COS30019 Introduction to AI Assignment - 1

Robot Navigation Problem

Syed Omair Maqdoom Mohiuddin
102863768

Table of Contents

Instructions

This program is developed in C++, a sophisticated general-purpose programming language. It is commonly used to develop operating systems, browsers, games, and a variety of applications. Before running a C++ program, it needs to be compiled. The executable file can be run immediately with a double-click, or it can be called from another application or run from the command line (terminal). This project is a command line application that takes two arguments: A Source File and The Algorithm to be Used.

To compile a C++ program, we use the following syntax: **g++ file_name.cpp**
If there are multiple independent files that needs to be compiled together, we use following syntax:
g++ file_1 file_2 file_3
By default, an executable file called a.out is created. The following syntax is used to modify the name of the compiled file: **g++ file.cpp -o output_file** where output_file is the compiled executable file.

To compile and run the program use the below syntax:

Compile: g++ Main.cpp gridDisplay.cpp filehandler.cpp node.cpp -o output

Run: ./output <Input_File> <Algorithm>
Where Algorithm can be either one of these [DFS, BFS, GBFS, AS, CUS1 and CUS2]
For Instance: ./output TestFile.txt GBFS

Introduction

An intelligent autonomous robot is required in a range of applications, including space, transportation, industry, and the military. Mobile robots may perform tasks such as material handling, disaster relief, patrolling, and rescue missions. As a result, an autonomous robot capable of freely navigating in both static and dynamic environments is required. The primary purpose of mobile robot navigation is to navigate a mobile robot efficiently and securely from initial position to destination in a cluttered environment, while maintaining a safe path and delivering the lowest path length possible. Researchers have looked at a variety of ways for robot navigation path planning in this area.

Physical duties that were initially believed to be conducted by humans are now performed by machines to decrease the burden of labor on individuals in the modern day. However, machines must not only do physical tasks, but also think and make judgments in the same manner that people do. Artificial intelligence knowledge will play a key part in making things intelligent and reaching this aim. Path planning, in which robots must go from a starting point to a destination point while avoiding obstacles, is the most common difficulty in robot navigation.

The built agent(robot) has the necessary functionality to navigate around the grid, keep track of its state and identify which pathway it is capable of taking, as well as whether it is in a goal state. Uninformed and Informed are search algorithms developed for this robot. Depth-first, Breadth-first and Dijkstra search methods are employed for uninformed. For Informed, Greedy best-first, A* and modified Dijkstra search methods are implanted.

Search Algorithms

i. Depth-First Search (DFS)

The depth-first search (DFS) technique is used to traverse or explore data structures such as trees and graphs (Rodrigues et al., 2019). The algorithm starts from the root node (in the case of a graph, any random node can be used as the root node) and examines each branch as far as feasible before retracting (Rodrigues et al., 2019). In DFS, we will expand the deepest unexpanded node. Last In First Out (LIFO) queue, i.e., successor will come first, is the frontier in this search. It is complete in finite spaces. Although the search in not optimal and incomplete in infinite-depth spaces, spaces with loops. Below figure shows the successful implementation of DFS.

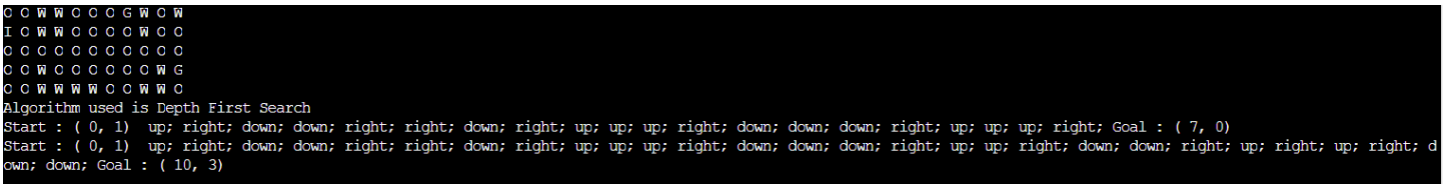

Figure 1: Successful implementation of DFS method.

