

Swinburne University of Technology

The Robot Navigation Problem

COS30019 - Introduction to AI

Assignment 1 – Tree Based Search

Student ID: 103579765

Name: Hoang Nhat Quang

Date: 26/06/2022

Table of Contents

Instructions	3
Introduction	4
Search Algorithms.....	4
Depth-first search	4
Breadth-first search	5
Greedy best-first search	5
A*	6
Uniform-cost search	6
Iterative deepening A*	7
Implementation	7
Create map	7
Create solution	7
Depth-first search	7
Breadth-first search	8
Greedy best-first search	8
A*	8
Uniform-cost search	8
Iterative deepening A*	8
Limitations.....	8
Research	9
Conclusion	9
Acknowledgements.....	9
References	10

Instructions

To run the program well, here are some important guidelines as follows:

Set up working environment:

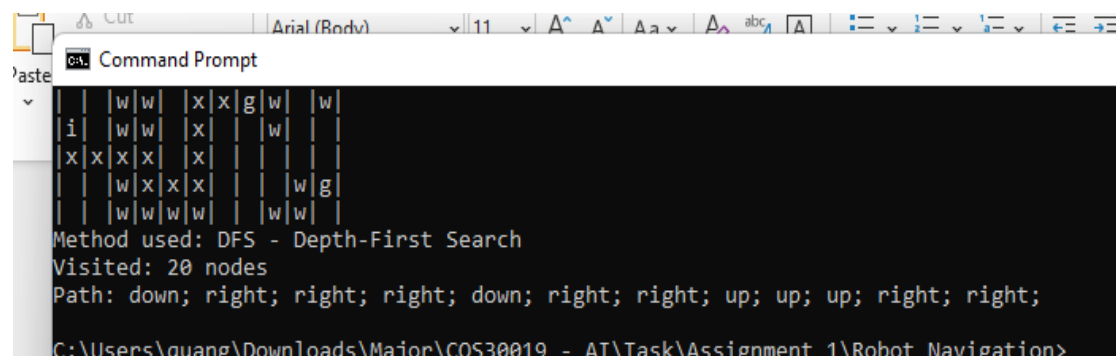
- +) Window 10, Windows 11 also works fine
- +) Installed Python 3.9.7
- +) Library Used: sys, os, random, time, math
- +) Correct input file format. Please check file path in main.py line 6 if change test input file

User Guide:

- +) Correct file path
- +) Type in cmd: `py main.py <method>`
- +) There are 7 methods available: DFS, BFS, GBFS, AS, CUS1, CUS2, CUS3
- +) Example:

```
ds\Major\COS30019 - AI\Task\Assignment 1\Robot_Navigation>py main.py DFS
```

- +) There will be a command-line GUI showing the program's search progress. Then the results will be displayed.



```

| | w w | x x | g w | w | | |
| i | w w | x | | w | |
| x | x | x | x | | | |
| | w x | x | x | | w | g |
| | w w | w | w | | w | w |
Method used: DFS - Depth-First Search
Visited: 20 nodes
Path: down; right; right; right; down; right; right; up; up; up; right; right;
C:\Users\quang\Downloads\Major\COS30019 - AI\Task\Assignment 1\Robot Navigation>

```

Read Source Code Guide:

- +) Correct file path
- +) “/Project” contains all source code
- +) The part “extension” is not the original problem, it is for the research part, please check the research section below for details.

```
# EXTENSION
def VisitAllGoalShortest(self):
```

Introduction

In this assignment, I create a Robot Navigation program. The program finds a path from an initialize point to the goal point in an environment which designed in NxM (grid) format and has walls. In case has multiple-goal points, the program will find a path to one of them. There were 6 search algorithms that use to find the path include: depth-first search, breadth-first search, greedy best-first, A*, uniform-cost search, and iterative deepening A*. Each of them has its advantages and disadvantages which will be mentioned in this paper. I also demonstrate how I implement these algorithms in the Python programming language. Some missing could be improved in the future will also be mentioned. Moreover, I present some ideas to make the program more complicated, more efficient

