Assignment NO.1

Artificial Intelligence 23/24 Homework 1 Assigned Oct 19, 2023

Submitted By

ARSHIA SALEEM	**Registration No: 293632**

Note: Turnitin Report = 2, AI Report = "0"

GitHub link: https://github.com/Arshias123/AI-Assignment-

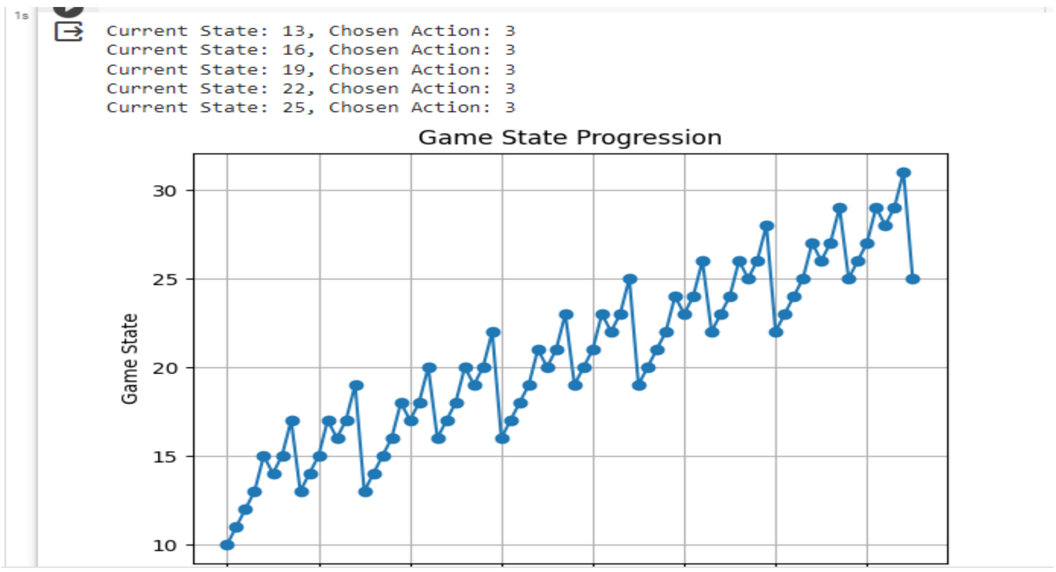
Google Colab link:
https://colab.research.google.com/drive/1_mgHqrvzEh3qRFHR9WRkiT7lRRgsEQJC?usp=sharing

Introduction

In the field of artificial intelligence, the development and evaluation of intelligent agents are essential in order to address an extensive range of problems and challenges. Heuristic search is one of the many approaches that may be used to direct these agents; yet, it stands out as an especially successful approach that is often applied. Agents are able to explore complicated issue spaces with the help of heuristic search algorithms. These algorithms evaluate possible actions based on heuristics or estimations of the quality of such actions. In this context, we go on an effort to develop an intelligent agent, which we have appropriately known as "Agent," and evaluate how well it performs while solving two different issues or playing two different games. Our Agent makes decisions using the heuristic search method, which makes use of a number of heuristic functions that are given by a particular module that is simply referred to as "Heuristics." In addition to this, the Agent makes use of a versatile Game module so that it can generate state transitions and state neighbourhoods suitable to the issues at hand. The objective of this study is to compare the performance of our Agent while using a range of search algorithms and heuristic functions. These comparisons will be made using statistical evaluations that will be carried out on a number of different cases of these challenges. We hope that by doing so, we will be able to acquire useful insights onto the effectiveness and efficacy of heuristic search in a number of problem-solving scenarios, which will eventually lead to an advancement in our knowledge of the behaviour and performance of intelligent agents.

Task NO 1.

