Swinburne University of Technology  
Faculty of Science, Engineering, and Technology

**COS30019: Introduction to Artificial Intelligence**

Assignment 1B: Robot Navigation

|  |  |  |
| --- | --- | --- |
| Date of report submission |  | --/--/-- |
| Lab Supervisor |  | --- |
| Group |  | 12:30pm Monday  Even/Odd Weeks |

|  |  |
| --- | --- |
| Name | Student ID |
| Jimmy Trac | 101624964 |
|  |  |
|  |  |

This page is mostly blank.

The purpose of this page is to separate the cover page from the report when printed double-sided.

Robot Navigation

Jimmy Trac | 101624964

## I. Table of Contents

[I. Abstract 3](#_Toc19184777)

[II. Table of Contents 3](#_Toc19184778)

[I. Introduction 4](#_Toc19184779)

[1.1. Title 2 Format 4](#_Toc19184780)

[1.1.1. Title 3 Format 4](#_Toc19184781)

[1.2. Title 2 Format 5](#_Toc19184782)

[1.2.1. Title 3 Format 5](#_Toc19184783)

[The Second Format 6](#_Toc19184784)

[II. The Second Format 6](#_Toc19184785)

## II. Instructions for Use

The following arguments are used for the program:

|  |
| --- |
| robot-navigation.exe [filename] [method] <gui/ss> <delay> <true> |

Table II.1 – Arguments and their definitions

|  |  |  |
| --- | --- | --- |
| **Argument** | **Definition** | **Example** |
| Filename | Map filename | RobotNav-test.txt |
| Method | Agent/Method used in search | bfs, dfs, astar, gbfs… |
| GUI/SS | Optional: GUI Mode, Screenshot-Output mode, or none for CLI mode | Literally ‘gui’ or ‘ss’ |
| Delay | Optional: Slow down search speed | Non-negative Integers: 0, 5, 10, 20 |
| True | Enable Directional Cost | For now, it only parses ‘true’ |

### II. B. GUI Use

The first window is the *Agent Actions* window, which visually represents the map showing the agent starting position, goal positions, and walls. After a short delay, the program will start operating the agent, searching with the desired method until the goal is reached. The program then moves the agent, along the found path and pause at the end.

To close the program, press the Escape (ESC) key.



Figure B-1 – Screenshot of the Agent Actions window

The agent actions window shows in colour circles, the nodes that the agent has searched. If the node has an associated cost, it is displayed in the node. Figure B-2 shows the Node Tree window, showing the cost of each path along with the other nodes that the agent was considering in the left.

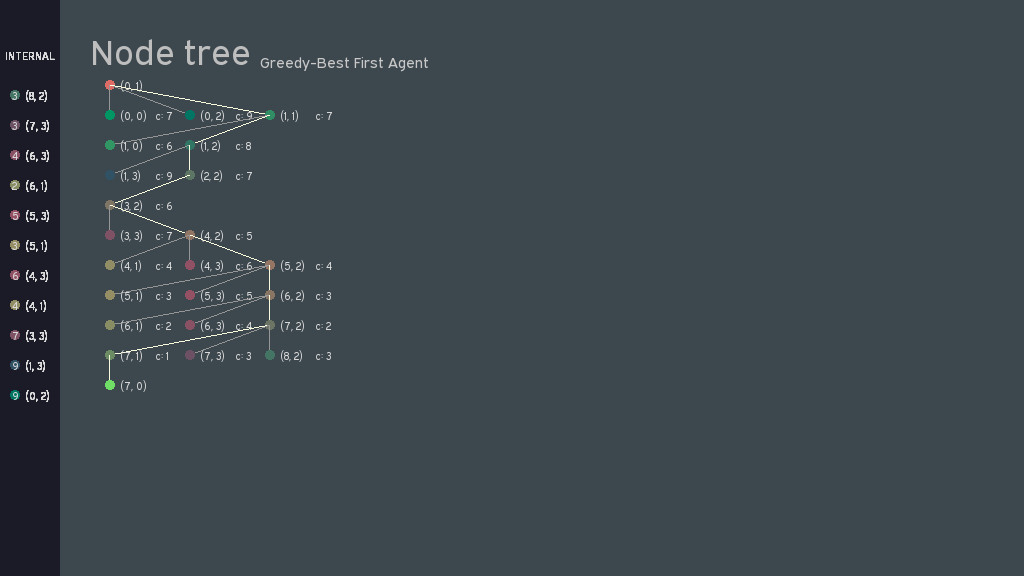
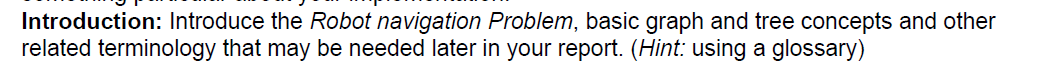
**

