YILDIZ TECHNICAL UNIVERSITY

FACULTY OF CHEMICAL AND METALLURGICAL ENGINEERING

DEPARTMENT OF MATHEMATICAL ENGINEERING

INTRODUCTION TO ARTIFICIAL INTELLIGENCE

PROJECT REPORT

Berkay AHİ, 19058042
Emirhan ZENGİN, 17058034
Zeynep DEMİR, 20058042
Helin BARDAK, 21058038
Suzal DEMİRCİLER, 22058609

Istanbul, 2024

CONTENT Page

1. INTRODUCTION

This report outlines strategies for solving pathfinding challenges on designated terrain maps. The main objective is to find the most cost-effective route from a randomly assigned starting point to a specific target location. This involves navigating through different types of terrain, such as plains, seas, hills, and barriers, each presenting unique costs. Two terrain sizes are analyzed: a medium-sized 5x5 map and a larger 10x10 map. A key part of this approach is developing a path cost matrix that can be used for any terrain size, which allows for testing various pathfinding algorithms. This method ensures that the solution can be applied to a broad range of situations. The project utilizes several algorithms to find the shortest path from randomly chosen state to another state in the terrain. These include Breadth-First Search, Uniform-Cost Search, Depth-First Search, Depth-Limited Search, Iterative Deepening Search, Greedy Search, A* Search, and Generalized A* Search. Each algorithm is assessed based on how effectively it finds the shortest path, taking into account the different costs and challenges of the terrain. The report details how each algorithm is implemented, their performance, and the results, providing a comprehensive analysis of their practical effectiveness.

2. DEFINITION OF THE PROBLEM

In the pathfinding challenge presented in this report, the terrain map is divided into different types of regions, each with a specific migration cost associated with moving through it. The regions are classified as plain, sea, hill, and barrier. The movement cost varies by region type: plains have a cost of 1, seas have a cost of 3, hills carry a cost of 5, and barriers are impassable, represented by an infinite cost. Movement on the map is constrained to four directions: up, down, left, and right.

```
enum ACTIONS {
    Go_Up, Go_Down, Go_Left, Go_Right
};
```

Figure 2.1 Definition of actions in the code.

```
if (parent_state->cell != new_cell && PATH_COST[parent_state->cell][new_cell] > 0) {
    // Check if there's a valid path cost and it's not the same cell
    State new_state = *parent_state;
    new_state.cell = new_cell;
    trans_model->new_state = new_state;
    trans_model->step_cost = PATH_COST[parent_state->cell][new_cell];
    return TRUE;
```

Figure 2.2 Transition model used in the project.

The cost incurred for a move is dictated by the region type of the current location. The initial and goal states in the scenarios covered by this report are not fixed; they are programmatically determined, allowing for a wide range of possible routes to be explored and analyzed. This variability introduces a dynamic aspect to our problem, ensuring that the algorithms need to be adaptable to any given start and end points on the map. For heuristic purposes, Euclidean distance is employed as the metric to estimate the cost from any point on the map to the goal. This choice aids in efficiently guiding search-based algorithms towards the goal state by providing a direct-line distance estimate, which is particularly useful for algorithms like A* and Greedy Search that rely on heuristic functions to prioritize their search paths. In order to make applications related to the problem, two terrains with different characteristics, medium and large size, were created. Images of the two different terrains can be seen below.