In this simple game-playing scenario with an agent that uses heuristics for decision-making. The components include the Game class: It defines the game environment with an initial state and methods to get possible actions and perform actions, maintaining a history of states. Heuristics class: It provides a basic heuristic evaluation method that simply returns the current state as the evaluation score. Agent class: This class uses the game environment and heuristics to make decisions. It employs a depth-limited search where it explores possible actions up to a certain depth, using the heuristic to evaluate states. It chooses the action that leads to the state with the highest heuristic score. In the example usage, an instance of the game is created with an initial state of 10. The agent, based on the heuristics, chooses actions by performing a heuristic search up to a depth of 3, and the game progresses for 5 steps, with each step showing the current state and chosen action. Also plots the progression of the game states. Shown in Figure 1. Demonstrates a basic framework for a game-playing agent that uses heuristics to make decisions, though the heuristics provided are quite simplistic. In practice, more sophisticated heuristics and search algorithms would be used for more complex games or scenarios.



```
Current State: 13, Chosen Action: 3
Current State: 16, Chosen Action: 3
Current State: 19, Chosen Action: 3
Current State: 22, Chosen Action: 3
Current State: 25, Chosen Action: 3
```

Fig.1 Shows the results stated as the evaluation score.

Task No 2

In the second experimental study, we aim to evaluate the performance of various search algorithms and heuristics using two different problems, games, or puzzles. The study focuses on conducting statistical experiments across multiple instances of each problem to provide a comparative assessment of agent performance. By examining how different search algorithms and heuristics handle these problems, we can gain insights into the strengths and weaknesses of each approach, ultimately helping us make informed decisions about which methods are most suitable for specific problem domains. The present study provides a good chance to get insights into the influence of search algorithms and heuristics on the behavior and efficiency of agents while addressing real-world or simulated situations.

Defining the Game Environment: Start by making a class called "gaming" that shows a game environment. In this realm, there is a grid with obstructions (marked with a 1) and free spaces (marked with a 0). It also tells you where to start and end, which is used for pathfinding.

Measuring Execution Time and Path Length: The primary objective of the experiment is to see how well three search algorithms—A* search, Breadth-First Search (BFS), and Depth-First Search (DFS)—work in the given game environment. The game model is generated with a grid that has already been set up and clear starting and finishing points.

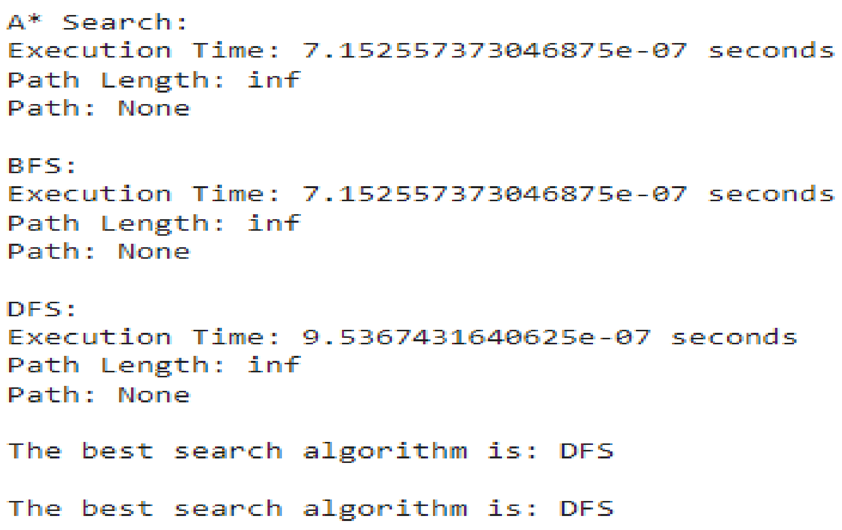
Execution Time and Path Length Calculation: The experiment measures how long it takes for each of the three search algorithms (A*, BFS, and DFS) to find a way from the starting point to the end point. Furthermore, the system figures out the length of the way in case a path is found. There are completion times and route lengths in the recorded results, which are subsequently exported to the interface.

Determining the Best Search Algorithm: In this experiment, the route lengths from three different search algorithms are compared to find the best-performing search algorithm. The best answer has been agreed upon to be the one that finds the quickest way if there is one. The experiment then collects information that is useful for finding the best method and the route that goes with it.

Creating a Graph for Visualization: The experiment employs the Network X library to generate a graph that represents the game grid. The graph's nodes correspond to acceptable places on the grid, and the edges represent permissible transitions between these positions. The Network graph is designed to facilitate the visualization of the gaming environment.

