Comparative Analysis of Pathfinding Algorithms Online DFS and Directed Online DFS on Grid World

Date: 12/18/2023

Ketan Inampudi
Ronit Motwani
Chiemeka Nwakama*
inamp005@umn.edu
motwa11@umn.edu
nwaka013@umn.edu

1 Abstract

The Grid World problem type requires a pathfinding algorithm to navigate an agent from a start to a goal node on a grid. The choice of algorithm impacts the agent's route and computational efficiency. This study compared the Online Depth First Search (Online DFS) algorithm with a directed variant of this algorithm for Grid World based on processing time and spaces explored. The Online DFS algorithm performs a depth-first search to incrementally discover a path to the goal. The directed version incorporates heuristic guidance on exploration direction. The comparison was conducted on a 25x25 square grid with a randomized layout each time, forcing the agent to explore a new environment. By running both algorithms on identical Grid World instances, the time elapsed before the goal node was reached was measured, alongside tallying the number of grid spaces explored during computation. Results provide insight into the tradeoffs of informed directional guidance versus more flexible exploratory behavior when solving randomized 25x25 Grid World layouts. Findings discuss which algorithm more efficiently solves Grid World across the measured metrics. This supports the selection of the superior approach for path-planning applications.

Keywords: Pathfinding, Online DFS, Directional Huersitic, Grid World

2 Introduction

In the intricate realm of artificial intelligence and robotics, the quest for optimal pathfinding algorithms stands as a pivotal challenge. Navigating the complex landscape of an unexplored environment, where every twist and turn presents a new set of decisions, demands not only computational prowess but also strategic insight. In an era where artificial intelligence propels us toward uncharted territories, the significance of pathfinding algorithms cannot be overstated. These digital trailblazers are the architects, guiding not only autonomous robots through complex terrains but also influencing the efficiency of countless applications that weave into our daily lives. The seamless choreography of autonomous vehicles navigating urban jungles, or the meticulous coordination of drones surveying disaster-stricken landscapes are all real-world applications of the algorithms[6] covered within this paper. The very essence of these algorithms lies in their ability to rapidly decipher intricate puzzles to allow for the most effective action. The Online DFS algorithm, employing an incremental depth-first search methodology, is tested against its directed variant, where heuristic guidance dictates the agent's strategic exploration. Each iteration of the agent's search is timed in milliseconds (ms) to quantify its efficiency. Concurrently, we tally the grid spaces traversed, creating a structured comparison between informed guidance and flexible exploration.

2.1 Foundational Pathfinding Algorithms

Pathfinding algorithms are required to determine the fastest path possible. There are several algorithms to find the fastest path. These algorithms are A*, Dijkstra, Breadth First Search (BFS), and Depth First Search (DFS). These algorithms serve as the foundations of pathfinding algorithms. [10] Each of these algorithms possesses unique strengths and weaknesses in the process of isolating the fastest path, though something they have in common is the requirement of full knowledge of the problem. These algorithms, though effective, demand the full data set of every possible move with all possible successor states to identify the fastest path. These foundational algorithms can be characterized as a calculation of set positive and negative values, compared to the iterative building process required to solve the Grid World problem type.

2.2 Research Objectives

This study aims to analyze the efficiency of the Online Depth First Search algorithm as heuristic additions are made to the algorithm's criteria for determining successor states upon completing a given move. The following study will be made in randomized, unknown environments constructed as a 25 square by 25 square Grid World. Online DFS is a graph-search algorithm that explores unknown environments by

*ChatGPT was used to proofread this paper, as well as Grammarly to edit and correct grammatical mistakes.

expanding the deepest node in the search tree first. But, unlike the regular Depth First Search algorithm which assumes full knowledge of the environment, Online DFS conducts the search in an unknown environment while developing its knowledge incrementally as it encounters elements. This algorithm stands out particularly due to its real-world application in complex and uncharted environments especially when compared to its elementary, unrealistic counterpart which requires full knowledge as a prerequisite to complete a search. As addressed in the proceedings "An Improved Algorithm for Searching Maze Based on Depth-First Search" by Ying-Hsuan Chen and Chang-Ming Wu [3], one key factor that impacts the performance of Online DFS is the order in which successor nodes are expanded - as different successor orderings can lead to exploring different parts of the state space first, some far more favorable than others. This problem is an interesting one and is applicable because finding an effective arrangement of successor states can potentially enhance the speed at which Online DFS discovers solutions in unfamiliar maze environments.

2.3 Research Procedures

To investigate this inquiry on the difference in efficiency between the standard Online DFS and a version that intelligently orders successor states, an adaptation of the algorithm code is generated in the Python programming language using structural pseudo code provided by a scholastic textbook called "Artificial Intelligence: A Modern Approach, 4th Edition" by Stuart Russell and Peter Norving [10]. To execute the tracing of each iteration, a table was recorded with columns for iteration number (i), result state (s'), action (a), states going into the iteration (s), as well as dictionaries for results, untried moves, and backtracked moves. Using this method we can begin a timer as the original Online DFS agent begins to solve the maze and populates this table to reach the goal square. Squares moved in the computation process are calculated from the number of squares (nodes) counted before the pathfinder determines its path.

2.4 Research Purpose

The purpose of this research is to identify the possible benefits of appending directional heuristics to the Online Depth First Search algorithm. Comparisons will be made between these algorithms to shine light on future applications of either version considering the data used to determine effectiveness and efficiency. The results are ultimately expected to assist further research into the Online DFS algorithm with other possible heuristic additions, and their effects on efficiency.

3 Related Work

3.1 Pathfinding in Strategy Games and Maze Solving Using A* Search Algorithm

A paper related to this research is titled "Pathfinding in Strategy Games and Maze Solving Using A* Search Algorithm" by Barnouti et. al. [1] In this paper on pathfinding in a maze using the A* algorithm, the authors address the challenges that occur with moving an agent in computer games that require navigation through a course of obstacles and walls. Often, the objective in these games is to navigate the shortest path from start to goal state. This is a common problem type in the field of AI, with the agent in an environment known as a maze. Using images to create the maze environment, the author employs A* in an effort to evaluate the effectiveness of the algorithm as it finds the most efficient path in a given set of environments. Environments vary in size and shape but also use 100 images in their construction to ensure randomness. Ultimately, the authors found that A* search is an entirely viable algorithm. Based on their experiment and trials, they found that approximately 85 percent of the time, A* was able to accurately find the shortest path for a given map. This shows that it has potential use in situations outside of a controlled study, with applications in games being mentioned as a substantial usage of the algorithm [1].