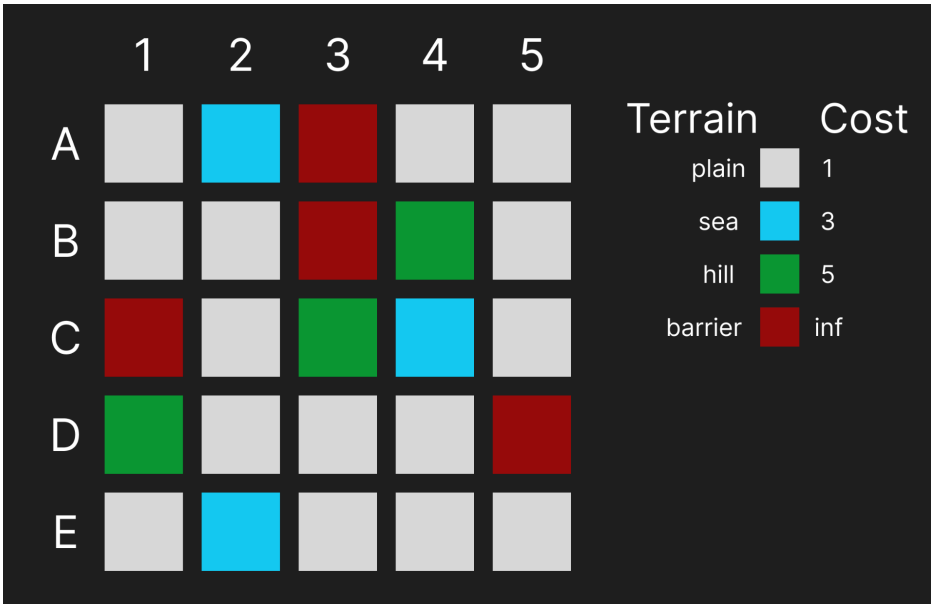

Figure 2.3 5x5 medium sized terrain.