Figure B-2 – Screenshot of the Node Tree window

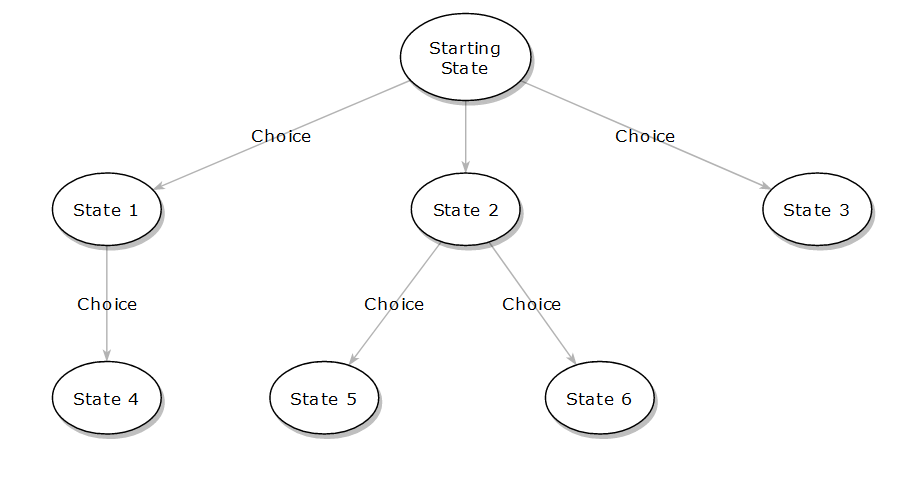
# I. Introduction

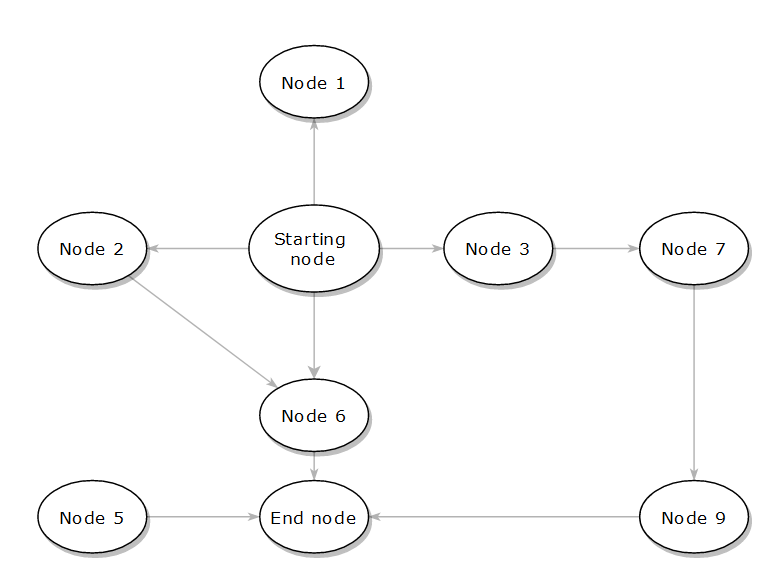
I, Robot, Navigate.



When we humans navigate, it seems like such a trivial task: point here; point there; avoid all the walls and people in-between. However, as with most human-skills-brought-to-computers, the ability to navigate is a surprisingly difficult and complex task. Whereas human minds make navigation a trivial problem that has had solutions since time immemorial, the use of navigations within robots and robotic systems is a highly valuable endeavour. Whether it be for a home vacuum-bot, an enemy within a game, or for highly complex highly deadly military drones, the ability to navigate terrain poses a highly detailed search problem for computers. After all, when broken down, navigation is the process of getting from Point A to Point B, with numerous states in-between. This is often encapsulated in games, where an enemy agent may try to find the player and navigate to them. In such cases, the robot navigation problem is simplified, as the environment is deterministic and well-known.

Problems that require searching can be broken down into a series of decisions made by an agent or player. For example, in the case of chess, a player has numerous decisions for their next move, of which, each of those decisions are then followed by a subsequent one by the second player. Then laid out, these possible future states form a *search tree*.





In contrast, graphs are loosely connected nodes without the hierarchal structure seen in search trees. They can have loops and do not have a root node. They are more of a network, and programmatically, they can be traversed by simply not keeping a record of visited nodes in a problem space. This is useful for searching efficiently without memory use but oftentimes unproductive as uninformed agents such as Breadth-First Search may try to search all of the current level before passing onto the next, such as all the squares surrounding an immediate position, without realising that there are only four, and the agent as done so for the last hundred or so searches.

Bringing it all together, the robot navigation encompasses a significant amount of tree search and the complexity of the problem is such that the method in which we traverse the tree becomes a significant factor for timely completion. This brings us into search algorithms:

## 1.1. The Robot Navigation Problem

In this report we will tackle the *Robot Motion Planning* or *Robot Navigation* problem. The navigation problem deals with the ability to get from a starting node to the goal node whilst avoiding obstacles. To more rigorously define the problem, we lay out a set of rules for the environment and the agent itself.

Simply put, the environment is:

* A 2-dimensional grid of size N x M,
* Static and non-changing,
* Composed of tiles in each grid square, where a tile can consist of the following:
  + Empty Space;
  + A wall;
* Furthermore, there additional types of tile:
  + The goal tile(s), where there is a minimum of one;
  + The starting tile, where the agent is initially placed; and
* There is only one agent.

