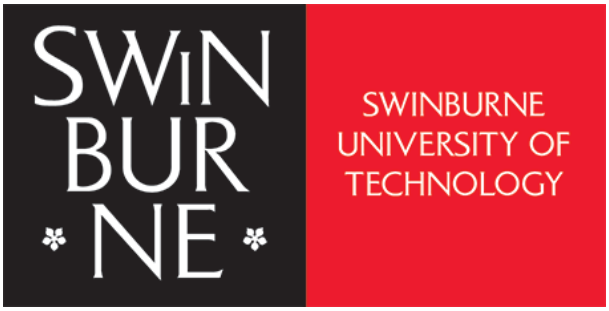
Swinburne University

COS300019 - Introduction to AI

Assignment Report

# Assignment 1 Tree Based Search

*Author:*
Carey McManus
*Student No.:*
7381247



# Contents

# 1 Introduction

# 2 Search Algorithms

## 2.1 Breadth-First Search BFS

There are many ways to choose how you are going to move through a search tree and one of these methods is Breadth-First Search. The main idea of BFS (Breadth-First Search) is that the first node you add into the list of nodes to be searched is also the first node that you expand (FIFO - First In First Out). This means that every node in a layer will be explored before any nodes in the next layer will be expanded.

### Properties

$b = Branching\ Factor$
$d = Depthofleastcostsolution\ Factor$

- Complete : Yes (if branching factor is finite)
- Time Complexity: $O(b^d)$
- Space Complexity: $O(b^d)$
- Optimal: Yes if cost = 1 per step

## 2.2 Depth-First Search DFS

Depth-First Search takes a different approach to BFS by expanding the last node added to the search list first (LIFO - Last In First Out).This has the effect of searching the first node of each layer until there are no more layer before proceeding to the second node of layer 1.

### Properties

$b = Branching\ Factor$
$d = Depthofleastcostsolution\ Factor$
$m = Maximum\ depth\ of\ space$

- Complete : No

- Time Complexity: $O(b^m)$
- Space Complexity: $O(b * m)$
- Optimal: No

## 2.3 Greedy Best First

Greedy Best First is an informed search algorithm meaning that it has some knowledge of the domain that it is searching in, in this case Greedy Best First Search has knowledge of where its goal is. To take advantage of this knowledge when the GBF is selecting the next node to search it does so by choosing the node that is closest to the goal.

### Properties

$b = Branching\ Factor$
$d = Depthofleastcostsolution\ Factor$
$m = Maximum\ depth\ of\ space$

- Complete : No
- Time Complexity: $O(b^m)$
- Space Complexity: $O(b * m)$
- Optimal: No

## 2.4 A*

Greedy Best First has a major problem in the sense that it does not always find the optimal path to the goal. A* addresses this problem by adding a path cost to GBF's heuristic approach.

$$f(n) = h(n) + g(n)$$

By adding the path cost to the function the algorithm will always return the cheapest path as long as the path to goal is not overestimated.

## 2.5 Uniform-Cost Search - Custom Search 1

Uniform cost search is a search function that determines the order of search based on the cost from the start point. This means that the search function will find the cheapest route not the shallowest route. When all costs are equal uniform cost search will behave like a breadth-first search as the shallowest route is also the cheapest route.

### Properties

$b = Branching\ Factor$
$d = Depthofleastcostsolution\ Factor$
$e = min(d[i, j])$

- Complete : Yes
- Time Complexity: $O(b^{\frac{d}{e}+1})$
- Space Complexity: $O(b^{\frac{d}{e}+1})$
- Optimal: Yes

## 2.6 Iterative Deepening Depth Limited Search - Custom Search 2

Iterative deepening search is an uniformed search algorithm that extends on Depth Limited Search. Depth Limited Search is itself a variation of the Depth First Search the difference between the two being that Depth Limited Search set a maximum depth on the depth first search. The extension that Iterative Deepening does is to repeatedly do a depth limited search but increasing the depth limit on each iteration.

### Properties

$b = Branching\ Factor$
$d = Depthofleastcostsolution\ Factor$
$m = Maximum\ depth\ of\ space$

- Complete : Yes

- Time Complexity: $O(b^d)$
- Space Complexity: $O(b * d)$
- Optimal: Yes if step cost = 1

## 2.7 Algorithm Performance

**Algorithm performance on base maze described in assignment sheet**

| | BFS | DFS | UCS | GBF | A* | IDDLS |
|---|---|---|---|---|---|---|
| Cost to Goal | 26 | 30 | 26 | 26 | 26 | 26 |
| Number of Nodes explored | 39 | 21 | 39 | 13 | 37 | 236 |

From these results we can see that the best performing algorithm is the GBF however the downfalls over GBF are not visible due to the fact that there was no major blockages in the path of least resistance. The A* algorithm still performed well but on such a small maze not much better than Breadth First. The Breadth first algorithm found the optimal path as it should but searched the most number of nodes. The Depth first did not find the optimal path but found it with many less nodes explored. Finally the iterative deepening search found the optimal path but explored a significantly larger number of nodes, while this is bad on time complexity the space complexity is actually quite small because it dumps stored nodes after each iteration.

