**Robot Navigation Problem – Search Algorithms**

Author: Jared Teng Qi Peng

Student Number: 103492121

Date Created: 22nd April 2022

**Table of Contents**

1. Introduction
   1. Glossary
2. Search Algorithms
   1. Instructions
   2. Breadth-first Search
   3. Depth-first Search
   4. Greedy-best-first Search
   5. A\* Search
   6. Depth-limited Search
   7. Dijkstra
   8. Bugs and missing features
3. Results
4. Discussion
   1. Uninformed Search
   2. Informed Search
5. Conclusion
6. Acknowledgments & Resources
7. References

**Introduction**

The Robot navigation is a search problem where multiple search algorithms is applied to a grid. The problem will investigate two categories of pathfinding algorithms, uninformed and informed search algorithms. A pathfinding algorithm serves the purpose of finding the shortest path possible from a given initial state to the goal state.

This investigation aims to understand and pick out the best pathfinding algorithms and to understand the application of each algorithm.

**Glossary**

|  |  |
| --- | --- |
| Word | Definition |
| Cell | A cell or node with a position of (x,y) in a 2D grid |
| OpenSet | A sequence of cells that have been visited but unexplored |
| ClosedSet | A sequence of cells that have been explored |
| Back Tracking | A technique used to backtrack to a node where alternative paths are possible |
| Initial State | The agent’s initial position in an environment |
| Goal State | The state in which the agent wishes to solve |
| Tree | A hierarchal tree structure that has nodes connected to each other |

**Search Algorithms**

**Instructions**

The system was created using the C# language complemented with Splashkit toolkit for its graphical user interface and is created using Visual Studio Code. The system consists of a terminal window for user input and a GUI grid window for pathfinding visualization. Using a MSYS2 MinGW terminal window, execution of program is done by the following command on their file location

skm dotnet run

Alternatively, the program can be executed using Visual Studio 2019. Moreover, a batch file has been built and can be run in windows command line prompt. Below are the instructions to be entered:

1. search Data.txt (method name)
2. List of methods that can be entered
   1. BFS
   2. DFS
   3. GBFS
   4. A\*
   5. DLS
   6. DJKS
3. A graphical interface will pop up and the search will commence
4. Repeat step 1 to execute other methods