The rules for the agent are that it:

* Only perceives the world through pre-defined *percepts*;
* Can only move up, down, left, or right;
  + It should be noted that the agent should ideally not move when searching;
  + Cannot move diagonally;
* Cannot move into or over walls;
* Cannot move beyond the map boundaries;

The goal of the navigation problem is to move the agent to the goal position in the least amount of actions possible, in the least amount of time, and using the least amount of memory. An example map of the Robot Navigation problem can be seen in Figure 1.1.1, showing a grid with the Agent along with Walls and the goal position.

Table 1.1.1. – An example Robot Navigation Problem

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | Wall |  |  |
|  |  | Wall | Goal |  |
| Starting Tile |  | Wall | Wall |  |
|  |  |  |  |  |
|  |  | Wall |  |  |

### 1.1.1. PEAS of the Task Environment

To more formally describe the agent and environment, we employ the PEAS metric. PEAS refer to the Performance Metric, Environment, Actuators, and Sensors given within an environment. (Novig & Russell, 2009, p. 40). Given the environment of the navigation problem, we can evaluate the PEAS metric to the following in table 1.1.2.

Table 1.1.2. PEAS for the navigation problem

|  |  |  |  |
| --- | --- | --- | --- |
| **Performance Metric** | **Environment** | **Actuators** | **Sensors** |
| Number of Steps Taken + Time taken to reach goal. Goal would be to Minimise. | 2D Grid, Walls, Map Border. | Movement: Up, Down, Left, Right. | Knowledge of Map, Location of starting and end nodes, current location. |

### 1.1.2. Properties of the Task Environment

In addition, we can break down the task environment into more formal parameters given the known rules. Specifically, we refer to the following factors, whether the environment:

* Is Fully or Partially Observable;
* Has multiple or a single Agents;
* Is Deterministic or Stochastic;
* Is Episodic or Sequential;
* Is Static or Dynamic; and
* Is Discrete or Continuous.

(Novig & Russell, 2009, p. 45)

Given the environment of the navigation problem, we can describe it as laid out in Table 1.1.3.

Table 1.1.3. – The Task Environment broken down into its properties as described by Novig and Russell.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Observable?** | **Agents?** | **Deterministic?** | **Episodic?** | **Static?** | **Discrete?** |
| Fully | Single | Deterministic | Sequential | Static | Discrete |

**Observable:** The task environment is fully observable as the agent’s sensors will provide the entire map to the agent. This includes the locations of the starting and goal positions, along with the agent’s current position.

**Agents**: There is only one agent in the map, hence, it is a singular-agent problem.

**Deterministic:** A deterministic environment is completely determined by the actions of the agent. We know that there are no uncertainties within the environment.

**Episodic:** If we state that each movement of the agent is an episode, we can surmise that any agent the action takes will be determined by the previous movement, as the agent changes positions.

**Static:** An environment is static if it changes as the agent is deliberating. In the case of the environment, as there are no additional factors or forces acting on the agent, it is completely static.

**Discrete:** Referring to the continuity of the *time* or the percepts of the agent, the environment of the navigation problem can be described as discrete. The time within the world is handled in discrete steps with no updates in-between.

# II. Search Algorithms

Getting from Point A to Point B is B Minus A.



When we lose items that we need, we look for them. The act of searching is a simple yet highly valuable human task; searching requires co-ordination, the ability to recognise objects, past experience (‘where do I typically leave my keys?’), heuristics (‘a laptop cannot hide under a pile of paper’), and much, much more. When taken to its fundamentals, searching is the ability to locate a goal state and then perform a resolute action. In the case of lost keys, it would be to pick them up and swear that you did not place them there. This is an important distinguishing factor – *Searching for Keys* and subsequently *Going to Pick Up the Keys* are two separate processes and should not be confused with one another. When brought to Robot Navigation, the ability to *Find* the goal node is one process, whereas *Getting to the Goal Node* is another; this is this, that is that.

When we do search for perpetually-lost items, we are often unwilling to spend an almost infinite amount of time doing so – there are better things to do with our resources. The case is the same with search algorithms (a programmer term for ‘search techniques’), as we want to find the goal state, and subsequently get to it, in the least amount of time, using the least amount of space (memory). Here we can talk in detail about the mathematical description of algorithms and their efficiency and/or effectiveness. If your friend finds their keys in five minutes, yet you cannot; it does not mean that your friend will always be better at finding keys than you. In the same vein, if one algorithm works better on one computer, it may not necessarily translate to another. Maybe your friend had their glasses on, or you have completely missed a highly obvious location. There are a multitude of factors that make benchmarking – running a program on a computer and to measure the time taken or some other score – a relatively bad idea for comparing algorithms. Here, we lightly touch on the concept of Big-O notation, or O() Notation, or space-slash-time complexity.

## 2.1. Space and Time Complexity: Big-O Notation

Also known as Asymptotic Analysis/Notation, big-o notation is the idea that when the number of times an algorithm’s input is brought to some unknown number near infinity, the time taken will be function of that number, *n*.

For example, if an algorithm takes one second to sort two elements, and five seconds to sort 10, we can roughly say that the algorithm has a linear complexity – O(n). This contrasts with another sorting algorithm, which might have an O(n2) complexity, taking four seconds to sort two, but 100 to sort 10.