There are a few requirements for the pathfinder's solution, primarily, it must provide a way to get from point A to point B. Secondly, the path must work around any obstacles. It must also be the shortest possible path, and finally, it must be able to find the path in a reasonable amount of time. Many algorithms have been trialed to solve shortest-path problems, such as depth-first search, breadth-first search, and, of course, A*. It is a problem that has been at the cornerstone of the development of many games that include situations in which finding the shortest path to an item or even another entity is sought for [1]. Given the authors' research on the topic using A*, their findings serve as a comparative agent for the proposed online depth-first search algorithm. Though the proposed solution does not utilize this approach, it will still serve as a good comparative measure to see how the solution approach stands when it comes to maze-like problems.

3.2 Building an AI to navigate a maze

In an article called "Building an AI to navigate a maze" by Magnus Engström, Engström investigates creating a digital representation of a city block for simulated citizens to navigate. [5] This problem could be quantified as a maze problem with buildings serving as obstacles and streets serving as paths. Thus, the author conducts a pathfinding experiment to explore the development of a foundational pathfinding agent that controls the navigation of the simulated citizens. The author starts with a simple maze environment where agents move randomly. When an agent hits a wall, it gets deactivated. This could be used to represent a dead end as

a result of an agent making an incorrect move toward the goal but into an obstacle. The simulation ends when a few active agents remain, which serve as the seeds for the next generation of agents which evolve based on data collected by the past generation. Evolution comes in the form of improvements to agent movements and strategy. Data from agents with the shortest path to the target area are used to train a neural network model. Agents communicate their observations to the model, and the model provides instructions on efficient navigation of the maze-like environment. [5]

In the article, Engström emphasizes the importance of maze design. Though the proposed solution does not implement a neural network, the article provides key insights into the rationale for the improvement of the search algorithm of an agent in a maze-like environment, particularly heuristic addition. [5] The article's iterative problem-solving approach will be implemented within this experiment.

3.3 Visualising and Solving a Maze Using an Artificial Intelligence Technique

In a research paper titled "Visualising and Solving a Maze Using an Artificial Intelligence Technique", the authors Sagming et. al. present the idea of implementing the Genetic Algorithm for pathfinding maze problems.[9] The Genetic Algorithm is an artificial intelligence technique for solving randomly generated mazes of varying sizes and complexity. The Genetic Algorithm is compared with non-AI techniques, including Depth-First Search, Breadth-First Search, A* Algorithm, Dijkstra Algorithm, and Greedy Best-First Search. The Genetic Algorithm is a method based on natural selection, which is supposed to mimic biological evolution, suitable for solving both constrained and unconstrained problems. [9] The non-AI algorithms, utilizing graph concepts and data structures, are employed to solve randomly generated mazes. The paper includes the results of five experiments, recording the number of steps and time taken to solve the maze for each algorithm. Graphs are plotted to compare the Genetic Algorithm's performance with non-AI search algorithms. Genetic Algorithm consistently found the shortest path in the study but exhibited slower performance than non-AI algorithms for maze dimensions exceeding the size of 10x10. [9] Though the proposed solution does not implement the Genetic Algorithm, this paper will provide good context for how to assess a created/implemented algorithm when compared to many others.

3.4 Heuristic algorithms for reliability estimation based on breadth-first search of a grid tree

The article "Heuristic algorithms for reliability estimation based on breadth-first search of a grid tree" by Chen et al.[2] proposes three new heuristic search algorithms that proved quite interesting. The key idea is to use breadth-first search (BFS) to efficiently traverse the entire input space represented as a grid tree while reducing redundant sampling. The three algorithms are RASSA, LSSRSSA, and RIPSSA. RASSA can guarantee a prescribed accuracy of reliability estimation. LSSRSSA can probe large curvatures on limit-state surfaces. [2] RIPSSA quickly estimates the reliability index. These algorithms are clever because they aim to improve computational efficiency and accuracy compared to conventional search methods which require many samples to cover the input space. Like BFS, Online-DFS attempts to efficiently search an entire graph space by expanding the most promising unvisited node at each step. The heuristics used in RASSA, LSSRSSA, and RIPSSA to guide BFS search seem similar to Online-DFS using heuristics to select the most promising node. Both aim to reduce redundant sampling to improve efficiency. A key difference is that Online-DFS is an online algorithm without prior knowledge of the graph, while these algorithms utilize problem-specific knowledge to construct an effective grid tree to search. Overall, the algorithms demonstrate the power of guided heuristic search to efficiently cover large spaces, much like Online-DFS for graphs. This article provided new insight into how heuristic search can be applied in reliability estimation.

3.5 An Improved Algorithm for Searching Maze Based on Depth-First Search

Another article providing background knowledge is "An Improved Algorithm for Searching Maze Based on Depth-First Search" by Chen et al.[3] This article presents an interesting maze-searching algorithm that builds upon depth-first search. The key idea is to use a depth-first search strategy to fully map an unknown maze environment. Stack data structures store the current frontier of unexplored locations to guide the search; the search completes when all locations have been explored and the stack is empty. The core contribution is an optimization using two simultaneous search agents that exchange information when they intersect. This allows them to jointly explore the maze space faster than a single agent. [3] After the search completes, the shortest path between any points in the maze can be determined from the full map. The multi-agent search idea is quite clever as it essentially parallelizes the exploration. This provided insight into how communication and coordination between distributed searchers can improve efficiency over a solitary search. The algorithm maintains the simple elegance and guarantee of completeness from depth-first search while enhancing performance. It demonstrates how even classic algorithms can be modified to better suit different problems, like maze mapping. In conclusion, the work highlights how creative augmentations to existing techniques can form novel solutions, much like this multi-agent depth-first maze search algorithm.

3.6 Pathfinding in Random Partially Observable Environments with Vision-Informed Deep Reinforcement Learning

In an article titled "Pathfinding in Random Partially Observable Environments with Vision-Informed Deep Reinforcement Learning", Dowling investigates the application of Deep Reinforcement Learning (DRL) to pathfinding challenges in partially observable environments. The study focuses on training Deep Q-Network (DQN) agents, including DQN-GRU and DQN-LSTM, to navigate a 2-dimensional environment with restricted vision. The agent's decisions are based on visual input, aiming to reach a target zone efficiently. The environment is modeled to simulate scenarios like vision-based navigation for autonomous vehicles and drones, where agents have limited visibility [4].