**Class diagram**

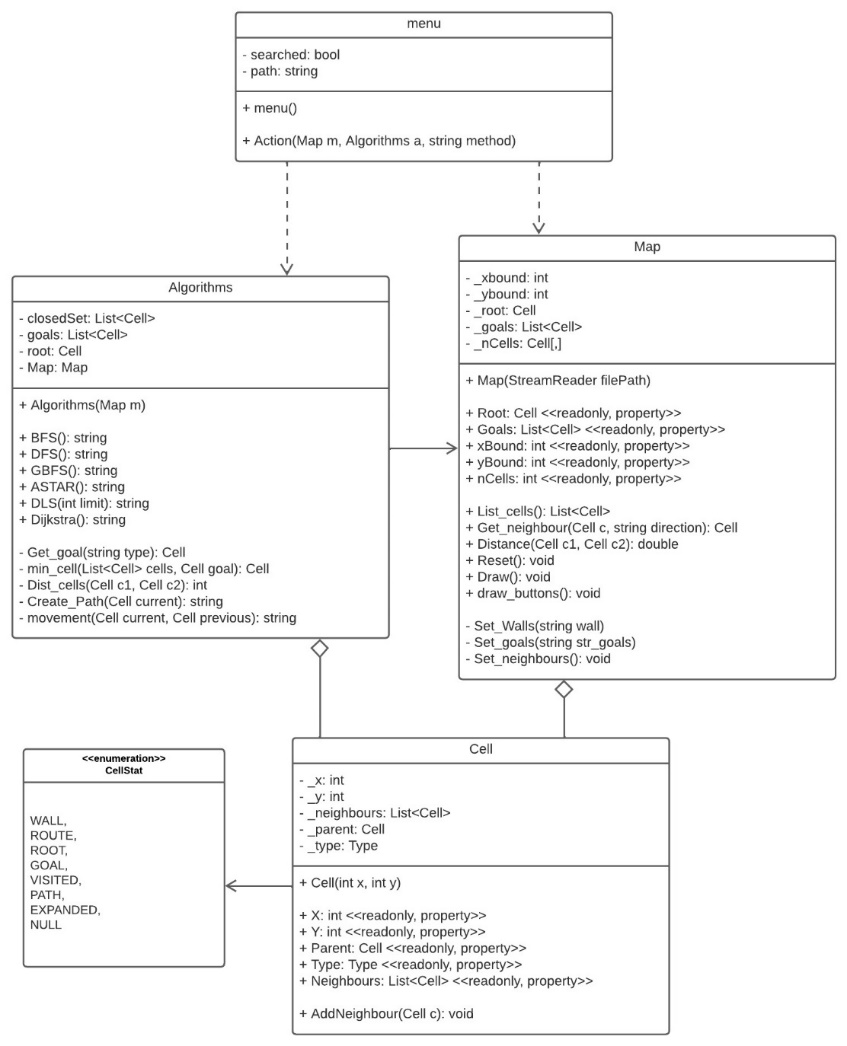


Figure 1: class diagram to Pathfinding Program

Search algorithms are implemented in one class called Algorithms and is being used by class menu to execute methods. The class map handles methods related to its environment such as its cells and information of the cells. This class is also in charge of reading the file, setting up the environment and drawing grid on the window for visualisation. The statuses of cells are represented by the data type enumeration in class CellStat. This would be later used to update the cells for visualisation.

**Map Grid**

Search algorithm will be applied on a 11x5 (x,y) grid consisting of the following and their respective description and colour

* Root (initial state) GREEN
* Goal (multiple goal states) RED
* Path (walkable states) WHITE
* Walls (unwalkable states) BLACK

A screenshot of a computer

Description automatically generated with low confidence

Figure 2: Map grid with dimensions 11x5

**Rules**

1. Order of expansion: UP, LEFT, DOWN, RIGHT

**Breadth-first Search (BFS)**

The algorithm uses a first-in-first-out queue. Both breadth-first search and Depth-first search uses similar structures. However, breadth-first search searches all nodes each level of the search tree. Below is a pseudocode:

Graphical user interface, text, application

Description automatically generated

Figure 3: breadth first search pseudocode

In the case of the program, cells are marked as explored if they are contained in the closed set. The openSet are cells that are yet to be explored but have been visited and is placed in a queue.

**Depth-first Search (DFS)**

This algorithm uses a last-in-first-out queue. This is done by using a stack data type rather than a queue like BFS. The order in which neighbouring cells are pushed in the stack are UP, LEFT, DOWN, RIGHT. However, by doing so, the first cell to be expanded will be the cell in the RIGHT. By pushing the cells in the opposite side RIGHT, DOWN, LEFT, UP, we make sure that the order of expansion satisfies rule. This is also implemented by depth-limited search as it is the same as depth first search with the exception of a limiter. Below is a pseudocode:

Text

Description automatically generated

Figure 4: depth first search pseudocode

**Greedy-best-first Search (GBFS) & A\* Search**

Both algorithms use the best-first search algorithm but with different evaluation functions. The heuristic of GBFS is h(n) where it expands the nodes with the lowest h(n). In contrast, A\* has the heuristic of g(n) + h(n) (h(n) is admissible). Because informed search requires a goal state to be known, a function called Get\_goal(string type) is created to get the closest goal or furthest goal by typing “MIN” or “MAX”.

Text

Description automatically generated

Figure 5: Greedy best first search pseudocode

Graphical user interface, text, application

Description automatically generated

Figure 6: A\* pseudocode

**Depth-limited Search (DLS)**

This is an uninformed search algorithm. It is like DFS except for a limiter. Given a depth limit, the search will iterate through a search and will terminate when it reaches the limit. The addition of a limiter solves the problem of infinite loops caused by DFS [1].