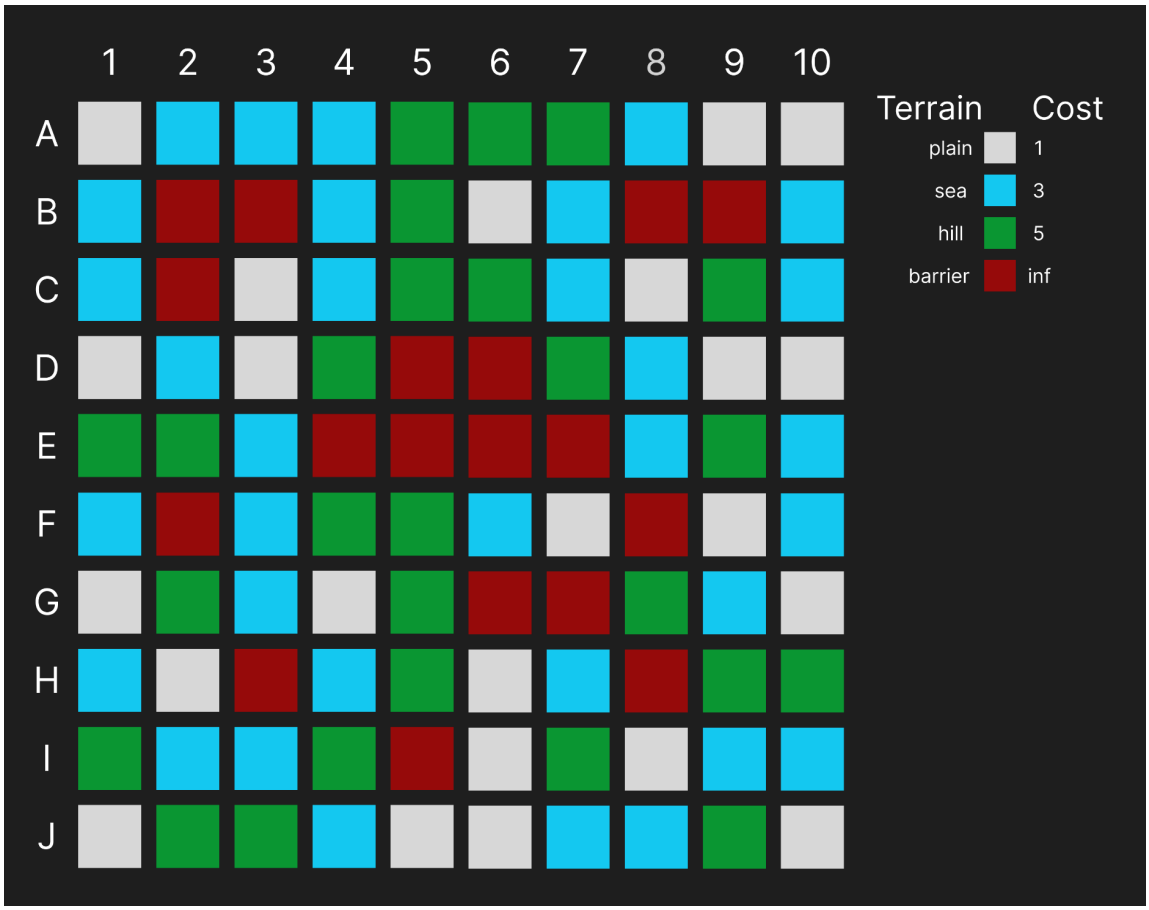

Figure 2.4 10x10 large sized terrain.

3. HEURISTIC FUNCTION

The Euclidean distance is employed as the heuristic function in finding the shortest path in terrain problems. This metric calculates the straight-line distance between two points on a plane, making it an ideal heuristic for algorithms like A*. The formula for Euclidean distance is the square root of the sum of the squares of the differences in the x and y coordinates of the two points. In mathematical terms, it is expressed as:

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

where x_1,y_1 and x_2, y_2 are the coordinates of the starting point and the goal, respectively. In the context of pathfinding, this heuristic is particularly valuable as it provides a direct and intuitive estimate of the distance to the goal. It helps algorithms like A* search to prioritize paths that appear to be closer to the target, thereby potentially reducing the total number of paths explored and increasing the efficiency of the search process. The implementation of the Euclidean distance function and its integration into the pathfinding algorithms is central to our approach, ensuring that the movements across the terrain are both optimal and efficient in reaching the destination.

```
// ============================== SLD HEURISTIC MATRIX GENERATOR ==================================
float calculateDistance(int start, int end, int size) {
    int startRow = start / size;
    int startCol = start % size;
    int endRow = end / size;
    int endCol = end % size;

    int dx = abs(startRow - endRow);
    int dy = abs(startCol - endCol);

    return sqrt(dx * dx + dy * dy);
}

// Initialize the heuristic matrix (SLD Matrix)
void initializeSLDMatrix(int size) {
    float** heuristicMatrix = (float**)malloc(size * size * sizeof(float*));
    for (int i = 0; i < size * size; i++) {
        heuristicMatrix[i] = (float*)malloc(size * size * sizeof(float));
        for (int j = 0; j < size * size; j++) {
            heuristicMatrix[i][j] = calculateDistance(i, j, size);
        }
    }
}
```

Figure 3.1 Heuristic functions' SLD matrix generator code in our problem.

4. APPLICATION AND RESULTS

The application phase involved executing the developed program across the two different terrains defined in the project—medium-sized 5x5 and larger 10x10 maps. For each terrain, the program was run using each of the eight pathfinding algorithms detailed earlier. This extensive testing allowed us to observe and record the performance and efficacy of each algorithm under varying conditions. The results from these simulations are systematically organized into two comprehensive tables, one for each terrain size. These tables encapsulate key data from the simulations, including the initial and goal states, the solution path taken, the cost associated with the path, and the computational performance indicators such as the number of nodes searched, generated, and held in memory, along with the computation time for each run. This data provides a detailed comparative analysis of each algorithm's performance, highlighting their strengths and limitations in different scenarios.

