Searching Report

Introduction

The problem we will be working with is **an agent finding a path from one position to a different position in a grid**. Each position in the grid is represented by a numeric value indicating the cost to move to that position from any neighbor. For example,

1 5 2 4 1 5 0 0 4 0

1 5 0 2 4 1 0 2 5 0

1 0 5 5 1 2 1 4 4 3

2 1 2 2 0 0 2 4 4 0

2 0 4 2 0 0 1 5 1 2

0 3 4 1 3 0 4 2 1 3

1 0 3 5 2 5 1 0 4 3

2 4 0 1 0 2 4 2 4 0

0 4 5 5 2 5 3 1 4 3

3 0 3 3 5 5 3 4 1 1

**A value of 0 indicates that the location cannot be traversed**. The agent can move in the four cardinal directions one position at a time. The agent **cannot** move diagonally.

Problem Formulation

The problem is formulated using a 2D array state representation and actions to move the agent in any of the four directions.

**State representation**: 2D array of integers. The 0 values correspond to out of bounds areas, while the other numbers correspond to the cost it takes to move to that tile. The arrangement of numbers in the 2D array correspond to the grid that is traversed upon by the agent.

**Actions:** The list of actions moves the agent into the space on the board in that direction.

* Up: Moves the agent up one tile
* Down: Moves the agent down one tile
* Left: Moves the agent left one tile
* Right: Moves the agent right one tile

All actions have a step cost that equals the number the agent is moving to.

**Transition Model**: The transition model accepts a state, get that states neighbors, and returns a new state based on those neighbors. The transition model checks that the agent doesn’t move to any 0s on the grid.

**Path**: The path cost will be the sum of the step costs on the path.

Uninformed Search

Uninformed search randomly expands nodes until it reaches the goal state. This randomness makes it the simplest way of searching through a grid.

Breadth-First Search (BFS) expands nodes from an open list at the front. This makes it so the algorithm starts at the root, gets the nodes that are neighbors, then gets those neighbors, and so on.

Depth-First Search (DFS) also expands nodes from an open list, but this time from the back. This means that the algorithms iterate down a path until it hits a dead end and then backtracks.

The only difference between DFS and BFS are the FIFO and LIFO structure of the nodes.

Both algorithms use a for loop, meaning they have a linear time and space complexity.

The results from these algorithms were mostly random based on where the 0s were in the grid.

Informed Search

Informed search differs from uninformed search in the way of a heuristic function. In this case, Euclidean distance was used to determine how far we were from our goal state.

The final if statement was changed when iterating through the children to include this new information. The data structure is also a priority queue this time around to sort by path cost and heuristic.

These changes resulted in less nodes explored as well as less path cost overall.

Local Search

We used local search to figure out an attacking queens problem. We were given a grid with a certain number of chess queens, and we had to move the queens around to create a board where the least amount was attacking each other.

Local search algorithms are only obsessed with a current state. The goal is to find a local extremum, but they can get trapped in the wrong places.

The methods we were already given combined with the code from the slides allowed us to solve the problem with simulated annealing.

We tested multiple board sizes, and we found that the larger the board size, the more attacking queens there were after the algorithm was completed. This has to be because the larger the space, the more likely it is to find the wrong extremum.