A notable contribution is the introduction of the Seeker environment, which utilizes a solvable grid world map for the random generation of agent, obstacle, and target positions. In this environment, obstacles are represented as 4-sided walls, requiring agents to navigate by turning or moving forward, aligning more closely with realistic constraints. This design choice differentiates Seeker from traditional grid world environments [4]. The paper proposes a novel reward function tailored for effective RL model training in pathfinding scenarios. The reward function includes proximity to obstacles or boundaries, movement towards or away from the target, and successful target reaching. Each condition is associated with a specific reward value, contributing to the overall learning process [4]. Performance evaluations are conducted on randomized environment maps with variable obstacle numbers during both the training and evaluation phases. The study also explores model performance in scenarios where knowledge of the target location is either known or unknown. Dowling's work distinguishes itself by systematically comparing DQN models with recurrent counterparts, providing insights into the strengths and weaknesses of each approach [4]. Dowling's study carries on work from existing literature by incorporating depth-based visual information and offering a systematic comparison of DQN models in partially observable environments [4].

3.7 Pathfinding Algorithms in Game Development

In an article by Abdul Rafiq et al. titled "Pathfinding Algorithms in Game Development", Rafiq offers a comprehensive exploration of pathfinding algorithms and their impact on game development. The primary focus is on categorizing algorithms based on their search performance, coupled with an examination of developments over the last decade. The authors are affiliated with the Soft Computing and Intelligent System Research Group at the University Malaysia Pahang defining them as a reputable source [8].

In the introduction, the paper delves into the growing research interest in addressing artificial intelligence challenges within video games, covering aspects such as decision-making, movement, strategy, and pathfinding. The concept of pathfinding is characterized as the plotting of a node to determine the shortest or minimum path between two points, which is applicable in various domains such as video games, robotics, crowd simulation, and GPS. The paper demonstrates the significance of optimal pathfinding for achieving realistic Non-Playable Character (NPC) behavior in video games. Pathfinding algorithms like Dijkstra, A* algorithm, genetic algorithms, and ant colony optimization are discussed, alongside an exploration of static and dynamic (real-time) pathfinding scenarios [8].

Heuristic algorithms aimed at efficiently finding fast and good solutions but necessarily optimal to specific problems are introduced. This article, like many of the others, showcases the A* algorithm, which is no surprise given its popularity in pathfinding due to its accuracy and performance, emphasizing the pursuit of the minimum value or lowest path for optimal solutions. Dijkstra's algorithm is presented as a classic graph search algorithm specializing in finding the shortest path [8].

The A* algorithm is portrayed as selecting the minimum solution by identifying the shortest path. Its utilization of a heuristic function to evaluate the lowest path, incorporating the cost to reach the target node and the heuristic value, contributes to its effectiveness. The algorithm is favored for its simplicity and ease of understanding in solving pathfinding problems. Dijkstra's algorithm is characterized as a classic graph search algorithm proficient in finding the shortest path. The algorithm's approach involves visiting the path with the smallest distance from the starting node and updating the shortest path for each neighboring node [8].

This paper concludes with the assessment that metaheuristic (heuristics that are self-generative and self adaptable depending on the given problem) techniques demonstrate superior performance in terms of time and memory when compared to heuristic techniques within the realm of pathfinding algorithms for game development [8]. This will be something that will be good to consider during the generation of the proposed solution of a pathfinding algorithm.

3.8 Comparative Analysis of Pathfinding Algorithms A *, Dijkstra, and BFS on Maze Runner Game

A research paper by Permana et. al. explores the efficiency of pathfinding algorithms in the context of a Maze Runner game. [7] The study compares three algorithms – A*, Dijkstra, and Breadth First Search – by measuring the processing time, path length, and the number of blocks played during the computation process. The goal is to identify the most suitable algorithm for the Maze Runner game. The Maze Runner game involves an NPC (Non-Player Character) agent

navigating a maze from a start node to a destination node, encountering Tetris-like obstacles placed by players. The study aims to determine the algorithm that provides the shortest path for the agent in terms of computational efficiency [7]. The results indicate that A* is identified as the most efficient algorithm for pathfinding in the Maze Runner game. The evaluation is based on factors such as the minimal computing process needed, relatively short searching time, and lower memory consumption compared to Dijkstra and BFS algorithms. [7]

The research methodology involves a comparative test using three levels of the Maze Runner game, each with different obstacle layouts. The algorithms are tested based on the process time, path length, and the number of blocks played in the computation process. The study concludes that the choice of the right algorithm significantly impacts the game's computational process, memory usage, and computing time. [7]

The paper suggests further research avenues, such as modifying the A* algorithm for improved results and testing the algorithm in different gaming scenarios beyond the maze context. [7] This paper will serve as a guide to how algorithms can be compared against one another. Overall, the research provides valuable insights into the performance of pathfinding algorithms in the specific context of a Maze Runner game, offering practical recommendations for researchers to continue the exploration of maze-based games.

4 Methods

Two versions of the online_dfs algorithm were written, one where the order in which neighbors are explored is predetermined, and another where the order in which neighbors are explored is dependent on the current location of the agent and the goal. Both versions were heavily based on the pseudo-code provided in the AIMA textbook [10]. Both algorithms include some of the following functions: make_list, isGoal, result, test_result, possible_actions, find_back_action, solve, and online_dfs_algorithm. However, only possible_actions and make_list will need to be changed between the two versions.

4.1 Global Variables

Both implementations have certain global locations that are necessary for the program to work. The basic global variables include environment, loc, goal, width, and height. environment is a dictionary where the key is an x, y tuple representing a point on the maze, while the value is a list of characters representing where the walls are relative to the location it relates to. The list of characters would be U, D, L, and R, U stands for a wall above the location, D stands for a wall below the location, and so on. loc is a tuple that represents the current location of the agent, this variable is constantly changing. goal is a tuple that represents the location of the goal, this variable does not change. width and height are single integers that represent the width and height of the maze, usually, these values should be equal. "Results", "Untried", and "Unbacktracked" are the final three global variables that are each dictionaries. First, "results" stores a visited location with the location it came from and the action used to get there. Next, "untried" stores locations with a list of the possible actions the agent could take from that location. Lastly, "unbacktracked" stores visited locations along with the locations taken to get there.

4.2 Unchanging Functions

There are many functions that allow the implementations to work properly and are unchanged between the two versions. The unchanged functions are: result, test_result, possible_actions, isGoal, and find_back_action all work in a fixed way. The result function will globally change the location of the agent based on the action passed into the function. test_result works similarly, but will only temporarily change the location of the agent without actually changing its global location. possible_actions function will check the neighboring locations for walls or out-of-bounds problems and make a list of the possible actions the agent can make from its current location. The isGoal function will simply check if the global location is at the goal location. find_back_action takes in two neighboring locations on the maze, the first is the location of the agent, and the second is where the agent wants to go. The function will loop through the four possible actions the agent could take and find which one will lead the agent to where it wants to be.