Text

Description automatically generated

Figure 7: Depth-limited search pseudocode

**Dijkstra’s Search (DKJS)**

The algorithm was implemented using the pseudocode from Wikipedia [2]. Instead of using a list prev[] to trace back the shortest path, the path is created by assigning a parent to a node and tracing back from the goal node to the root node. Moreover, a function called, Dist\_cells(c1,c2), is uses Manhattan distance to calculate distance between neighbouring cells. This is used to calculate alt in the pseudocode.

The array dist[v] is implemented using a Dictionary<Cell, int>, where Cell is a key and the integer is value. Infinity values for these distances are represented as int.MaxValue.

Text

Description automatically generated

Figure 8: Dijkstra’s pseudocode [2]

**List of Bugs and missing features**

1. DFS: Cell at position (3,3) has a slight delay. This is also present in DLS.
2. Feature: No tree structure present and its changes.
3. DJKS: Order of expansion does not follow the order UP, LEFT, DOWN, RIGHT.

**Results**

**Measurements**

Two variables are used to compare each algorithm.

1. Length of path (Number of Cells, excluding initial and goal states)
2. Number of expanded nodes

Figure 9: BFS visualisation

A screenshot of a computer

Description automatically generated with low confidence

Figure 10: DFS visualisation

A screenshot of a computer

Description automatically generated with low confidence

Figure 11: GBFS visualisation

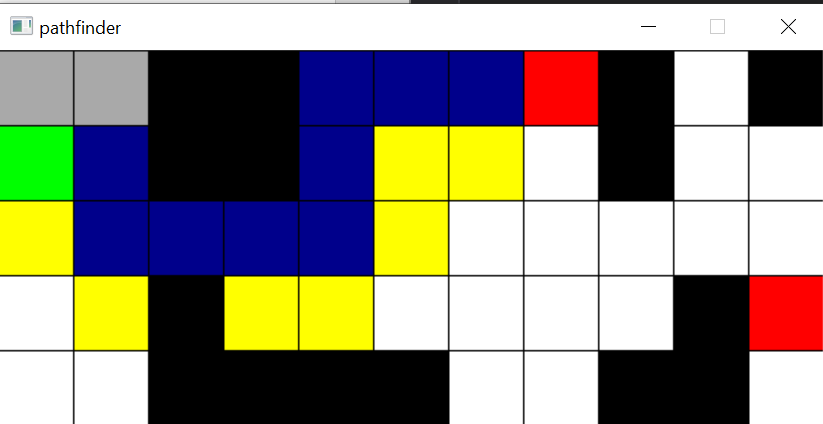


Figure 12: A\* visualisation

A picture containing text, crossword puzzle

Description automatically generated

Figure 13: DLS visualisation

A screenshot of a computer

Description automatically generated with low confidence

Figure 14: DJKS visualisation

A screenshot of a computer

Description automatically generated with low confidence

Table 1: Comparison between different Search Algorithms

|  |  |  |
| --- | --- | --- |
|  | Path Length (No. Cells) | No. Expanded Cells |
| BFS | 9 | 33 |
| DFS | 19 | 28 |
| GBFS | 9 | 13 |
| A\* | 9 | 13 |
| DLS (limit 28) | 19 | 28 |
| DJKS | 9 | 31 |

**Discussion**

**Uninformed Search (BFS, DFS, DLS)**

BFS has the shortest path of 9 cells in contrast to DLS and DFS with path length of 19. However, BFS requires more memory as it expands 33 cells in contrast to DFS and DLS with 28. DFS only requires following a single thread from top to bottom in a tree structure. BFS on the other hand, expands every node each level of the search tree which requires more space.

DFS and DLS are equivalent given that the depth limit given to DLS is enough to reach the closest goal in Cell (8,1). Further cases need to be implemented in a case where DFS reaches an infinite loop. By doing so, an understanding of the impact of DLS can be understood [1] as it allows the loop to reach an end when a depth of the search tree is equivalent to the limit set. Furthermore, DLS and DFS has the same number of expanded nodes. This means that DLS does not affect the space requirement of the algorithm.