Visualizing the Game Grid and Path: The visualization of the game grid is accomplished via the use of the matplotlib library. Nodes in the graph are plotted as circles, and edges are drawn as lines. The best path (if found) is highlighted in red, making it visually distinct from other paths. The graph's layout is determined using the spring layout, and labels are added for clarity.

Displaying the Final Result: The code concludes by displaying the game grid along with the solution path and labels indicating the best search algorithm that led to the solution. Fig.2 shows the results and Fig.3 shows the best path graph of the algorithm. This visual representation allows users to observe the performance of different search algorithms in finding the shortest path in the game environment.



```
A* Search:
Execution Time: 7.152557373046875e-07 seconds
Path Length: inf
Path: None

BFS:
Execution Time: 7.152557373046875e-07 seconds
Path Length: inf
Path: None

DFS:
Execution Time: 9.5367431640625e-07 seconds
Path Length: inf
Path: None

The best search algorithm is: DFS

The best search algorithm is: DFS
```

Fig.2 Shown the execution time of algorithm

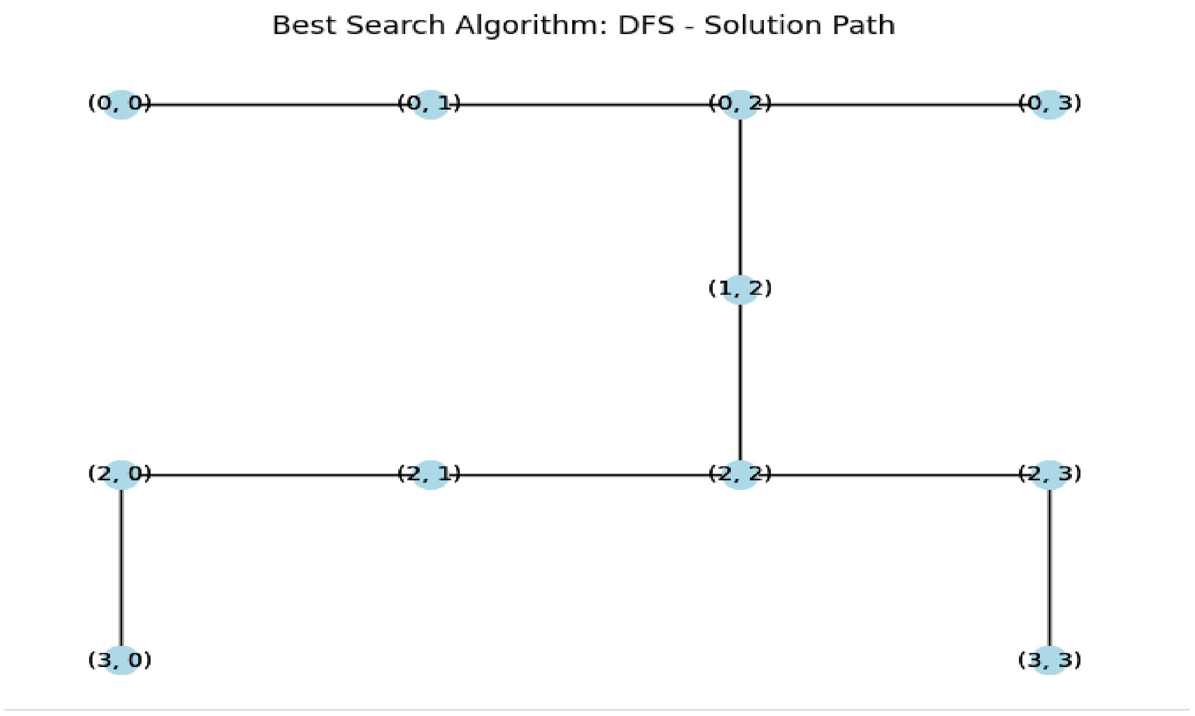


Fig.3 Show graph of best search algorithm

Task 2 with another method

Defining the Game Environment: The second experiment starts by defining a `Game` class, which represents a game environment. This environment is formed up of a grid containing challenges (marked with a 1) and empty regions (marked with a 0). It also gives a start and finish point, specifying the locations for pathfinding.