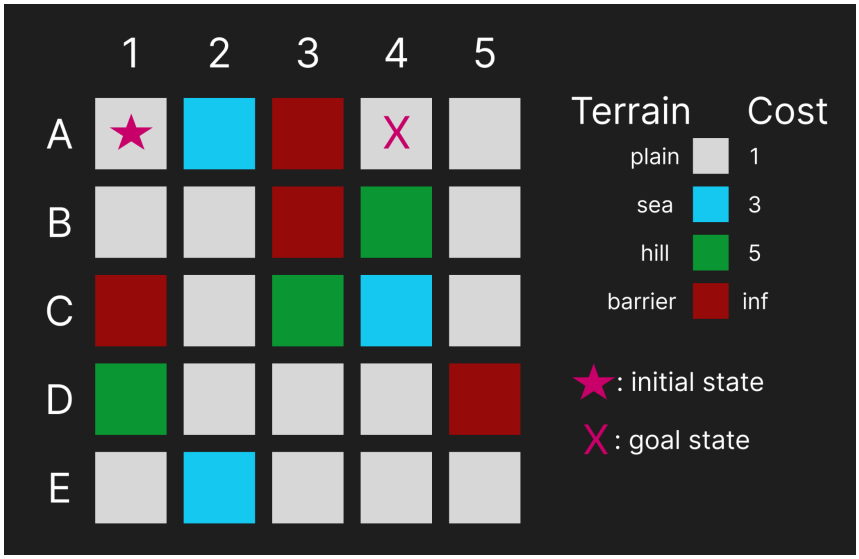

Figure 4.1 Different algorithms are performed for initial state A1 to goal state A4 in 5x5 Terrain.

Search Algorithm	Init_State	Goal_State	Solution_Path	Path_Cost	Nodes_Searched	Nodes_Generated	Nodes_In_Memory	Run_Time (secs)	Is_Optimal
Breadth-First	A1	A4	A1->B1->B2->C2->C3->C4->B4->A4	17	18	42	42	0.000956	No
Uniform-Cost	A1	A4	A1->B1->B2->C2->D2->D3->D4->C4->C5->B5->A5->A4	13	20	52	52	0.001297	Yes
Depth-First	A1	A4	A1->A2->B2->C2->C3->C4->C5->B5->A5->A4	17	15	25	15	0.000949	No
Depth-Limited (l=7)	A1	A4	A1->A2->B2->C2->C3->C4->B4->A4	19	15	24	11	0.000623	No
Iterative Deepening	A1	A4	A1->A2->B2->C2->C3->C4->B4->A4	19	62	104	11	0.006589	No
Greedy	A1	A4	A1->B1->B2->C2->C3->C4->B4->A4	17	14	23	23	0.000627	No
A*	A1	A4	A1->B1->B2->C2->D2->D3->D4->C4->C5->B5->A5->A4	13	19	50	50	0.000825	Yes
Generalized A* (a=0.25)	A1	A4	A1->B1->B2->C2->D2->D3->D4->C4->C5->B5->A5->A4	13	19	50	50	0.000657	Yes
Generalized A* (a=0.75)	A1	A4	A1->B1->B2->C2->C3->C4->C5->B5->A5->A4	15	20	52	52	0.000993	No

Figure 4.2 Result table for the 5x5 Terrain.

When the table is analyzed it can be seen that The Uniform-Cost, A*, and Generalized A* (a=0.25) algorithms found the optimal path, with the lowest path cost of 13. This suggests these algorithms are effective in finding the most cost-efficient path in a controlled setting. Also, Breadth-First and Iterative Deepening algorithms are not optimal because the step costs are not all identical in this problem.