4.3 Main online_dfs algorithm description

The code for the functions solve, and online_dfs are technically unchanged and are heavily based on the pseudo-code given in the AIMA textbook. solve is a wrapper function that saves some basic information and makes the main loop that runs online_dfs. Variables such as the current location, previous location, action, and the list of all actions taken are saved in solve. Additionally, there is a loop that continuously runs online_dfs and updates the aforementioned variables until the agent is at the goal. How online_dfs works is talked about extensively in the AIMA textbook, along with the pseudo-code, and will not be discussed here.

4.4 Key Functions to Change Implementation

Apart from the functions mentioned thus far, the function that causes the two versions to work differently is make_list. In the undirected version, make_list will randomly generate a list of the four unique directions that the agent can go in, this list is then passed into possible_actions and the rest of the code runs as described above. In the directed version however, the function will calculate the difference between the x and y values of the agent's current location to the goal and will make a list that prioritizes the direction that is

more important to moving towards the goal. In the directed version this function is called every time possible_actions is called, so every time a new location is explored the directed list for that location is made. If the agent revisits the location (and is not backtracking) then the agent will already know which direction is more important.

4.5 Evaluation Functions

The two algorithms are run through an evaluator that could be set to run several simulations. Taking this statistical approach of running the algorithms in multiple sets of separated simulation numbered instances will give credibility and evidence to the results generated. It will control for variability that could occur by just conducting the evaluation a single or few times. For a given maze size, several simulations would be running, and then the average actions for each of the algorithms would be recorded through the duration of the simulations. The averages would be returned, as well as the actions taken for each of the unique maze simulations and the number of actions in each simulation, so you can see specifically how well each algorithm ran in a given simulation.

4.6 Unsolvable Cases

Not all mazes generated are solvable, so to remedy this issue, mazes were first checked to see if they were solvable before being tested on the two dfs algorithms in a given simulation. Checking if a randomly generated maze was solvable was based on time in terms of whether or not the maze was solvable within a certain time frame. If not, the maze would be deemed unsolvable, and a new maze would be generated and benchmarked using the same procedure. The performance will be based on the conduction of four separate trials of 5, 10, 15, and 20 simulations of 25x25 mazes. This will help show if an algorithm does well in short-term vs. long-term scenarios, testing the consistency of the approaches. A text file will show each simulation how the two algorithms compare in terms of the number of actions taken and the amount of time it took to reach the mazes' goals. At the end of the file would be the average actions taken between the two algorithms in solving all N number of maze simulations as well as their average time to finish a given maze. Based on these results, both the algorithms will be accessed in terms of fewer actions and quicker solutions being better. Through analysis, insights about the algorithms will be learned, and through analysis, a better algorithm is expected to be determined.

5 Results

5.1 Experiment Design

As mentioned earlier, the experiment was conducted on randomly generated mazes with dimensions of 25 by 25. Not all generated mazes are guaranteed to be solvable. During the trials and simulations, each maze was assessed for solvability within a time limit of four seconds before attempting to solve it using either the vanilla, unchanged Online Depth First Search or the heuristic, Directed Online Depth First Search. If the maze could not be solved within four seconds, the solver timed out and generated a new random maze. It is acknowledged that using only online dfs to check for solvability could introduce bias towards mazes that online dfs can solve quickly; however, as each of these algorithms is meant to be able to perform well in any environment they are given and because directed online dfs has the expectation of being better on average, despite this condition, this concern can be resigned.

Trials were conducted using two dfs algorithms: undirected dfs, representing pure Online Depth First Search, and directed dfs, which was Online Depth First Search with the distinction that, when determining possible actions, choices were based on the proximity of the agent to the goal. Based on this information and the location the agent is at, the algorithm will allow the agent to try to more strategically and, in theory, accurately pick actions that will lead it to get to the goal quicker than it would have if it did not have said heuristic.

Through experimentation, each trial (trial 1, trial 2, trial 3, and trial 4) consisted of 5, 10, 15, and 20 simulations, respectively. Parameters were set for the wait time for deeming a maze to be unsolvable to four seconds; the maze size was set to 25 by 25, and the number of simulations was set to their appropriate values per trial order. As mentioned previously in the methods, all results shown below are the outputs of each trial's text document. The averages are displayed below, though resulting text documents were not limited to just having trials' averages, including the number of actions for each respective search algorithm as well as the time taken to find a goal/complete search in each simulation within a given trial.

The results from trial to trial varied quite a bit:

5.2 Trial 1

————————————————-
Maze size: 25 x 25
Wait time before deemed unsolvable = 4 seconds
Simulation 0
Non-directed DFS Number of Actions: 716
Directed DFS Number of Actions: 372
Non-directed DFS Solve Time: 0.01590418815612793
Directed DFS Solve Time: 0.009135246276855469
————————————————-

Simulation 1
Non-directed DFS Number of Actions: 674
Directed DFS Number of Actions: 682
Non-directed DFS Solve Time: 0.015892744064331055

Directed DFS Solve Time: 0.015948772430419922
———————————————-

Simulation 2
Non-directed DFS Number of Actions: 60
Directed DFS Number of Actions: 66
Non-directed DFS Solve Time: 0.016222238540649414
Directed DFS Solve Time: 0.011983871459960938
———————————————-
Simulation 3
Non-directed DFS Number of Actions: 1072
Directed DFS Number of Actions: 308
Non-directed DFS Solve Time: 0.13164973258972168
Directed DFS Solve Time: 0.012555837631225586
———————————————-
Simulation 4
Non-directed DFS Number of Actions: 256
Directed DFS Number of Actions: 314
Non-directed DFS Solve Time: 0.10082769393920898
Directed DFS Solve Time: 0.01634836196899414

Ran for 5 simulations
Avg Non-directed DFS Number of Actions: 555
Avg Directed DFS Number of Actions: 348
Avg Non-directed DFS Solve Time: 0.056099319458007814
Avg Directed DFS Solve Time: 0.013194417953491211