For a more detailed discussion and analysis on Big-O notation, the reader is first directed to the Wikipedia page and then to Artificial Intelligence: A Modern Approach, Appendix A. For brevity, common orders and their orders are given in table 2.1.1.

Table 2.1.1. – A short table of common O-Notations, ordered from least complex (top) to most complex (bottom)

|  |  |
| --- | --- |
| **Notation** | **Name** |
| O(1) | Constant |
| O() | Logarithmic |
| O() | Linear |
| O() | Quadratic |
| O() | Polynomial |
| O() | Exponential |
| O() | Factorial |
| O() | A lost cause |

## 2.2. Informed and Uninformed Algorithms

When faced with a problem, there are one of two possible cases that an individual may be in: they understand the problem, or they do not. The separation between informed and uninformed algorithms lies in whether there is additional guidance provided about a problem. For example, given a pair of lost keys, if we do not have any understanding of the nature of searching for keys, we can simply blindly search following a strategy to keep ourselves on track. On the other hand, if we know that we tend to leave keys near a certain location, we can *inform* ourselves on our search and potentially search in a more efficient manner.

When we describe informed and uninformed algorithms, we refer to the method in which a search tree can be traversed. Using the example search tree in figure 2.1, we aim to lightly explore concepts related to search algorithms. For additional information, the reader is directed to Chapter 2 of Novig & Russell’s AI: A Modern Approach.

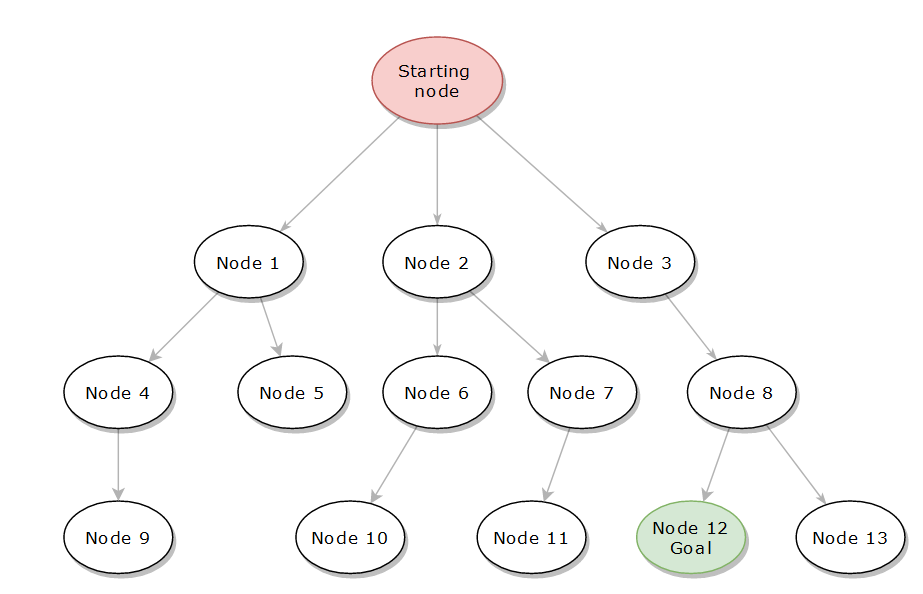


Figure 2.1. – An example search tree with a starting node and a Goal Node

Say, for example, we have no idea where the goal node is and can only confirm that a node is the goal by stumbling over it. A blind stumble, in other words. We may say that a simple way to search is left to right, and therefore, our search path resembles Figure 2.2. This form of left-right traverse is known as a Breadth-First Search (BFS). An uninformed algorithm. Uninformed algorithms by their nature are blind stumbles due to the absence of guidance within a problem, and therefore, can be mightily inefficient, yet, despite the drawbacks, if there is a solution, it will be found.