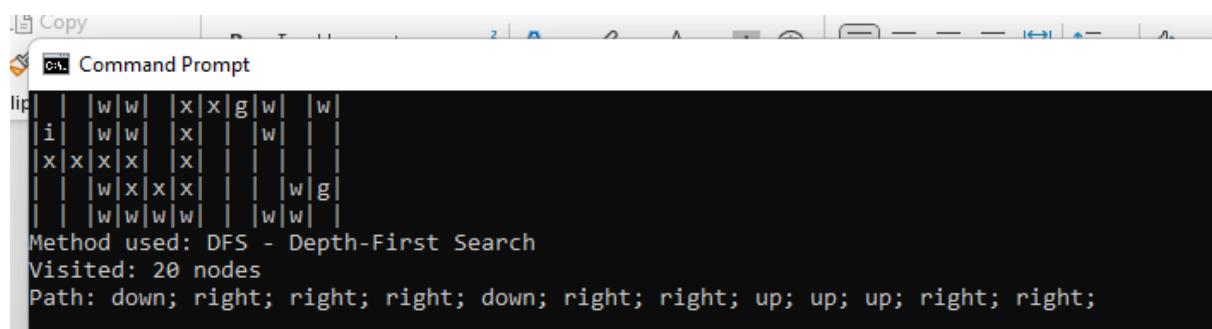
Search Algorithms

Depth-first search

How to run:

```
ds\Major\COS30019 - AI\Task\Assignment 1\Robot_Navigation>py main.py DFS
```

Results:



```
lip | | w | w | | x | x | g | w | | w |
    | i | | w | w | | x | | | w | |
    | x | x | x | x | | x | | | |
    | | | w | x | x | x | | | w | g |
    | | | w | w | w | w | | w | w |
Method used: DFS - Depth-First Search
Visited: 20 nodes
Path: down; right; right; right; down; right; right; up; up; up; right; right;
```

Discussion:

Depth-first search is an uninformed search. It uses last-in-first-out strategy (a stack) to implement. Start at the root node and explore as far as possible along each branch before backtracking. [1]

d = the depth of the search tree

n^i = number of nodes in level i

Time complexity: $O(n^d)$

Space complexity: $O(n \times d)$

Completeness: Yes, if the solution exists

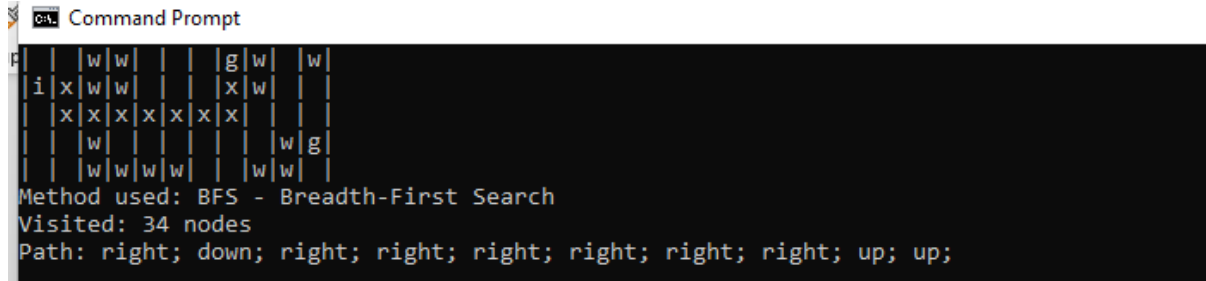
Optimality: No, cost spent in reaching it is high

Breadth-first search

How to run:

```
ds\Major\COS30019 - AI\Task\Assignment 1\Robot_Navigation>py main.py BFS
```

Results:



```

i | x | w | w |   |   | g | w |   | w |
| i | x | w | w |   |   | x | w |   |   |
| | x | x | x | x | x | x |   |   |   |
| |   | w |   |   |   |   | w | g |   |
| |   | w | w | w | w |   | w | w |   |
Method used: BFS - Breadth-First Search
Visited: 34 nodes
Path: right; down; right; right; right; right; right; right; up; up;

```

Discussion:

Breadth-first search is an uninformed search. It uses first-in-first-out strategy (a queue) to implement. Start at the root node and explore all of the neighbor nodes at the present depth level after that moving to the nodes at the next depth level. [1]