5.3 Trial 2

Maze size: 25 x 25
Wait time before deemed unsolvable = 4 seconds
Simulation 0
Non-directed DFS Number of Actions: 116
Directed DFS Number of Actions: 46
Non-directed DFS Solve Time: 0.0011301040649414062
Directed DFS Solve Time: 0.0
———————————————-
Simulation 1
Non-directed DFS Number of Actions: 988
Directed DFS Number of Actions: 644
Non-directed DFS Solve Time: 0.002134561538696289
Directed DFS Solve Time: 0.06246223
———————————————-
Simulation 2
Non-directed DFS Number of Actions: 974
Directed DFS Number of Actions: 254
Non-directed DFS Solve Time: 0.0
Directed DFS Solve Time: 0.029246223
———————————————-
Simulation 3
Non-directed DFS Number of Actions: 970
Directed DFS Number of Actions: 1230
Non-directed DFS Solve Time: 0.03183770179748535
Directed DFS Solve Time: 0.03936338424682617
———————————————-
Simulation 4
Non-directed DFS Number of Actions: 732
Directed DFS Number of Actions: 706
Non-directed DFS Solve Time: 0.01622462272644043
Directed DFS Solve Time: 0.03170347213745117
———————————————-
Simulation 5
Non-directed DFS Number of Actions: 1182
Directed DFS Number of Actions: 2458
Non-directed DFS Solve Time: 0.051444053649902344
Directed DFS Solve Time: 0.6960771083831787
———————————————-
Simulation 6
Non-directed DFS Number of Actions: 228
Directed DFS Number of Actions: 314
Non-directed DFS Solve Time: 0.07120847702026367
Directed DFS Solve Time: 0.26210713386535645
———————————————-
Simulation 7
Non-directed DFS Number of Actions: 1008
Directed DFS Number of Actions: 362
Non-directed DFS Solve Time: 0.10730814933776855
Directed DFS Solve Time: 0.5914697647094727
———————————————-
Simulation 8
Non-directed DFS Number of Actions: 662
Directed DFS Number of Actions: 214
Non-directed DFS Solve Time: 0.007985115051269531
Directed DFS Solve Time: 0.5003049373626709
———————————————-
Simulation 9
Non-directed DFS Number of Actions: 108
Directed DFS Number of Actions: 90
Non-directed DFS Solve Time: 0.016205787658691406
Directed DFS Solve Time: 0.03222799301147461
———————————————-
Ran for 10 simulations
Avg Non-directed DFS Number of Actions: 696
Avg Directed DFS Number of Actions: 631
Avg Non-directed DFS Solve Time: 0.0305478572845459
Avg Directed DFS Solve Time: 0.21532537937164306

5.4 Trial 3

———————————————-
Maze size: 25 x 25
Wait time before deemed unsolvable = 4 seconds
Simulation 0
Non-directed DFS Number of Actions: 428
Directed DFS Number of Actions: 1590
Non-directed DFS Solve Time: 0.029266834259033203

Directed DFS Solve Time: 0.06406021118164062
——————————————-
Simulation 1
Non-directed DFS Number of Actions: 88
Directed DFS Number of Actions: 1212
Non-directed DFS Solve Time: 0.0874488353729248
Directed DFS Solve Time: 0.06544327735900879
——————————————-
Simulation 2
Non-directed DFS Number of Actions: 192
Directed DFS Number of Actions: 1290
Non-directed DFS Solve Time: 0.05620098114013672
Directed DFS Solve Time: 0.14310455322265625
——————————————-
Simulation 3
Non-directed DFS Number of Actions: 746
Directed DFS Number of Actions: 1236
Non-directed DFS Solve Time: 0.050463199615478516
Directed DFS Solve Time: 0.014934301376342773
——————————————-
Simulation 4
Non-directed DFS Number of Actions: 210
Directed DFS Number of Actions: 524
Non-directed DFS Solve Time: 0.07548999786376953
Directed DFS Solve Time: 0.15024852752685547
——————————————-
Simulation 5
Non-directed DFS Number of Actions: 110
Directed DFS Number of Actions: 1352
Non-directed DFS Solve Time: 0.07361197471618652
Directed DFS Solve Time: 0.08857059478759766
——————————————-
Simulation 6
Non-directed DFS Number of Actions: 342
Directed DFS Number of Actions: 1600
Non-directed DFS Solve Time: 0.13059759140014648
Directed DFS Solve Time: 0.07336783409118652
——————————————-
Simulation 7
Non-directed DFS Number of Actions: 1086
Directed DFS Number of Actions: 2130
Non-directed DFS Solve Time: 0.27042508125305176
Directed DFS Solve Time: 0.057463645935058594
——————————————-
Simulation 8
Non-directed DFS Number of Actions: 558
Directed DFS Number of Actions: 2048
Non-directed DFS Solve Time: 0.06302118301391602
Directed DFS Solve Time: 0.22019696235656738
——————————————-
Simulation 9
Non-directed DFS Number of Actions: 688
Directed DFS Number of Actions: 1306
Non-directed DFS Solve Time: 0.5256531238555908
Directed DFS Solve Time: 0.20340228080749512
——————————————-
Simulation 10
Non-directed DFS Number of Actions: 384
Directed DFS Number of Actions: 1956
Non-directed DFS Solve Time: 0.0160675048828125
Directed DFS Solve Time: 0.45796895027160645
——————————————-
Simulation 11
Non-directed DFS Number of Actions: 520
Directed DFS Number of Actions: 592
Non-directed DFS Solve Time: 0.18042778968811035
Directed DFS Solve Time: 0.01663661003112793
——————————————-
Simulation 12
Non-directed DFS Number of Actions: 728
Directed DFS Number of Actions: 1276
Non-directed DFS Solve Time: 0.23483514785766602
Directed DFS Solve Time: 0.06553363800048828
——————————————-
Simulation 13
Non-directed DFS Number of Actions: 282
Directed DFS Number of Actions: 1498
Non-directed DFS Solve Time: 0.007910490036010742
Directed DFS Solve Time: 0.5828523635864258
——————————————-
Simulation 14
Non-directed DFS Number of Actions: 384
Directed DFS Number of Actions: 1068
Non-directed DFS Solve Time: 0.17980480194091797
Directed DFS Solve Time: 0.28311872482299805
——————————————-
Ran for 15 simulations
Avg Non-directed DFS Number of Actions: 449
Avg Directed DFS Number of Actions: 1378
Avg Non-directed DFS Solve Time: 0.13208163579305013
Avg Directed DFS Solve Time: 0.16579349835713705

5.5 Trial 4

——————————————-
Maze size: 25 x 25
Simulation 0
Non-directed DFS Number of Actions: 576
Directed DFS Number of Actions: 1236
Non-directed DFS Solve Time: 0.031400442123413086
Directed DFS Solve Time: 0.050418853759765625
——————————————-
Simulation 1
Non-directed DFS Number of Actions: 4
Directed DFS Number of Actions: 1248
Non-directed DFS Solve Time: 0.0160062313079834
Directed DFS Solve Time: 0.03188371658325195