ii. Breadth-First Search (DFS)

A graph's Breadth-first search (or traversal) is similar to a tree's Breadth-First Traversal (Zhang et at., 2018). The only difference is that, unlike trees, graphs can contain cycles, which means we might end up at the same node multiple times (Zhang et al., 2018). An array of Boolean is used to mark visited nodes to avoid processing a node multiple time. For the sake of simplicity, all vertices are assumed to be reachable from the starting vertex (Zhang et al., 2018). In BFS, we will expand the shallowest unexpanded node. Here frontier is a First In First Out (FIFO) queue, i.e., new successor will go at the end. It is complete and optimal in finite conditions. Although, space is a problem instead of time for this search method. Below figure shows the successful implementation of BFS.

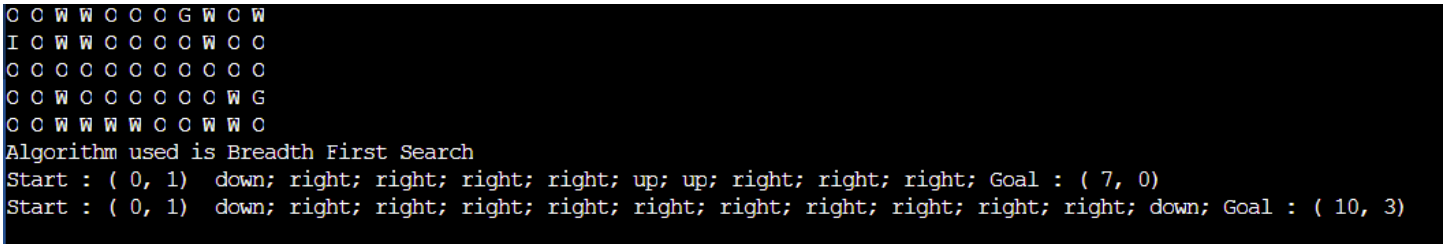

Figure 2: Successful implementation of BFS method.

iii. Greedy Best-First Search (GBFS)

The greedy best-first search (GBFS) algorithm always chooses the path that appears to be the most appealing at the time (Fickert, 2020). It's the result of combining depth-first and breadth-first search algorithms (Fickert, 2020). It makes use of the heuristic function as well as search. Best-first search combines the benefits of both methods. This search is incomplete, not optimal and in worst case scenario it becomes similar to depth first search. Below figure shows the successful implementation of GBFS.

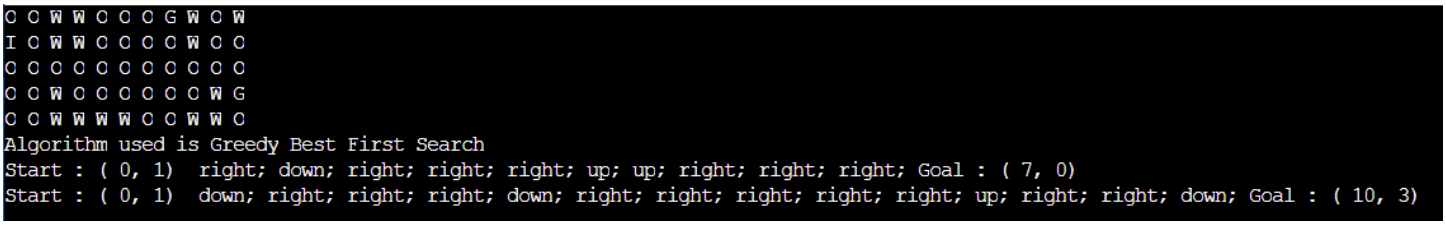

Figure 3: Successful implementation of GBFS method

iv. A Search (AS)*

The A* search method is one of the most widely used pathfinding and graph traversal techniques (Wang and Chen, 2018). Informally, unlike other traversal approaches, A* search algorithm have "brains", i.e., it is a sophisticated algorithm that distinguishes itself from other algorithms (Wang and Chen, 2018). It is best-known of best-first search. It is an important AI algorithm developed by Fikes and Nilsson in early 70s. It was originally used in Shakey robot. The basic working of this algorithm is