s = the depth of the shallowest solution

n^i = number of nodes in level i

Time complexity: $O(n^s)$

Space complexity: $O(n^s)$

Completeness: Yes, if the solution exists

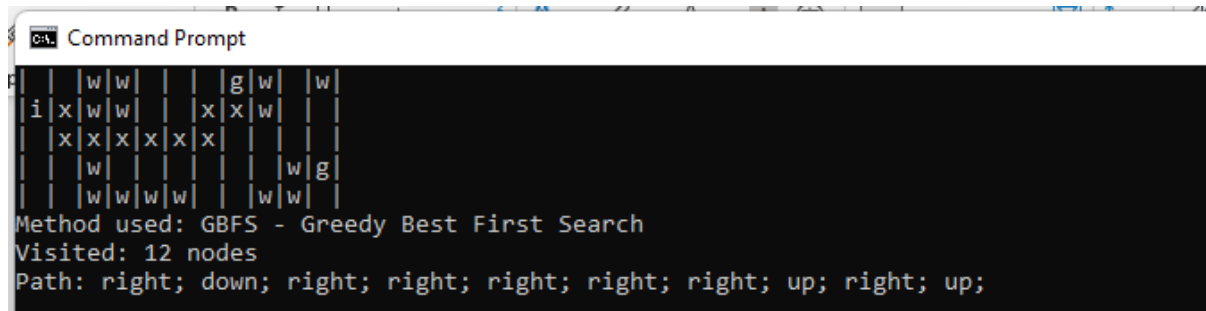
Optimality: Yes

Greedy best-first search

How to run:

```
ds\Major\COS30019 - AI\Task\Assignment 1\Robot_Navigation>py main.py GBFS
```

Results:



```

i | x | w | w |   |   | g | w |   | w |
| i | x | w | w |   |   | x | w |   |   |
| | x | x | x | x | x |   |   |   |   |
| |   | w |   |   |   |   | w | g |   |
| |   | w | w | w | w |   | w | w |   |
Method used: GBFS - Greedy Best First Search
Visited: 12 nodes
Path: right; down; right; right; right; right; right; up; right; up;

```

Discussion:

Greedy best-first search is an informed search. It expands the node closest to the goal node by calculating heuristic $h(x)$ value. Lower the value of $h(x)$, closer is the node from the goal. [1]

Advantage: With fewer steps to reach a goal.

Disadvantage: Can turn into unguided DFS in the worst case.

A*

How to run:

```
s\Major\COS30019 - AI\Task\Assignment 1\Robot_Navigation>py main.py AS
```

Results:

```

Method used: AS - A* Search
Visited: 23 nodes
Path: right; down; right; right; right; right; right; up; right; up;

```

Discussion:

A* search is an informed search. It combines the strengths of uniform-cost search and greedy search. It calculates $f(n) = g(n) + h(n)$ where $g(n)$ is the summation of the cost in UCS and $h(n)$ is the cost in the greedy search. The algorithm chooses the node with the lowest $f(x)$ value. [1]

Advantage: Shortest path to goal

Disadvantage: It may take time re-exploring the branches it has already explored.

Uniform-cost search

How to run:

```
s\Major\COS30019 - AI\Task\Assignment 1\Robot_Navigation>py main.py CUS1
```

Results:

```

Method used: CUS1 - Uniform Cost Search
The cost between two points is a random number in the range 1-5. So the result will be different every time the program
run
If the cost is the same the result will be the same as BFS
Visited: 32 nodes
Path: right; down; right; right; right; right; right; up; right; up;

```

Discussion:

Uniform-cost search is an uninformed search. The goal is to find a path where the cumulative sum of costs is the least. [1]

Advantages: UCS is complete only if states are finite and there should be no loop with zero weight.

Disadvantages: Explores options in every direction.

Iterative deepening A*

How to run:

```
Downloads\Major\COS30019 - AI\Task\Assignment 1\Robot_Navigation>py main.py CUS2_
```

Results:

```

C:\Users\quang\Downloads\Major\COS30019 - AI\Task\Assignment 1\Robot Navigation>py main.py CUS2_
Method used: CUS2 - Iterative Deepening A*
Visited: 21 nodes
Path: right; down; right; right; right; right; right; up; right; up;
C:\Users\quang\Downloads\Major\COS30019 - AI\Task\Assignment 1\Robot Navigation>

```

Discussion:

The Iterative Deepening A* algorithm is an informed search. It builds on Iterative Deepening Depth-First Search (ID-DFS) by adding an heuristic to explore only relevant nodes. Iterative Deepening A Star uses a heuristic to determine which nodes to explore and at which depth to stop. [2]

Implementation

Create map

In the beginning, the program will be provided a text input file that contains all the information about the working environment of the robot such as map structure, start point, goal points, and walls. The program will process it into lists of integers for the map and robot initializing section. The map is an NxM grid system with opened and closed blocks (wall). Each valid movement of a block can be represented as point2D for tree search. The robot knew its start point and goal lists.