——————————————-
Simulation 2
Non-directed DFS Number of Actions: 962
Directed DFS Number of Actions: 1034
Non-directed DFS Solve Time: 0.01585698127746582
Directed DFS Solve Time: 0.06302094459533691
——————————————-
Simulation 3
Non-directed DFS Number of Actions: 970
Directed DFS Number of Actions: 2468
Non-directed DFS Solve Time: 0.044342756271362305
Directed DFS Solve Time: 0.17979836463928223
——————————————-
Simulation 4
Non-directed DFS Number of Actions: 868
Directed DFS Number of Actions: 2466
Non-directed DFS Solve Time: 0.09551835060119629
Directed DFS Solve Time: 0.2327406406402588
——————————————-
Simulation 5
Non-directed DFS Number of Actions: 2
Directed DFS Number of Actions: 1582
Non-directed DFS Solve Time: 0.026668310165405273
Directed DFS Solve Time: 0.07271218299865723
——————————————-
Simulation 6
Non-directed DFS Number of Actions: 694
Directed DFS Number of Actions: 1536
Non-directed DFS Solve Time: 0.06531310081481934
Directed DFS Solve Time: 0.03240060806274414
——————————————-
Simulation 7
Non-directed DFS Number of Actions: 1210
Directed DFS Number of Actions: 876
Non-directed DFS Solve Time: 0.06835007667541504
Directed DFS Solve Time: 0.01601099967956543
——————————————-
Simulation 8
Non-directed DFS Number of Actions: 646
Directed DFS Number of Actions: 712
Non-directed DFS Solve Time: 0.07885622978210449
Directed DFS Solve Time: 0.015740394592285156
——————————————-
Simulation 9
Non-directed DFS Number of Actions: 14
Directed DFS Number of Actions: 1258
Non-directed DFS Solve Time: 0.14056086540222168
Directed DFS Solve Time: 0.05650639533996582
——————————————-
Simulation 10
Non-directed DFS Number of Actions: 1078
Directed DFS Number of Actions: 1410
Non-directed DFS Solve Time: 0.2463207244873047
Directed DFS Solve Time: 0.32435059547424316
——————————————-
Simulation 11
Non-directed DFS Number of Actions: 460
Directed DFS Number of Actions: 2458
Non-directed DFS Solve Time: 0.03126955032348633
Directed DFS Solve Time: 0.41860318183898926
——————————————-
Simulation 12
Non-directed DFS Number of Actions: 6
Directed DFS Number of Actions: 1394
Non-directed DFS Solve Time: 0.20412135124206543
Directed DFS Solve Time: 0.2360382080078125
——————————————-
Simulation 13
Non-directed DFS Number of Actions: 700
Directed DFS Number of Actions: 1092
Non-directed DFS Solve Time: 0.2852959632873535
Directed DFS Solve Time: 0.016021251678466797
——————————————-
Simulation 14
Non-directed DFS Number of Actions: 218
Directed DFS Number of Actions: 356
Non-directed DFS Solve Time: 0.8778567314147949
Directed DFS Solve Time: 0.07883644104003906
——————————————-
Simulation 15
Non-directed DFS Number of Actions: 14
Directed DFS Number of Actions: 1594
Non-directed DFS Solve Time: 0.023754119873046875
Directed DFS Solve Time: 0.19286060333251953
——————————————-
Simulation 16
Non-directed DFS Number of Actions: 6
Directed DFS Number of Actions: 1362
Non-directed DFS Solve Time: 0.12900114059448242
Directed DFS Solve Time: 0.08197784423828125
——————————————-
Simulation 17
Non-directed DFS Number of Actions: 588
Directed DFS Number of Actions: 1344
Non-directed DFS Solve Time: 0.899477481842041
Directed DFS Solve Time: 0.04686784744262695
——————————————-
Simulation 18
Non-directed DFS Number of Actions: 102
Directed DFS Number of Actions: 758
Non-directed DFS Solve Time: 0.04542851448059082
Directed DFS Solve Time: 0.10390615463256836
——————————————-
Simulation 19
Non-directed DFS Number of Actions: 532
Directed DFS Number of Actions: 1764
Non-directed DFS Solve Time: 0.01673579216003418
Directed DFS Solve Time: 1.7358248233795166

———————————————-
Ran for 20 simulations
Avg Non-directed DFS Number of Actions: 482
Avg Directed DFS Number of Actions: 1397
Avg Non-directed DFS Solve Time: 0.16710673570632933
Avg Directed DFS Solve Time: 0.19932600259780883

———————————————-

Trial 1, was conducted with a 25 x 25 maze in each of it's five simulations, the simulations had a wait time of 4 seconds before deeming the maze unsolvable. The average number of actions for Non-directed Depth-First Search (DFS) was 555, while for Directed DFS, it was 348. The average solve time for Non-directed DFS was 0.0561 seconds, whereas Directed DFS achieved a faster solve time of 0.0132 seconds.

Trial 2 was also conducted using a 25 x 25 maze in each of the 10 simulations with a 4-second wait time for unsolvability. This trial consisted of 10 simulations, resulting in an average number of actions for Non-directed DFS and Directed DFS at 696 and 631, respectively. The solve time for Non-directed DFS was 0.0305 seconds, whereas Directed DFS took longer with an average solve time of 0.2153 seconds.

In Trial 3, again was conducted with a 25 x 25 randomly generated mazes and a 4-second wait time; 15 simulations were conducted. The average number of actions for Non-directed DFS was 449, whereas for Directed DFS, it was significantly higher at 1378. The solve time for Non-directed DFS was 0.132 seconds, while Directed DFS had a slightly lower solve time of 0.166 seconds.

Finally, in Trial 4, which also utilized 25 x 25 randomly generated mazes with a 4-second wait time, 20 simulations were carried out. The average number of actions for Non-directed DFS and Directed DFS were 482 and 1397, respectively. The solve time for Non-directed DFS was 0.167 seconds, and for Directed DFS, it was 0.199 seconds.

6 Analysis

Analyzing the results of the four different trials, it is clear that it is difficult to declare a clear winner between the two trials. This is due to results not being consistent as in trials 1 and 2.

6.1 Trial 1 Analysis

Directed dfs had taken fewer actions on average in trial 1, which had five simulations. Directed dfs had taken 207 fewer actions at 348 actions than regular online dfs at 555 actions, respectively. Knowing that directed dfs performed significantly fewer actions to find solutions in the five generated mazes, it is to no surprise that directed dfs had outperformed regular dfs in speed as well, being almost twice as fast at 0.0132 seconds compared to dfs's 0.0561 seconds. From the trial, it seems that directed dfs is the clear winner; however, in the next trials, it starts not look that way anymore.

