, 6), (3, 6), (3, 5), (2, 5), (2, 6), (2, 7), (3, 7), (3, 6), (3, 7), (3, 6), (2,
6), (2, 5), (2, 4), (3, 4), (4, 4), (4, 5), (5, 5), (4, 5), (4, 4), (4, 5), (3, 5), (4, 5),
(3, 5), (2, 5), (1, 5), (1, 4), (1, 5), (1, 6), (2, 6), (2, 7), (3, 7), (3, 6), (2, 6), (2,
7), (2, 6), (1, 6), (0, 6), (0, 7), (0, 6), (0, 7), (0, 6), (0, 7), (0, 6), (0, 7), (0, 6),
(1, 6), (0, 6), (1, 6), (1, 5), (1, 6), (1, 5), (1, 4), (2, 4), (2, 5), (2, 4), (3, 4), (3,
5), (4, 5), (4, 4), (4, 3), (4, 4), (3, 4), (3, 5), (2, 5), (2, 6), (2, 5), (1, 5), (2, 5),
(1, 5), (1, 6), (2, 6), (2, 5), (2, 6), (2, 5), (2, 6), (3, 6), (2, 6), (2, 7), (3, 7), (2,
7), (3, 7), (3, 6), (3, 5), (2, 5), (1, 5), (1, 6), (1, 5), (1, 4), (1, 3), (0, 3), (0, 2),
(0, 3), (1, 3), (2, 3), (1, 3), (1, 4), (1, 5), (1, 4), (2, 4), (2, 5), (2, 6), (2, 5), (1,
5), (2, 5), (2, 6), (1, 6), (0, 6), (1, 6), (1, 5), (1, 4), (1, 3), (2, 3), (1, 3), (1, 4),
(1, 3), (1, 4), (1, 5), (1, 6), (2, 6), (2, 5), (2, 4), (1, 4), (2, 4), (1, 4), (2, 4), (2,
3), (2, 2), (2, 3), (2, 4), (2, 5), (3, 5), (3, 4), (4, 4), (4, 3), (4, 4), (4, 3), (4, 4),
(4, 3), (4, 2), (3, 2), (4, 2), (3, 2), (2, 2), (3, 2), (2, 2), (2, 3), (2, 2), (2, 3), (2,
2), (2, 3), (2, 4), (3, 4), (3, 5), (2, 5), (2, 6), (2, 7), (2, 6), (3, 6), (2, 6), (2, 5),
(1, 5), (1, 4), (2, 4), (2, 5), (2, 4), (2, 5), (3, 5), (2, 5), (2, 6), (2, 5), (1, 5), (2,
5), (3, 5), (3, 4), (3, 5), (4, 5), (4, 4), (3, 4), (4, 4), (5, 4), (4, 4), (4, 5), (5, 5),
(5, 4), (6, 4), (5, 4), (5, 5), (4, 5), (3, 5), (2, 5), (3, 5), (2, 5), (2, 6), (2, 5), (3,
5), (4, 5), (3, 5), (4, 5), (3, 5), (3, 4), (4, 4), (5, 4), (6, 4), (5, 4), (6, 4), (6, 3),
(6, 4), (7, 4), (7, 3), (7, 2), (7, 1), (7, 0), (6, 0), (5, 0), (6, 0), (7, 0), (7, 1), (7,
2), (7, 3), (7, 2), (7, 1), (7, 2), (7, 1), (7, 0), (7, 1), (7, 2), (7, 3), (7, 4), (6, 4),
(7, 4), (7, 3), (6, 3), (7, 3), (7, 2), (7, 3), (6, 3), (6, 4), (6, 3), (6, 4), (5, 4), (5,
5), (4, 5), (5, 5), (5, 4), (6, 4), (6, 3), (7, 3), (7, 2), (7, 3), (7, 2), (7, 3), (6, 3),
(7, 3), (7, 4), (7, 3), (7, 4), (7, 3), (6, 3), (6, 4), (6, 3), (7, 3), (7, 2), (7, 3), (6,
3), (6, 4), (7, 4), (6, 4), (5, 4), (5, 5), (5, 6), (5, 7), (6, 7), (5, 7), (5, 6), (5, 5),
(5, 4), (4, 4), (5, 4), (6, 4), (5, 4), (4, 4), (3, 4), (4, 4), (4, 5), (3, 5), (3, 4), (3,
5), (4, 5), (3, 5), (2, 5), (1, 5), (2, 5), (1, 5), (1, 6), (0, 6), (0, 7), (0, 6), (0, 7),
(0, 6), (0, 7), (0, 6), (1, 6), (0, 6), (1, 6), (2, 6), (2, 7), (2, 6), (2, 7), (3, 7), (3,
6), (3, 7), (2, 7), (2, 6), (2, 7), (2, 6), (2, 5), (3, 5), (4, 5), (3, 5), (2, 5), (1, 5),
(1, 6), (1, 5), (1, 6), (0, 6), (0, 7), (0, 6), (1, 6), (1, 5), (1, 4), (1, 3), (0, 3), (1,
3), (0, 3), (0, 2), (0, 3), (0, 2), (0, 1), (0, 0), (0, 1), (0, 2), (0, 1), (0, 0), (0, 1),
(0, 2), (0, 3), (0, 2), (0, 3), (0, 2), (0, 1), (0, 2), (0, 3), (1, 3), (2, 3), (2, 2), (3,
2), (4, 2), (5, 2), (4, 2), (4, 3), (4, 2), (4, 3), (4, 4), (4, 3), (4, 2), (4, 3), (4, 2),
(3, 2), (2, 2), (2, 3), (2, 2), (2, 1), (2, 2), (2, 1), (2, 2), (2, 1), (2, 2), (2, 1), (2,
2), (3, 2), (4, 2), (4, 3), (4, 2), (5, 2), (5, 1), (5, 2), (5, 1), (5, 2), (4, 2), (4, 3),
(4, 2), (4, 3), (4, 4), (5, 4), (4, 4), (3, 4), (4, 4), (3, 4), (4, 4), (4, 3), (4, 4), (3,
4), (2, 4), (3, 4), (2, 4), (2, 3), (2, 2), (2, 3), (2, 2), (3, 2), (2, 2), (3, 2), (2, 2),
(2, 1), (2, 2), (2, 1), (2, 2), (2, 3), (1, 3), (0, 3), (0, 2), (0, 1), (0, 2), (0, 1), (0,
0), (0, 1), (0, 0), (1, 0), (2, 0), (1, 0), (0, 0), (0, 1), (0, 2), (0, 1), (0, 0), (1, 0),
(0, 0), (0, 1), (0, 0), (1, 0), (2, 0), (1, 0), (0, 0), (0, 1), (0, 0), (1, 0), (0, 0), (1,