**Algorithm Performances 3 randomly generated mazes**

| Breadth - First Search | Trial 1 | Trial 2 | Trial 3 |
|---|---|---|---|
| Cost to Goal | 19 | 18 | 11 |
| No. Nodes Searched | 123 | 137 | 52 |

| Depth - First Search | Trial 1 | Trial 2 | Trial 3 |
|---|---|---|---|
| Cost to Goal | 79 | 238 | 91 |
| No. Nodes Searched | 279 | 96 | 91 |

| Greedy Best - First Search | Trial 1 | Trial 2 | Trial 3 |
|---|---|---|---|
| Cost to Goal | 19 | 18 | 11 |
| No. Nodes Searched | 10 | 15 | 9 |

| A* Search | Trial 1 | Trial 2 | Trial 3 | |
|---|---|---|---|---|
| Cost to Goal | 19 | 18 | 11 | |
| No. Nodes Searched | 49 | 59 | 15 | |

| Uniform Cost Search | Trial 1 | Trial 2 | Trial 3 e |
|---|---|---|---|
| Cost to Goal | 19 | 18 | 11 |
| No. Nodes Searched | 162 | 900 | 42 |

| Iterative Deepening Search | Trial 1 | Trial 2 | Trial 3 |
|---|---|---|---|
| Cost to Goal | 19 | 15 | 11 |
| No. Nodes Searched | 461 | 203 | 235 |

These generated trials show just how different various algorithms perform when it comes to larger environments.While all the algorithms except for depth first found the optimal path it was merely coincidence that the optimal path was found by the BFS, due to the optimal path being the shallowest. It is clear as well that when the depth of the search space is much larger than the branching factor that the depth first search performs terribly searching large numbers of nodes and getting very long paths to goal. The Uniform cost search in these cases searched large a number of nodes due to the path of least resistance being away from the goal state, ultimately though it found the optimal path. These trials also confirm what we have shown about A* and GBF.

# 3 Implementation

The program has been implemented in python. The files associated with the implementation are:

- RobotNavigation.py - This is the main entry point for the program.
- maze.py - This contains a class called maze that determines the world that the search algorithm is to search
- PuzzleState.py - This contains the class of puzzlestate which stores useful information about a state, these include:
  - location - where the state is
  - parentState - what the state that the search came from

  - direction - the direction that the search took to get to it
  - neighbours - the valid states that are next to it
  - directionCost - how expensive it is to move in a particular direction
  - cost - how expensive it has been to get to this state
  - heuristic - if set contains a measure of how far the state is to the goal state in terms of manhatten distance.
  - Depth - how deep in a search is the state

- strategy.py - this contains the class for strategy. the strategy class sets out the basic template for the various different search algorithms. This template includes the properties:

  - Frontier - a list of states to be searched
  - Searched - a list of states that have been search
  - Puzzle - a maze to be searched
  - score - how many states have been searched

  The template also includes the functions

  - getnextstate - this is altered by the various search algorithms and is used to determine which state to search next.
  - addtofrontier - This adds state to the frontier list from which to search from only if they are not already in the frontier or they are not in the searched list
  - showPath - this sets the path to goal to a different value and draws it.
  - drawMaze - This draws a visualization of the maze and how the search algorithm is searching through it
  - solve - This is the function that actually performs the search.

  The subclasses of strategy include:

  - BFSStrategy - Overrides the getnextstate function to choose the First Item in the Frontier list (FIFO)
  - DFSStrategy - Overrides the getnextstate function to choose the last item in the Frontier list (LIFO)

  - DLSStrategy - Overrides the getnextstate function to choose the last item in the Frontier list (LIFO). Also adds a maxDepth property and a function. Finally overrides the solve function to include a limit on how deep the function can search
  - IDDLSStrategy - Overrides the getnextstate function to choose the last item in the Frontier list (LIFO). Also adds a maxDepth property and a function. Finally Overrides the the solve function to iteratively perform a Depth limited search with increasing depth
  - UCSStrategy - Overrides the getnextstate function to choose the state with the lowest cost in the Frontier list.
  - GBFStrategy - Overrides the getnextstate function to choose the state with the lowest heuristic value in the Frontier list.
  - AStrategy - Overrides the getnextstate function to choose the state with the lowest value of the sum of cost and heuristic value in the Frontier list.

- mazeDrawer.py - This file draws the maze and how the search algorithm is searching to the screen. To do this it uses the python module pygame

# 4 Features/Bugs/Missing Features

## 4.1 Features

- Ability to Select Search Algorithm
- Ability to toggle cost function
- Prints path cost
- Prints number of nodes searched.
- Displays visualization of search being performed:
  - Displays White for unsearched node
  - Displays Black for unsearchable node
  - Displays Green for node in frontier

  - Displays Brown for searched node
  - Displays Red for node in found path

- Ability to input cost function

### 4.2 Bugs

- The way states get added to the Frontier not quite working correctly

### 4.3 Missing Features

- GUI implementation for choosing search function and inputing costs

## 5 Research

### 5.1 Cost Function

The program has the ability to change the costs of different moves. The command line will ask if you want to change the cost function if yes will ask for input costs. This will change how the Uniform-Cost Search and the A* Search will search the maze as they will take into consideration the cost of making a particular move. The other search algorithms will not search any differently but the change in the cost function will show that they do not always find the cheapest solution.

### 5.2 Visualization

The Program is capable of visualizing how the different search method find their way through the maze. To do this i used a python library called pygame, https://www.pygame.org , in particular I used their documentation section to teach me the library enough to produce a simple visualization.

## 6 Conclusion

For this project I have successfully implemented various tree based search algorithms including Breadth-First Search, Depth-First Search, Greedy Best First Search, A* search, Uniform Cost Search and Iterative Deepening Depth

First Search. To show how the various search algorithms work I have implemented a visualization and the resultant path is printed to the console alongside the cost of the path and the number of nodes explored.

# References


[1] . [Online]. Available: http://www.seas.upenn.edu/~cis391/Lectures/uninformed-search%20fall%202015.pdf.

[2] . [Online]. Available: http://www.seas.upenn.edu/~cis521/Lectures/informed-search-I.pdf.

[3] [Online]. Available: https://www.pygame.org.