6.2 Trial 2 Analysis

In trial 2, the two solve, search algorithms were run and evaluated on 10 simulations. Though directed dfs performed 68 fewer actions on average with 631 actions to undirected dfs' 696 actions respectively. Surprisingly, non-directed solve time was faster on average at 0.031 seconds compared to directed's .215 seconds. It is expected that the more actions a search algorithm performs, the slower it will be; however, it is possible that how the directed search determines the possible actions using its heuristic is slower or that some sort of processing slows down within a certain margin of error could have occurred outside of the algorithms control. This all means that the issue could be a mix of both or potentially just be caused by random chance when the algorithm ran for directed dfs or even potentially undirected dfs. This is not out of the realm of possibility, being quite probable since it takes only a few seconds or less to solve each maze, so even a slight slowdown caused by chance or other factors outside of the algorithm could significantly impact how much time it took to solve the maze. As such, an issue could arise if only basing performance off of solve time. To remedy this, it is assumed that both dfs algorithms scale the same in their approaches, making it more useful to analyze the number of actions taken instead of time or time alone. Time is still a component of interest; however, primarily, the main performance benchmark is the number of actions, as it can indicate the amount of "wrong" moves each search algorithm makes. As such, in trial 2 directed dfs performed better on average since directed dfs performed 68 fewer actions than its undirected counterpart.

6.3 Trial 3 Analysis

In trial 3, there was a change in the trend seen in trials 1 and 2. Trial 3 had the two algorithms run and evaluated on, now, 15 simulations. In this case, there was a clear winner, with undirected dfs performing only 449 actions as opposed to directed's 1378 actions on average: a 998 action difference. In time, undirected dfs was faster, with an average of 0.132 seconds compared to directed dfs' 0.166 seconds. Note that previously, undirected search, on average, took even fewer actions to solve the maze than the previous two trials at only 449 actions in trail 3 compared to the 600+ actions on average it performed in trails 1 and 2. This is not unexpected as mazes are completely randomly generated; thus, numbers are expected to not match up from trial to trial. However, compared to the average number of actions taken, directed dfs performs remarkably worse, more than four times worse than undirected dfs. Based on the big difference in the number of actions, it is possible that the heuristic is inconsistent, as doing four times worse on average would suggest that

the heuristic may not always be satisfactory for solving unknown mazes. It can also be thought that the reason why this may be occurring may be due to the heuristic not being admissible, an admissible heuristic being one that does not overestimate path cost. If a heuristic is not admissible, it would mean that many times when the agent is performing directed dfs, for example, it could be getting misled, overestimating the cost to the goal given a certain action, which would make it so that the agent would take paths that would not lead to the goal in a reasonable amount of actions. The agent may be running into walls and dead ends frequently due to the heuristic, making it so that it has to perform many actions to "make up" for its "bad" decisions (backtracking) and get back on track to the goal.

6.4 Trial 4 Analysis

Trial 4 had a similar result to trial 3. Trial 4 had the two algorithms run and evaluated on 20 simulations. The clear winner, once again, was undirected dfs performing only 482 actions as opposed to directed's 1397 actions on average: a 915 action difference. In time, undirected dfs was faster, with an average 0.167 seconds compared to directed dfs' 0.199 seconds. The same situation that happened in trial 3 occurs in this trial displaying a huge difference in the number of actions between the two algorithms.

6.5 Analyzing Heuristic Inconsistency

From all trials with widely varying results, the heuristic has been shown to be inconsistent in how dfs with the heuristic (directed dfs) outperformed dfs without heuristic (undirected dfs) in trial 1 by a sizable margin and in trial 2l by not too much. Trial 3 and 4 showed that the directed dfs (dfs with heuristic) performed worse than regular online dfs by a large amount. From this, it can be speculated that the heuristic is not admissible and is overestimating path cost for optimal or suboptimal paths on the maze some of the time, which leads to results going either way. For clarity, suppose the probability of the directed dfs being better than regular dfs is .5 (random chance), then it would make sense that in trials 1 and 2, when taking an average, it is more likely that there could be more times that directed dfs is better than the regular dfs online thus making it perform better than average in small simulation settings potentially. Once the simulation increased, however, it was seen that directed dfs no longer performed better and did remarkably worse. The heuristic being not the best and especially it not being admissible too could explain the inconsistency; however, there could always be other reasons, though this is a reasonable deduction. To further show this happening within a single trial, take for example, simulations 0, 1, and 3 in trial 1:

——————————————-
Simulation 0
Non-directed DFS Number of Actions: 716
Directed DFS Number of Actions: 372
Non-directed DFS Solve Time: 0.01590418815612793s
Directed DFS Solve Time: 0.009135246276855469s

——————————————-
Simulation 1
Non-directed DFS Number of Actions: 674
Directed DFS Number of Actions: 682
Non-directed DFS Solve Time: 0.015892744064331055s
Directed DFS Solve Time: 0.015948772430419922s

——————————————-
Simulation 3
Non-directed DFS Number of Actions: 1072
Directed DFS Number of Actions: 308
Non-directed DFS Solve Time: 0.13164973258972168s
Directed DFS Solve Time: 0.012555837631225586s
——————————————-

Even though trial 1 overall had directed dfs perform better on average, in simulation 1, nondirected dfs slightly outperformed directed dfs with nondirected dfs' 674 actions compared to directed dfs' 682 actions. Nondirected dfs also performed slightly faster at around .001 seconds faster than its heuristic-based counterpart. The reason why nondirected dfs did not perform better on average is that the extent to which it outperformed its directed counterpart in given simulations just did not happen enough times, or it did not outperform directed dfs with enough magnitude the times it did do better for nondirected dfs to outperform directed dfs overall. The disparity in performance between directed and nondirected dfs in simulation 1 of trial 1 raises questions about the reliability and effectiveness of the implemented heuristic. However, as directed dfs still perform better than nondirected dfs in the trial overall, it indicates that in trial 1, using five simulations of randomly generated 25 by 25 mazes to solve, directed dfs performs better than nondirected dfs.

However, Given more simulations, as seen in trials 3 and 4, despite trial 1 showcasing a favorable outcome for directed dfs, these subsequent trials revealed a decline in its performance. The recurring occurrence of scenarios similar to trial 1's simulation 1 in trials 3 and 4 suggests that the heuristic seems to not be consistently effective across various settings and conditions.