Create solution

To create a solution, we need a list of visited nodes containing the path to a goal. We get the point which matches the goal in visited nodes, then iterate back parent node until the parent node equals the start node

Depth-first search

The depth-first search could be implemented as follow. Create a stack and a normal list, in the program code my stack is initialized as a list but I make it behave like a stack (last-in-

first-out). Push the start point to the stack, then pop it to evaluate, and also append it to visited nodes. Check all possible movements of the current point and push it to stack, remember that the current point is the parent node of these movements. Keep doing it until getting the goal.

Breadth-first search

The breadth-first search has the same strategies as the depth-first search but instead of using a stack, we use a queue (first-in-first-out).

Greedy best-first search

The greedy best-first search could be implemented as follow. Create two normal lists which are "open" and "visited" list), and append the start point to the "open" list. Sort the "open" list by "distance to goal" in ascending order. Get and remove the lowest value in the "open" list (index 0) for evaluation, and also add it to the "visited" list. Check all possible movements of the current point and push it to the "open" list, remember that the current point is the parent node of these movements. Each movement has its "distance to goal" value by calculating the heuristic value. Keep doing it until getting the goal. Lower the value of heuristic, closer is the node from the goal.

A*

The A* search could be implemented as follow. Create two normal lists which are "open" and "visited" list), and append the start point to the "open" list. Sort the "open" list by "f score" in ascending order. Get and remove the lowest value in the "open" list (index 0) for evaluation, and also add it to the "visited" list. Check all possible movements of the current point and push it to the "open" list, remember that the current point is the parent node of these movements. Each movement has its "f score" value by calculating "f score" = "total cost to get to the current point" + heuristic value. Keep doing it until getting the goal.

Uniform-cost search

The uniform-cost search has the same strategies as the breadth-first search but for each movement, we calculate the total cost to get to the current point. We take the lowest total cost to evaluate. If the cost is the same the result will be the same as BFS. So to make it different with BFS, the cost is a random number in the range of 1-5. So the result will be different every time the program is run.

Iterative deepening A*

Since origin A* can create unnecessary access points, so we try to prevent it from moving too far in iterative deepening A*.

Limitations

Taking the sample test case provided to evaluate, the custom search 2 or iterative deepening A* did not perform as expected. The A* visits 23 nodes to get the goal, iterative deepening A* visits 21 nodes to get the same goal, but it is expected to visit just 19 nodes. So the iterative deepening A* is not fully implemented.

I did not apply pruning methods so the algorithms might take unnecessary steps.

The input file must strictly follow the rules.

Research

- To find the path that connects all goal with the shortest path, we should use the A* algorithm, since A* always provide the shortest path solution. The idea here is to use A* from the start point to get the shortest path to a goal in the goal list, then change the start point to that found goal, and after that remove the found goal from the goal list. Keep looping this process until the goal list is empty.

How to run:

```
s\Major\COS30019 - AI\Task\Assignment 1\Robot_Navigation>py main.py CUS3_
```

Result:

```
Method: Research - VisitAllGoalShortest
Start point to Goal 1: right; down; right; right; right; right; right; up; right; up;
Goal 1 to Goal 2: down; down; right; right; right; down;
Visited: 33 nodes
```

Details:

Find the “extension” part in robot.py

```
442
443     # EXTENSION
444     def VisitAllGoalShortest(self):
445         solution = "Method: Research - VisitAllGoalSh
```

- After running the program correctly, there is a command-line-based user interface that showing the searching process in each event that a new point appends to the “visited” list. Get “visited” list at index “-1” then clearing the console and drawing new status. Details in drawResult.py.

Conclusion

In this paper, I provide briefly about my robot navigation which includes all the algorithms used in the program, how it is implemented, and the answers to the research questions. Completing this robot navigation problem helps me improve my programming skills a lot.

Acknowledgements

This page provides all information about these above algorithms such as how they work, what is its advantages, disadvantages, and sample implementation in some popular programming languages

<https://www.geeksforgeeks.org/search-algorithms-in-ai/>

This page provides all information about the Iterative deepening A* algorithm. Following the guidelines, I apply it into my program as Custom search 2

https://en.wikipedia.org/wiki/Iterative_deepening_A*

References

- [1] GeeksforGeeks (2022) *Search Algorithms in AI*. Available at: <https://www.geeksforgeeks.org/search-algorithms-in-ai/> (Accessed: 26 June 2022).
- [2] Wikipedia (2022) *Iterative deepening A**. Available at: https://en.wikipedia.org/wiki/Iterative_deepening_A* (Accessed: 26 June 2022).