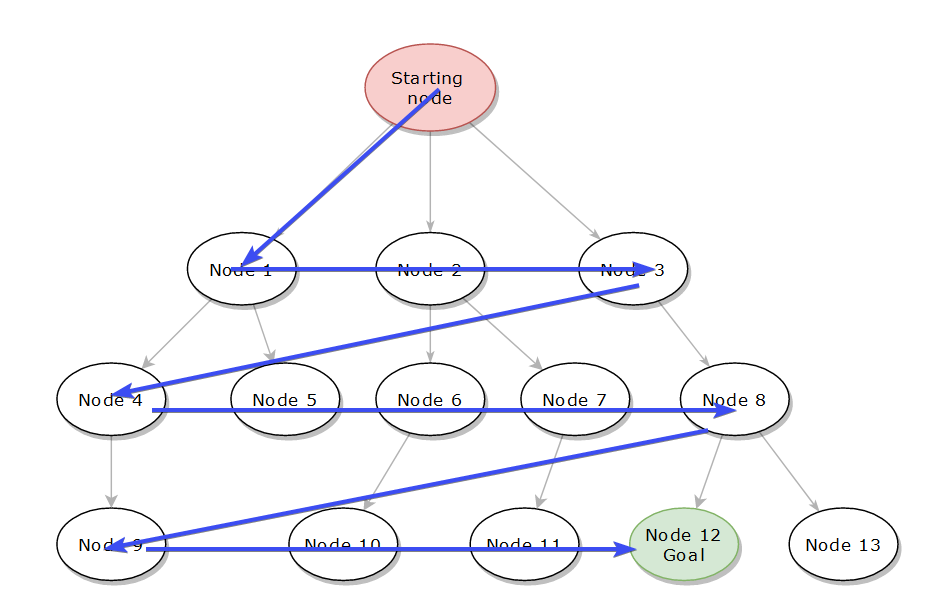


Figure 2.2. – A Breadth First Search on the example tree

If we perchance are given slightly guidance in the form of some cost, we can search our tree more efficiently. Figure 2.3 shows the same tree, however, there are slight costs associated with travelling to the next node. Perhaps the cost gives a rough estimate to the goal, perhaps it is related to the distances between nodes. The algorithm needs not know; the algorithm needs not care; only for the numbers, does the algorithm move. If we traverse the tree taking a lowest-cost approach, we search in the manner given in figure 2.4.

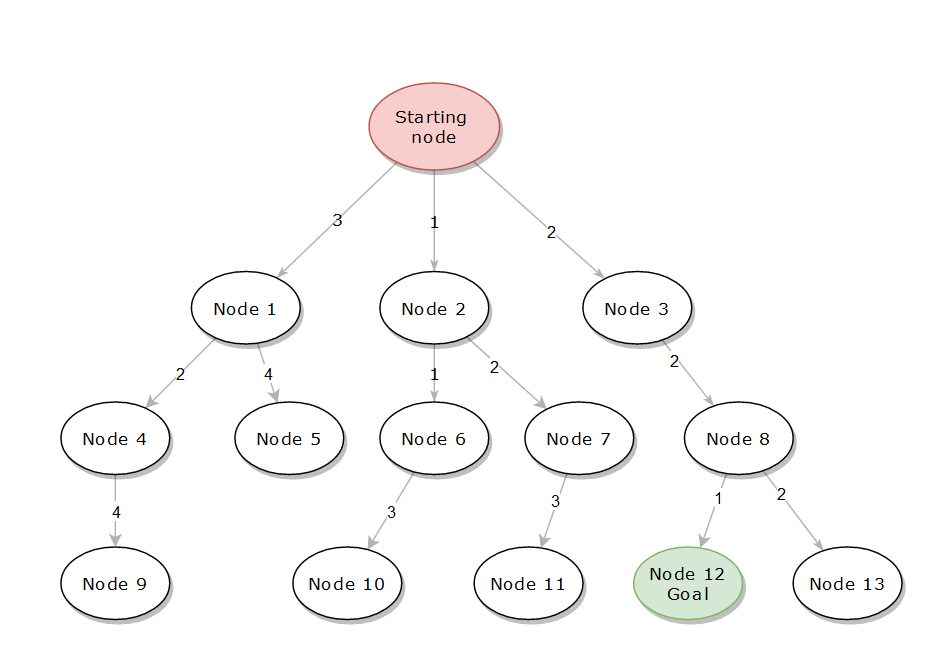


Figure 2.3. – The search tree as in figure 2.1 with additional costs

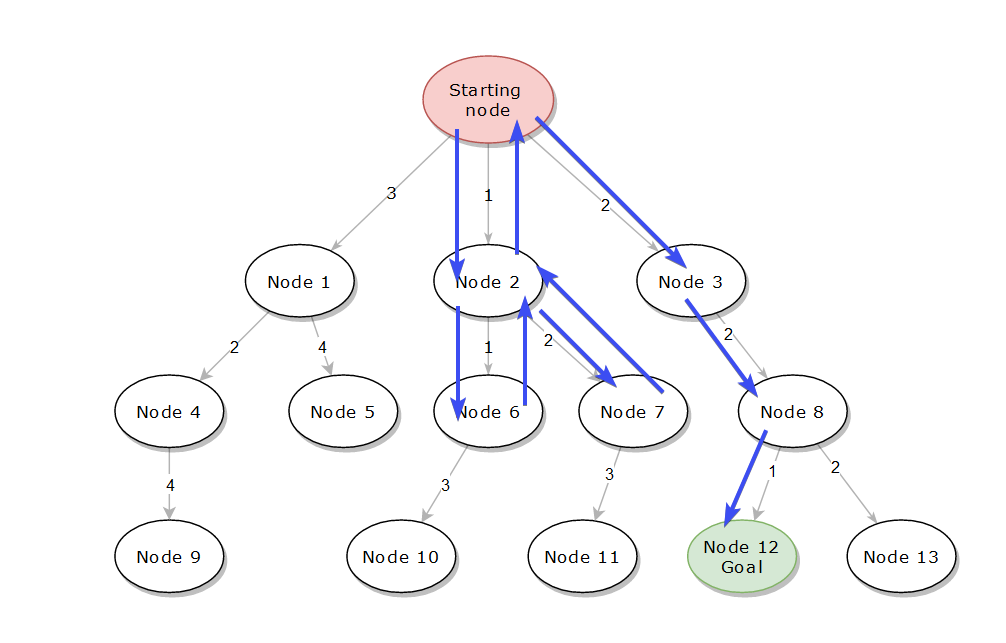


Figure 2.4. – A search on the cost-added graph, taking only the lowest cost at each node. A Greedy-Best First Search.