Through analysis, it would seem that the heuristic implemented in directed dfs is not a good choice as it seems to not be admissible and, as a result, seemingly performs no better than a random chance heuristic. To this end, analysis of the heuristic's admissibility is central to understanding its limitations. The lack of reliability observed in its performance indicates that the heuristic does not reliably provide accurate estimates of path costs. This inconsistency is a key factor contributing to the fluctuating results seen across trials

and simulations. The observed discrepancies in performance between directed and nondirected dfs, particularly in simulation 1 of trial 1, highlight potential shortcomings in the employed heuristic. The lack of admissibility in the heuristic raises concerns about its reliability and effectiveness in guiding the dfs algorithm. As future work unfolds, addressing these concerns and exploring alternative heuristics could lead to more consistent and reliable performance in various simulation scenarios. As it stands in this study, the heuristic, though seeming promising, has shown to not be as effective as hope and has instead proved to be quite detrimental to the performance of Online Depth First Search in a nonobservable maze setting. This is not to say that using a heuristic in general is not a beneficial idea; however, further research outside of the scope of this paper will need to be done to find a suitable heuristic that consistently outperforms the standard Online Depth First Search algorithm.

7 Conclusion

In retrospect, the initial theory of appending heuristic guidance to the Online Depth First Search algorithm would enhance its efficiency in unknown environments was not substantiated through experimentation. While the concept of prioritizing certain successors over others based on proximity to the goal state seems logically sound, the implemented directed heuristic failed to outperform the standard non-guided Online Depth First Search.

By tallying total iterations, computation time, and spaces traversed across multiple randomized Grid World test cases, we gathered ample data to compare the performance of both Online Depth First Search versions. Yet despite seemingly providing more informed path selections, the directed Online Depth First Search averaged nearly 367 more actions, and almost 100 additional milliseconds than its standard counterpart. Clearly, the intended optimization had the opposite effect under the experimental conditions.

While disproving our original hypothesis, these results still prove insightful for future Online Depth First Search research. The fact that informed guidance can actually hinder the algorithm's efficiency hints that more nuanced heuristics are needed. Simply prioritizing moves closer to the goal may overload Online Depth First Search with an excess of information in complex spaces. More balanced integration of long-term strategic nudges, rather than strictly short-term directional cues, could potentially improve performance. Additionally, testing on environments with different properties than our 25 by 25 Grid Worlds may better suit certain heuristics.

Considering the implications of the results, looking at the fact that the directed Online Depth First Search algorithm performed worse than the standard non-guided algorithm suggests that the heuristic used to prioritize certain successors over others based on proximity to the goal state was not effective. This is an important finding because it highlights the need for more nuanced heuristics that take into account the complexity of the environment being navigated.

One possible explanation for why the directed Online Depth First Search algorithm performed worse than the standard algorithm is that it was overloaded with too much information. By prioritizing moves closer to the goal, the algorithm may have been inundated with an excess of information in complex spaces, which could have slowed it down. A more balanced integration of long-term strategic nudges, rather than strictly short-term directional cues, could potentially improve performance.

Another important consideration is that the results of this study were obtained using a 25 by 25 Grid World. It is possible that different results would be obtained using different environments with different properties. For example, an environment with more obstacles or a more complex layout may require a different heuristic than the one used in this study.

Despite these limitations, this study establishes a framework for ongoing enhancement efforts in the field of Online Depth First Search. The vision of elevating algorithm efficiency through informed successor prioritization still shows promise, if proper adaptations are made. By learning from this study's shortcomings and honing in on more compatible forms of heuristic guidance, future work can build off these beginnings and get closer to efficiently navigating the intricate unknown.

8 Contributions

8.1 Ketan

- Methods
- Wrote Online_DFS function
- Wrote Directed_Online_DFS function
- Came up with project idea
- Did majority of coding

8.2 Ronit

- Abstract
- Introduction
- Background Research, Related Work
- Conclusion
- References
- Formatting, Latex generation

8.3 Chiemeka

- Results
- Analysis
- Wrote function to generate random mazes function
- Benchmarked Algorithms using timestamps and action amounts
- Wrote evaluation function to evaluate the performance of the two algorithms

- Formed trials with N number of simulations
- Methods
- Wrote check for maze solvability

References

[1] Nawaf Hazim Barnouti, Sinan Sameer Mahmood Al-Dabbagh, and Mustafa Abdul Sahib Naser. 2016. Pathfinding in strategy games and maze solving using a search algorithm. *SCIRP* (Sep 2016). https://www.scirp.org/journal/paperinformation.aspx?paperid=70460

[2] Xuyong Chen, Zhifeng Xu, Yushun Wu, and Qiaoyun Wu. 2023. Heuristic algorithms for reliability estimation based on breadth-first search of a grid tree. *Reliability Engineering System Safety* 232 (2023), 109083. https://doi.org/10.1016/j.ress.2022.109083

[3] Ying-Hsuan Chen and Chang-Ming Wu. 2020. An Improved Algorithm for Searching Maze Based on Depth-First Search. In *2020 IEEE International Conference on Consumer Electronics - Taiwan (ICCE-Taiwan)*. 1–2. https://doi.org/10.1109/ICCE-Taiwan49838.2020.9258170

[4] Anthony Dowling. 2022. Pathfinding in Random Partially Observable Environments with Vision-Informed Deep Reinforcement Learning. *arXiv preprint arXiv:2209.04801* (2022). https://doi.org/10.48550/arXiv.2209.04801 arXiv:2209.04801 [cs.LG]

[5] Magnus Engström. 2019. Building an AI to navigate a maze. *Medium* (mar 2019). https://magnus-engstrom.medium.com/building-an-ai-to-navigate-a-maze-899bf03f224d

[6] Gopi Gugan and Anwar Haque. 2023. Path Planning for Autonomous Drones: Challenges and Future Directions. *Drones* 7, 3 (2023). https://doi.org/10.3390/drones7030169

[7] Silvester Dian Handy Permana, Ketut Bayu Yogha Bintoro, Budi Arifitama, and Ade Syahputra. 2018. Comparative Analysis of Pathfinding Algorithms A *, Dijkstra, and BFS on Maze Runner Game. https://api.semanticscholar.org/CorpusID:55101893

[8] Abdul Rafiq et al. 2020. Pathfinding Algorithms in Game Development. *IOP Conf. Ser.: Mater. Sci. Eng.* 769 (2020), 012021. https://doi.org/10.1088/1757-899X/769/1/012021

[9] M.N. Sagming, R. Heymann, and E. Hurwitz. 2019. Visualising and Solving a Maze Using an Artificial Intelligence Technique. In *2019 IEEE AFRICON*. 1–7. https://doi.org/10.1109/AFRICON46755.2019.9134044

[10] Peter Norvig Stuart Russell. 2020. *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson. http://aima.cs.berkeley.edu/