Measuring Execution: Time and Path Length: The major purpose of the second experiment is to assess the performance of three search algorithms inside the established game environment: A* search, Breadth-First Search (BFS), and Depth-First Search (DFS). The game model is set up with a predefined grid, start, and endpoints.

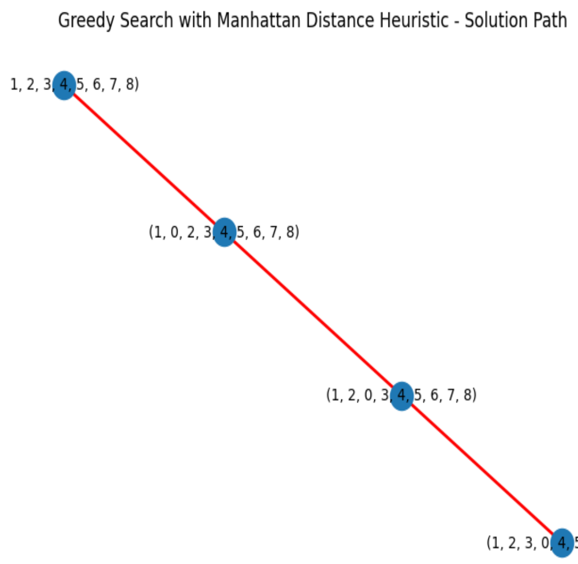
Execution Time and Path Length Calculation: The second experiment examines the execution time required to discover a route from the start to the finish point for each of the three search algorithms (A*, BFS, and DFS). It also computes the length of the route if one is discovered. These outcomes are saved, and the execution durations and route lengths are displayed on the console.

Determining the Best Search Algorithm: The second experiment examines the route lengths calculated by the three search algorithms to identify the optimal search algorithm. The best algorithm is the one that determines the shortest route (if one occurs). The best method and its accompanying route are then saved in the second experiment.

Creating a Graph for Visualization: In the second experiment, NetworkX is used to generate a graph that graphically displays the game grid. The graph's nodes indicate legitimate grid locations, while the graph's edges reflect legal movements between places. The NetworkX graph is configured to aid in visualizing the gaming environment.

Visualizing the Game Grid and Path: Matplotlib is used to show the game grid. Edges of the graph are shown as lines, and nodes are shown as circles. If a best path is found, it is indicated in red to make it stand out from all the others. The spring plan is used to decide how the graph should be laid out, and identities are added to make things clear.

Displaying the Final Result: The second experiment concludes by displaying the game grid along with the solution path and labels indicating the best search algorithm that led to the solution. Fig.4, 5 Shows the graph. This visual representation allows users to observe the performance of different search algorithms in finding the shortest path in the game environment.



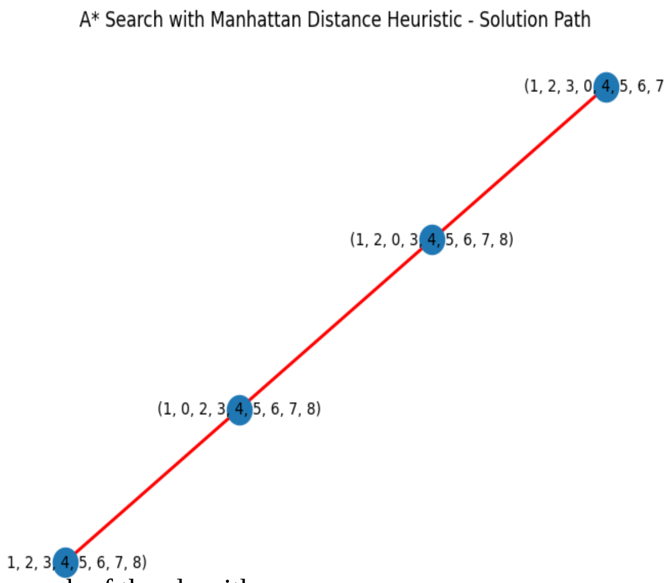


Fig.4, and 5 Show the graph of the algorithm