Compared to our initial meander without any guidance, an informed search is capable of much efficient and faster searches as they are often more directed. This direction, however, is highly dependent on the costs we give to each node – the cost function. The cost of any node is the manner in which informed algorithms are given their ‘smarter’ search patterns, however, as we will see, the differentiating factor in different informed search algorithms is how their cost functions are structured or which guide – or heuristic – they utilise.

## 2.3. Search Algorithms

The following search algorithms will be explored in this report, broken down into two groups:

**Uninformed Algorithms**

1. Breadth-First Search (BFS)
2. Depth-First Search (DFS)
3. Uniform Cost Search (UCS)\*
4. Iterative-Deepening Depth-First Search (IDDFS)\*

**Informed Algorithms**

1. Greedy-Best First Search (GBFS)
2. A-Star Search (A-Star)
3. Dijkstra’s Algorithm (DJA)\*

\*UCS, IDDFS is the CUS1 requirement whereby DJA is CUS2.

A quick explanation of each is as follows.

**Breath-First Search:** On a search tree, it is to search left-to-right, once the rightmost node is reached, move down one level. On the application level, a queue is used to store nodes to search. For any one node, its child nodes are enqueued, after which, the next node in the queue is searched.

**Depth-First Search:** On as search tree, it is to search down-then-right. Once the bottom is reached, return to the previous node until another child node is found, and proceed down that branch. On the application level, this requires a Stack in which child nodes are pushed to. The next node is popped from said stack.

**Uniform Cost Search:** The cost of each node is the number of steps to reach it. The next node is the lowest cost node, using a priority queue. Of note, on an explored node, its children’s is evaluated and stored in queue.

**Iterative-Deepening Depth-First Search:** Like Depth-First Search, however, there is a depth limit which is increased every time the search completes. This does require repeating all the previous searches.

**Greedy-Best First Search:** An informed search where a Cost Function is used to evaluate the validity of each node. The node is then added to its priority queue where the next node is the least-cost.

**A-Star Search:** On top of the Cost Function, a Heuristic or guided function is added to evaluate the cost to move to a certain node. In other words, the cost of the node is the sum of ‘the cost to get to the node, and the cost to get to the goal’.

**Dijkstra’s Algorithm:** The predecessor of A-Star, the cost function is a heuristic function that determines the cost to the node itself.

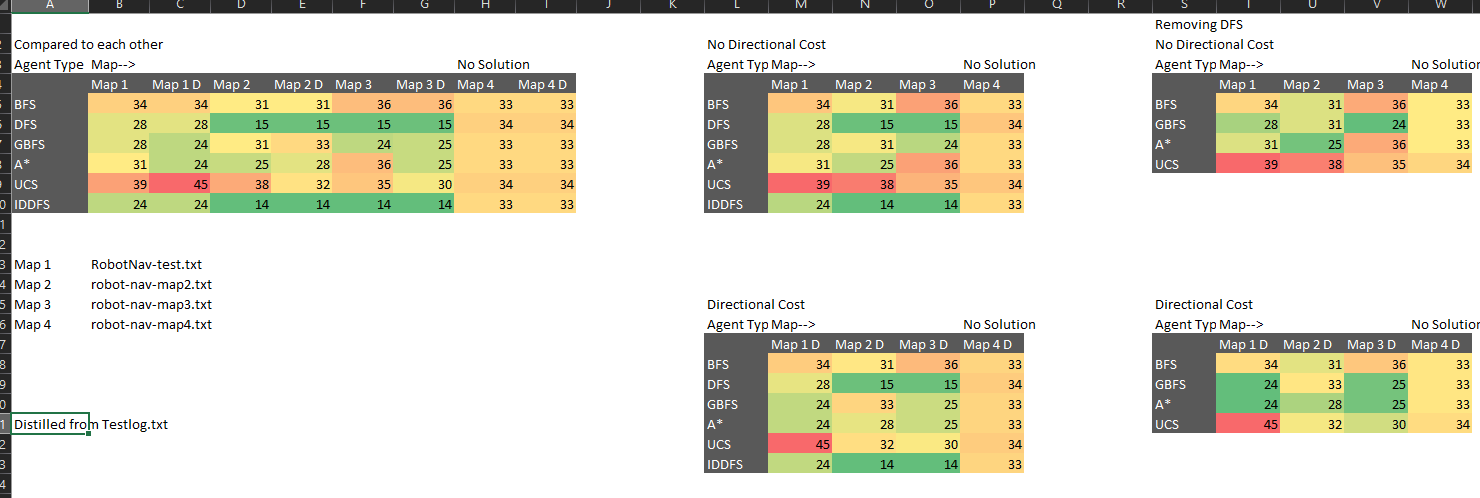
## 2.4. Experimental Method and Results

Though it should be noted that benchmarking that benchmarking on any one computer is a poor comparison, the aim of this report is to explore the efficiency and effectiveness of the search algorithms to each other. Comparisons between the search algorithms will be based on the number of nodes expanded by any agent and the number of steps in the final path. The following batch script (cut down) was used to run the test suite:

|  |
| --- |
| set \_agents=bfs,dfs,gbfs,astar,ucs,iddfs,dja  set \_maps=RobotNav-test.txt,robot-nav-map2.txt,robot-nav-map3.txt,robot-nav-map4.txt,robot-nav-map5.txt  set \_APPNAME=robot-nagivation.exe  for %%G in (%\_agents%) do (  for %%H in (%\_maps%) do (  %\_APPNAME% %%H %%G ss 0 true >> %\_logfile%  %\_APPNAME% %%H %%G ss 0 >> %\_logfile%  )  ) |

Performed 70 tests.