it avoids expanding paths that are already expensive. A* search uses an admissible heuristic (never overestimates the cost to reach the goal). Is complete (unless there are infinity many nodes) and optimal in most of the conditions.

```
C C W W C C C G W C W
I C W W C C C C W C C
C C C C C C C C C C C
C C W C C C C C C W G
C C W W W W C C W W C
Algorithm used is A* Search
Start : ( 0, 1)  right; down; right; right; right; up; up; right; right; right; Goal : ( 7, 0)
Start : ( 0, 1)  down; right; right; right; right; right; right; right; right; right; right; down; Goal : ( 10, 3)
```

Figure 4: Successful implementation of A method*

v. *Dijkstra's Algorithm (CUS1)*

The weights of the edges are used by Dijkstra's method to discover the path that minimizes the overall distance (weight) between the source node and all other nodes (Parungao et al., 2018). The single-source shortest route algorithm is another name for this approach. It finds the shortest path between a given node (source node) and goal node. It uses the weight/cost for each node to find the best path that minimizes the total distance (weight/cost) between the source node and goal node. As it does a blind search, it consumes a lot of time.

```
C C W W C C C G W C W
I C W W C C C C W C C
C C C C C C C C C C C
C C W C C C C C C W G
C C W W W W C C W W C
Algorithm used is Dijkstra's Algorithm
Start : ( 0, 1)  down; right; right; right; right; up; up; right; right; right; Goal : ( 7, 0)
Start : ( 0, 1)  down; right; right; right; right; right; right; right; right; right; right; down; Goal : ( 10, 3)
```

Figure 5: Successful implementation of first custom algorithm

vi. Dijkstra's Algorithm (CUS2): Visit all Goals with shortest path

By improving the Dijkstra's algorithm, it is now capable to search for multiple goals with shortest path. This can also be referred as 'Smart Dijkstra's Algorithm'. It starts the search with searching for the first nearest goal and then followed by searching next nearest goals.

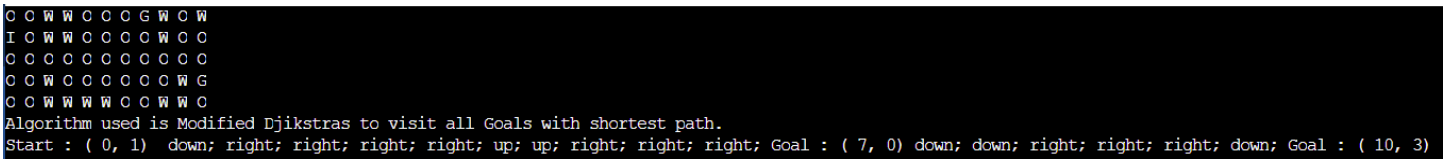

Figure 6: Successful implementation of second custom algorithm

Implementation

The program contains various program files, classes, and objects which jointly work to achieve the intended goal. All the files are discussed below.

Main

- **Main.cpp**

  This is the main file in the program which acts as the entry point into the program. It includes all the header files contained in the program and implements all algorithms mentioned earlier.

Source Code Files

- **node.cpp**

  This file contains implementation of different algorithms as declared in the gridDisplay.h file.

- **filehandler.cpp**

This file includes the file handler header file and implements the ParseLine method declared in the header file. In a nutshell, the program implements all methods needed to handle the file parsed when the program is run on the terminal.

- **gridDisplay.cpp**

  The file contains implementation of different algorithms as declared in the gridDisplay.h file.

Header Files

- **node.h**

  This file contains the declarations of codes used in the node.cpp and other source files.

- **filehandler.h**

  This file contains declaration of the Vector and GridClass objects

- **gridDisplay.cpp**

  The file contains declaration for all algorithm methods and Nodes.

