Course: Artificial Intelligence

Project 1: Labyrinth Search

Name: Konstantinos Vardakas

Student ID: 522

Email: pcs0522@uoi.gr

Introduction

The first project implements two search algorithms in the C programming language, Uniform Cost Search (UCS) and A*, to solve a robot navigation problem in an NxN maze. The maze is generated randomly with a probability p for free cells, while cells A (bottom left, coordinates (N-1, 0)) and B (top right, coordinates (0, N-1)) are always free, as are the start (S) and goal (G) cells. The robot can move horizontally, vertically, or diagonally at a cost of 1, as well as move directly between cells A and B at a cost of 2.

Implementation

Initially, the maze is created as an NxN table, where each cell is free with probability p. Cells S, G, A, and B are explicitly defined as free. The coordinates of cells S and G are entered by the user.

Implementation of UCS

The Uniform Cost Search (UCS) algorithm is implemented using a priority queue, which manages nodes based on the total cost g(n) from the initial cell S. Each node includes the coordinates (x, y), the cost g(n), and the coordinates of the parent for path reconstruction.

The process begins with the initialization of the priority queue, into which the initial node S is entered with a cost of g(n)=0. At each step, the node with the smallest g(n) is selected from the queue. If this node corresponds to the final cell G, then the path is reconstructed using the information about the parents of the nodes and the total cost is returned. Otherwise, all possible transitions to the eight neighboring free cells (horizontal, vertical, and diagonal) are examined, each with a cost equal to 1. If the new transition cost to a neighboring cell is less than what has already been recorded for that cell, then the node information is updated and re-entered into the queue. In addition, if the current node is the special cell A, the possible special transition to cell B with a cost equal to 2 is examined, and vice versa. The process continues until we reach G or until the queue is empty, at which point it is declared that there is no available path. Throughout the search, the number of nodes that were expanded is recorded.

The path is printed by reconstructing it from G to S, using a table to mark the cells of the path, while the maze is displayed with symbols for S, G, the path (*), free cells (blank), and obstacles (X).

Implementation of A*

The A* algorithm is implemented in a similar way to UCS, with the main difference being that it uses the function f(n)=g(n)+h(n), where h(n) is the heuristic estimate of the cost from node n to the goal, in order to guide the search more effectively. The structure of the nodes remains the same as that of UCS, with the addition of the value f(n), which is used for sorting in the priority queue.

Similar to the UCS algorithm, the process begins by inserting the initial cell S into the queue, with a value of g(n)=0, and thus the initial total cost is determined by the value of the heuristic function of A*, f(n)=h(S). At each iteration, the node with the smallest f(n) is selected. If this node corresponds to cell G, then the path is reconstructed through the parents and the total cost is returned. Otherwise, all possible moves to the eight neighboring free cells are examined, with a cost of 1 for each move. If the new value g(n) for a cell is less than the one already recorded, then h(n) is recalculated, f(n) is updated, and the node is added back to the queue. As in UCS, if the node corresponds to the special cell A or B, the special move between them is examined at a cost of 2. The search continues until it reaches the target or the nodes are exhausted, at which point the failure to find a path is declared. Throughout the process, the number of nodes that were expanded is recorded.

A* Heuristic Function

The heuristic function h(n) calculates the minimum estimated cost from a cell (x, y) to the final cell (gx, gy). It is calculated as the minimum of three options:

1. Direct distance: Uses the Chebyshev distance, defined as the maximum absolute difference between the x-coordinates and y-coordinates between (x,y) and (gx,gy).
2. Distance via cell A: Calculated as the Chebyshev distance from (x,y) to A (N-1,0), plus the transport cost (2), plus the Chebyshev distance from B (0,N-1) to (gx,gy).
3. Distance via cell B: Calculated as the Chebyshev distance from (x,y) to B (0,N-1), plus the transport cost (2), plus the Chebyshev distance from A (N-1,0) to (gx,gy).

The final heuristic is the minimum of these three values.

Admissibility of the Heuristic Function

The heuristic function used is admissible, as it never overestimates the actual cost of the optimal path. Specifically, the Chebyshev distance assumes that the robot can move diagonally and that there are no obstacles. Since each movement (horizontal, vertical, or diagonal) has a cost of 1, the Chebyshev distance provides the minimum possible cost in a maze without obstacles and is therefore always less than or equal to the actual cost of the route. Furthermore, when the heuristic includes the special transport through cell A, the Chebyshev distance from the current cell to A is calculated, the fixed cost of transport (2) is added, and then the distance from B to the goal G is added. The same applies to the reverse route via B.

Since the minimum of these three estimates is ultimately selected, it is impossible to overestimate the total cost. Furthermore, the heuristic function is consistent (or monotonic), as it satisfies the property that for every node n and every successor n′, the value of the heuristic at n, i.e. h(n), does not exceed the sum of the transition cost from n to n′, i.e. c(n,n′), and the heuristic at n′, i.e. h(n′). Mathematically, this is expressed as:

$$h(n) \leq c(n, n') + h(n')$$

Thus, this heuristic function allows the A* algorithm to find an optimal solution as it is acceptable. Also, each node is expanded at most once, and thus the algorithm becomes efficient due to the consistency of the function.

Results and Conclusions

To evaluate the algorithms, tests were performed with various mazes. Indicative results for 5 mazes with N=10, p=0.7 (the differences between the 5 mazes result from the randomness of the cells being free or blocked), S=(9.3), and G=(2.8) (with access to points A and B):

path cost					
experiment number	1	2	3	4	5
UCS	7	7	8	8	7
A*	7	7	8	8	7

Both algorithms find the optimal path through each maze. In these particular mazes, it required the use of cells A and B.

number of expansions						
experiment number	1	2	3	4	5	mean
UCS	55	52	63	59	61	58.0
A*	14	14	22	15	14	15.8
Difference	41	38	41	44	47	42.2

The A* algorithm is significantly more efficient than UCS, as it requires far fewer expansions to find the optimal solution. This behavior is expected, since A* also utilizes the heuristic function, which provides additional information about the estimated cost from each cell to the goal. Based on this approach, the priority of the cells in the queue changes, giving priority to those that are estimated to lead to the goal faster and more economically. Similar results were obtained for experiments with other values of the hyperparameters (N, p, S, G).