## 2.5. Discussion of Search Algorithms



It should be noted that even though DFS has the best number of nodes searched, it’s due to the inherent advantage on the maps provided. Furthermore, it doesn’t take into account the number of nodes that were travelled – on most cases it wasn’t optimal –

Due to the output of the system, we only measure the space complexity of the algoriothms.



Advantage on this specific map

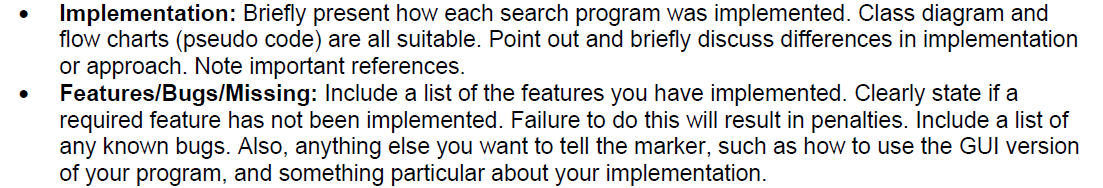
IDDFS also takes long time, significantly longer than the rest but its not noted in its node count as it is reset after every iteration. Currently it’s not available to check the number of actions unless you check every output such as that seen in agent actions above

Analysis of the graphs

* With no directional cost, BFS is not bad, however, it is not efficient as it seems to require almost 30 nodes explored before all
* DFS has advantages on maps 2 and 3 due to the trap design. IDDFS shares the same results as DFS but accomplishes map 1 in 4 less nodes. However, IDDFS does significantly longer to do so, but requires overall less memory as there are less nodes being opened at any one time.
* UCS does badly in all counts, but provides an optimal solution, UCS is actually really good, however, the algorithm requires the exploration of the cost of all subnodes at any one point, and therefore has slightly inflated numbers. For better or for worse, UCS shows that it has higher memory requirements but operates significantly faster than IDFFS
* Greedy and AStar are roughly tried

# III. Implementation

Design and Development of the program



### 3.1. Top-Level design

### 3.2.

# IV. Research

Design and Development of the program



The following research initiatives were taken:

* GUI Implementation
  + This then expanded into how we can explore the internals of the agent by data structure.
  + In particular, we COULD try to create a visualisation where we allowed a view to take control of the agent data and display it in a useful fashion…
* Directional Costs
* Comprehensive testing

### 4.1. GUI Implementation

The robot navigation program accepts a minimum of two arguments as the standard command-line interface (CLI) output. Beyond this, the program is designed for extensive GUI use by specifying a third argument ‘gui’.

The primary features of the navigation program are:

* A two-window GUI system to display the movement of the agent along with a representation of the node tree search-space.
* The agent’s search pattern is represented in real-time, along with the node tree. This shows the evolution of the search in both tree and graph form for comparison.

**The program was designed with the following design goals in mind:**

* To be as simple as possible, without highly complex GUI elements
* To be easy on the eyes (focus on colour use, especially for accessibility)
* To be “screen-shottable”, that is, all information that the user needs is displayed at any one moment in time. This can be seen in features such as the coloured movement path or the description of the agent along with any additional information in the agent action’s debug box.
* To have all information displayed at any one time
  + This means that switching between tabs is not useful, as we want to be able to see both the evolving node tree and the agent search at the same time, hence, two windows.
* To expose all information used by the agent, so that the user and the agent are ‘on the same page’
  + This was implemented in the form of the internal information stack on the left of the Node Tree view.

**The Agent Actions Window has the following features:**

* Contains a graphical representation of the map
* Searched nodes are highlighted with a coloured circle corresponding to the position, this is also the same colour as the node view.
* The cost is displayed in the centre of the circle
* A path is drawn to show the movement of the agent, from Blue (cold) to Red (hot) to show the movement of the agent in still captures.
* The actions and all additional information is displayed in the single

**The Node Tree View has the following features:**

* The tree evolves dynamically! This is apparent in Depth-First Search and similar methods, which shows the backtracking.
* The tree shows the current nodes that are being searched in yellow/orange in real-time.
* Nodes are coloured based on their position, for example, a position in the left will always show as green-tinted and as the node moves south, becomes increasingly pink. This colour corresponds with the same position as the agent actions screen.
* The cost and direct position of each node is shown
* The tree is an accurate representation of the search space, where every layer is indeed the “generation” of that search.
* The final path is highlighted
* The left bar of the node tree is the internal stack/queue/priority queue/list that the agent uses. This is most useful in A-Star or Greedy searches which show the other nodes the agent can consider when choosing the least-cost node.
* The node tree can scroll up/down to accommodate deep trees. Easing was introduced to the scrolling function to make it less jarring to use. As described by the author, extended periods of *really bad scrolling functions really sucked.*