0), (0, 0), (0, 1), (0, 2), (0, 1), (0, 0), (0, 1), (0, 0), (0, 1), (0, 0), (0, 1), (0, 0),
(0, 1), (0, 2), (0, 3), (1, 3), (1, 4), (2, 4), (1, 4), (1, 5), (2, 5), (3, 5), (4, 5), (3,
5), (2, 5), (3, 5), (4, 5), (4, 4), (4, 3), (4, 2), (5, 2), (4, 2), (5, 2), (5, 1), (5, 2),
(4, 2), (3, 2), (4, 2), (3, 2), (4, 2), (5, 2), (5, 1), (5, 0), (4, 0), (5, 0), (6, 0), (5,
0), (5, 1), (5, 0), (6, 0), (7, 0), (6, 0), (7, 0), (6, 0), (7, 0), (7, 1), (7, 0), (7, 1),
(7, 0), (7, 1), (7, 0), (6, 0), (5, 0), (6, 0), (5, 0), (6, 0), (5, 0), (4, 0), (5, 0), (5,
1), (5, 0), (6, 0), (7, 0), (7, 1), (7, 2), (7, 3), (7, 2), (7, 1), (7, 0), (6, 0), (7, 0),
(7, 1), (7, 0), (7, 1), (7, 2), (7, 1), (7, 2), (7, 3), (7, 4), (6, 4), (5, 4), (5, 5), (5,
6), (6, 6), (7, 6), (7, 7)]
```

Total path cost using Genetic Search Algorithm: 3505
Total Memory Usage Genetic Search Algorithm: 1000

G. Ankur Vatsa 2023aa05727
Nidasanametla Sree Sitamahalakshmi 2023aa05716
Prasenjit Samantha 2023aa05256
Randhawane Santosh 2023aa05828
Vedagiri Sai Krishna 2023aa05348



Time and Space complexity

Greedy Best First Search:
time complexity: 4782969
and
space complexity: 6

Genetic Search:
time complexity: 626200.0
and
space complexity: 7210

Comparative Analysis: Greedy Best First Search and Genetic Algorithms

This study delves into the comparative performance of Greedy Best First Search (GBFS) and Genetic Algorithms (GAs) in the context of pathfinding within an 8x8 grid environment containing fixed obstacles. The goal was to locate a target position at (7, 7) while starting from (0, 0). Key findings and implications are highlighted below:

Efficiency vs. Resilience:

- GBFS emerges as a memory-efficient and time-optimal algorithm for unobstructed scenarios. However, its vulnerability to dead ends necessitates restarts, potentially hindering performance in complex environments with numerous obstacles.
- GAs exhibit remarkable robustness in navigating blockages, drawing upon their ability to explore diverse pathways simultaneously. This versatility comes at a cost of

increased time and memory complexity, underscoring the importance of parameter tuning for optimal performance.

Recommendations and Future Directions:

- Algorithm Selection:
    - Prioritize GBFS when memory constraints are stringent, and the path is expected to be relatively clear.
    - Opt for GAs when dealing with intricate environments densely populated with obstacles, especially when time efficiency is less critical.
- Hybrid Approaches: Explore the potential of combining GBFS's speed with GAs' flexibility to create adaptive algorithms capable of handling diverse pathfinding challenges.
- Parameter Tuning: Devote efforts to fine-tuning GA parameters to strike a balance between exploration and exploitation, aiming to achieve optimal performance within acceptable time and memory constraints.
- Further Investigation: Conduct comprehensive research on the impact of grid size, obstacle density, and goal position on the algorithms' relative performance to uncover valuable insights for a broader range of pathfinding scenarios.

Conclusion:

The choice between GBFS and GAs hinges upon striking a delicate balance between efficiency, robustness, and resource constraints. Understanding their distinct strengths and limitations empowers informed decision-making in pathfinding applications across diverse domains, ranging from robotics and logistics to video game development and network optimization.

GIT Reference:

https://github.com/vatsaaa/mtech/blob/main/semester_1/03_assignments/aci/ACI_ASSIGNMENT_1/ipynb/ACI_Assignment_1_SolutionTemplate_PS_1.ipynb