Features/Bugs/Missing

The application has no graphical user interface and may only be used through the terminal, or command prompt, under windows operating systems. As C++ is a cross-platform programming language, the software may be compiled and run on windows, Mac, and Unix systems. Using a suitable operating system command line, we can compile and run the program. The require feature, which is finding shortest route a robot can follow on a grid is implemented successfully.

A visual showing the grid based on the input file is also developed to help the user understand the application effectively. Where 'R' represents the initial state of robot (agent), 'G' as the goal state. 'W' as blocked grids which cannot be bypassed and 'O' as normal grids which the robot can use for its movement.

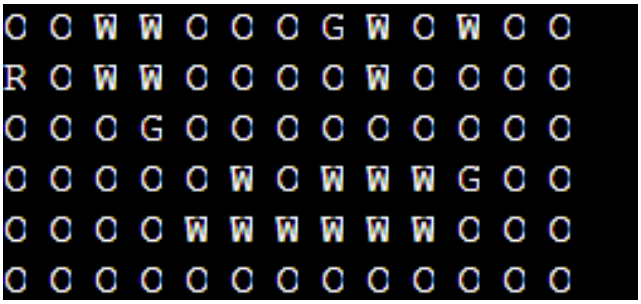

Figure 7: Grid Visual

If the user inputs a file which isn't available in/provided to the program, then it gives the error saying that the program couldn't load the given file.

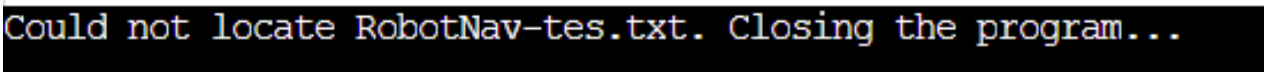

Figure 8: Error handling for input file argument

If the input file doesn't contain the initial state and destination state of the agent, then the program prompts the user saying the initial/goal state is missing. Moreover, if the given initial/goal state is missing from the given combination of the grid, the program gives an error saying the initial/goal state is missing.

Conclusion

A* algorithm would be a best strategy for this grid problem and in identical circumstances, as this is a basic path finding solution with no dynamic or conflicting entities (competitors). Although, A* is the best comparing to other developed algorithms, there might be another good algorithm which will be more effective and efficient than A*.

References:

Fickert, M. (2020, June). A novel lookahead strategy for delete relaxation heuristics in greedy best-first search. In *Proceedings of the International Conference on Automated Planning and Scheduling* (Vol. 30, pp. 119-123). https://ojs.aaai.org/index.php/ICAPS/article/view/6652

The conference paper aided me in understanding the functioning and implementation of greedy algorithms. With the help of the resource, I gained much more reliable understanding on the greedy-best first search algorithm.

Parungao, L., Hein, F., & Lim, W. (2018). Dijkstra algorithm based intelligent path planning with topological map and wireless communication. *ARPN Journal of Engineering and Applied Sciences*, *13*(8), 2753-2763. http://www.arpnjournals.org/jeas/research_papers/rp_2018/jeas_0418_6995.pdf

While the reference is based on designing wireless networks, it provided me with the general knowledge needed in implementing artificial intelligence algorithms, and consequently, the success of the project.

Quoc Vo, B. (2022), Problem solving and Search, Introduction to AI, Learning material via canvas, Swinburne University of Technology.

Lecture material provided a brief understanding of various algorithms and the robot navigation problem.

Wang, J., & Chen, H. (2018). BSAS: Beetle swarm antennae search algorithm for optimization problems. *arXiv preprint arXiv:1807.10470*. https://doi.org/10.48550/arXiv.1807.10470

The resource focuses on fine-tuning search algorithms for optimization of basic computing problems. I employed the skills learned there in to design and implement the project.

Zhang, F., Lin, H., Zhai, J., Cheng, J., Xiang, D., Li, J., ... & Du, X. (2018). An adaptive breadth-first search algorithm on integrated architectures. *the Journal of Supercomputing*, *74*(11), 6135-6155. https://doi.org/10.1007/s11227-018-2525-0

I relied on this resource to advance my skills and understanding of breadth-first search algorithms for different architectures. I later adopted the skills in designing and developing the project.