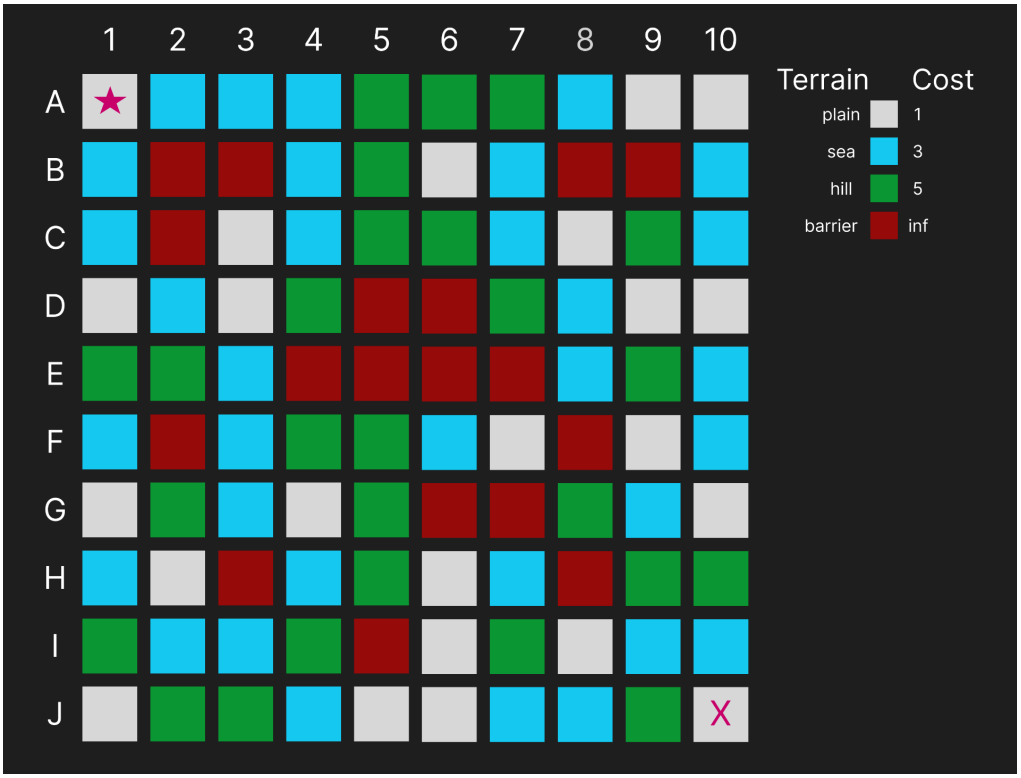

Figure 4.3 Different algorithms are performed for initial state A1 to goal state J10 in 10x10.

Search Algorithm	Init_State	Goal_State	Path_Cost	Nodes_Searched	Nodes_Generated	Nodes_In_Memory	Run_Time (secs)	Is_Optimal
Breadth-First	A1	J10	52	82	232	232	0.009294	No
Uniform-Cost	A1	J10	44	82	235	235	0.007789	Yes
Depth-First	A1	J10	108	62	104	62	0.00775	No
Depth-Limited (l=18)	A1	J10	54	65	142	30	0.009067	No
Iterative Deepening	A1	J10	54	680	1525	30	0.055667	No
Greedy	A1	J10	50	81	215	215	0.006822	No
A*	A1	J10	44	82	235	235	0.013704	Yes
Generalized A* (a=0.25)	A1	J10	44	82	235	235	0.016615	Yes

Figure 4.4 Result table for the 10x10 Terrain.

The comparative analysis of search algorithms across 5x5 and 10x10 terrains reveals significant differences in performance metrics, including nodes searched, nodes generated, memory usage, and computation times. In smaller terrains, algorithms generally execute faster and with fewer resources, as demonstrated by Breadth-First and Depth-First searches. However, as the terrain size increases to 10x10, there is a substantial rise in computational demands, illustrated by higher nodes generated and longer run times, particularly noticeable in Iterative Deepening Search which exhibited a marked increase in memory usage. Despite the increased complexity, algorithms such as Uniform-Cost and both versions of A* maintained efficiency, consistently finding optimal paths while controlling resource utilization effectively across both terrain sizes. These results underscore the scalability and efficiency of these algorithms, suggesting their applicability for complex pathfinding tasks in varying terrain sizes.