As all cells have a connection if and only if a cell exists and is not a wall, DFS and DLS does not have an optimal path. Starting from Cell (4,3), the path starts to create a path, given a cell exist to walk. This creates a long path as compared to BFS. This means that DFS and DLS may not work well in large environment. Further research can be conducted in the case where these algorithms are tested in a compact environment.

**Informed Search (GBFS, A\*, DJKS)**

GBFS and A\* are equivalent in path length and number of expanded nodes. This may be the case as the distance between each cell is equivalent and that both algorithms adopt the same structure of best-first search, apart from different evaluation functions.

As for DJKS, path length is equivalent to GBFS and A\*. However, space requirement is an issue as it expands 31 cells at end of the search. According to research in visualizing search algorithms in a grid [3], DJKS and BFS expands all nodes in the same pattern given that order of expansion is the same and no diagonal movement are allowed. There shouldn’t be any differences between BFS and DJKS in their expanded nodes. This may be due to DJKS having not expanded in the right order which was not implemented in this study.

Further research is needed to compare all three algorithms. An alternative is to create an environment where all cells do not have the same distance between each other.

Out of all the algorithms analysed, all algorithms found the path with 9 as their length except for DFS and DLS. Furthermore, GBFS and A\* have both found the optimal path but have also found to have needed less space requirements than others. This means that an informed heuristic search is the best choice for this problem. However, further research is needed to understand cases in which the GBFS and A\* are the worst choice.

**Conclusion**

Without the use of search algorithms, a person must search through data one by one or use a map to find their way. The use of search algorithms solves these problems by reducing the time and increasing the accuracy of the search given the appropriate search algorithm. From the study of 6 search algorithms, an informed heuristic search algorithm, GBFS and A\*, is not only optimal but also efficient in memory requirements for this specific problem. The worst algorithm found for this problem, are algorithms that either search through a single path (DLS and DFS) or expands every level of a tree (BFS and DJKS) which leads to large space requirements. Improvements can be made to the implementation of these algorithms and the quality of the research. First, allowing different environments to tested will allow for a better conclusion to the chosen algorithm. Second, efficiency can be made for each algorithm. The expansion of nodes (e.g., line 9-11 BFS pseudocode) uses a foreach loop. This can be replaced by calculating the location (x,y) of the current node to get the neighbouring cells rather than creating memory for accessing it. This can be done 4 times for each neighbours (e.g., current cell (x,y), north neighbouring cell (x-1,y)). This implementation may not work for some informed searches but works well for uninformed searches as they do not need an evaluation function.

**Acknowledgements & Resources**

1. Splashkit toolki (Splashkit.io)
2. “Breadth-first search,” *Wikipedia*, 14-Apr-2022. [Online]. Available: https://en.wikipedia.org/wiki/Breadth-first\_search. [Accessed: 22-Apr-2022].
3. <https://github.com/AtsushiSakai/PythonRobotics/tree/master/PathPlanning/DStar>
4. <https://www.youtube.com/watch?v=CLbqqb53DLA>
5. <https://www.geeksforgeeks.org/difference-between-bfs-and-dfs/#:~:text=BFS(Breadth%20First%20Search)%20uses,edges%20from%20a%20source%20vertex>.
6. <https://www.geeksforgeeks.org/best-first-search-informed-search/>

**References**

1. E. Forbes, “Depth limited search in Java,” *TutorialEdge*. [Online]. Available: https://tutorialedge.net/artificial-intelligence/depth-limited-search-in-java/. [Accessed: 19-Apr-2022].
2. “Dijkstra's algorithm,” *Wikipedia*, 18-Apr-2022. [Online]. Available: https://en.wikipedia.org/wiki/Dijkstra%27s\_algorithm. [Accessed: 22-Apr-2022].
3. N. Kanargias, *An attempt to visualize five search algorithms*, pp. 10–11, Nov. 2013.