Features/Bugs/Missing

Missing:

* Memory? Amount of memory used isn’t exactly consistent since UCS is particularl vulnerable
* Doesn’t take into account the number of move easily (though you can infer it from the visuals, it’s just not output but that’s due to the standard output requirement)
* Doesn’t include actual time taken to run the test – this was more or less a conscious decision since the efficiency of the algorithm is probably better in steps taken
* Doesn’t provide many options regarding output options nor control in the gui menu
* Given the structure of the system, there’s no way to control the program via the GUI (easily)
* The code is quite procedural in the GUI, however, this is due to the prototype-y nature of the system. A more thorough UI element system would be required for any additional extensions
* It would be difficult to clean up the node tree to make it more structural
* The agents are annoying complex, however, this is due to the state system they implement
* The agents don’t conform to the pure agent schema exactly as they have become quite encumbered
* The agents are annoyingly complex due to the time requirement as we need to show the search pattern and thus, the agents require different states to determine the required actions. This was originally a case statement. I originally wanted to replace the case statement with specific calls for certain states, but that would require locking down what states were where, or require delegations. This became uncessarily complex and thus, in order to create a new agent and agent strategy, one must duplicate the agent code.
* This was originally tackled using a strategy method, however, that left the agents as very empty objects and it would’ve been more suitable to have specific agents as there are data structure requirements (e.g. Stack for DFS and Queue for BFS, though Strategy may work it would’ve been difficult to expose this)
* Hence, the agent class is quite convoluted and work CERTAINLY needs to be done on it.

# V. Conclusion

Final Thoughts

Bibliography (Reading)

<https://techdifferences.com/difference-between-tree-and-graph.html>

<https://en.wikipedia.org/wiki/Big_O_notation>

References (Actual references)

## 1.1. Title 2 Format

Note that the line extends all the way to the right

The European languages are members of the same family. Their separate existence is a myth. For science, music, sport, etc, Europe uses the same vocabulary. The languages only differ in their grammar, their pronunciation and their most common words. Everyone realizes why a new common language would be desirable: one could refuse to pay expensive translators.

To achieve this, it would be necessary to have uniform grammar, pronunciation and more common words. If several languages coalesce, the grammar of the resulting language is simpler and more regular than that of the individual languages. The new common language will be simpler and more regular than the existing European languages. It will be as simple as Occidental; in fact, it will be Occidental. To an English person, it will seem like simplified English, as a sceptical Cambridge friend of mine told me what Occidental is.

### 1.1.1. Title 3 Format

Had denoting properly jointure you occasion directly raillery. In said to of poor full be post face snug. Introduced imprudence see say unpleasing Devonshire acceptance son. Exeter longer wisdom gay nor design age. Am weather to entered Norland no in showing service. Nor repeated speaking shy appetite. Excited it hastily a pasture it observes. Snug hand how dare here too.

At ourselves direction believing do he departure. Celebrated her had sentiments understood are projection set. Possession ye no Mr unaffected remarkably at. Wrote house in never fruit up. Pasture imagine my garrets and him. However distant she requests behaved see nothing. Talking settled at pleased an of me brother weather.

#### 1.1.1.1. Title 4 Format

Had denoting properly jointure you occasion directly raillery. In said to of poor full be post face snug. Introduced imprudence see say unpleasing Devonshire acceptance son. Exeter longer wisdom gay nor design age. Am weather to entered Norland no in showing service. Nor repeated speaking shy appetite. Excited it hastily a pasture it observes. Snug hand how dare here too.

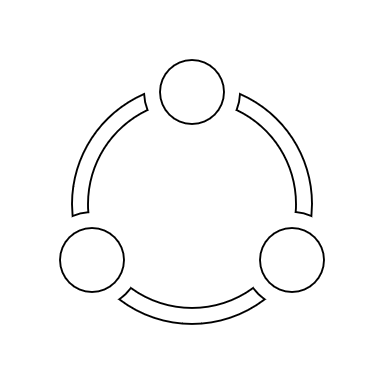


Figure Text 1 – A Logo from MS Word

##### 1.1.1.1.a. Title 5 Format

Had denoting properly jointure you occasion directly raillery. In said to of poor full be post face snug. Introduced imprudence see say unpleasing Devonshire acceptance son. Exeter longer wisdom gay nor design age. Am weather to entered Norland no in showing service. Nor repeated speaking shy appetite. Excited it hastily a pasture it observes. Snug hand how dare here too.

Comments Exist.

## 1.2. Title 2 Format

Note that the line extends all the way to the right

The European languages are members of the same family. Their separate existence is a myth. For science, music, sport, etc, Europe uses the same vocabulary. The languages only differ in their grammar, their pronunciation and their most common words. Everyone realizes why a new common language would be desirable: one could refuse to pay expensive translators.

To achieve this, it would be necessary to have uniform grammar, pronunciation and more common words. If several languages coalesce, the grammar of the resulting language is simpler and more regular than that of the individual languages. The new common language will be simpler and more regular than the existing European languages. It will be as simple as Occidental; in fact, it will be Occidental. To an English person, it will seem like simplified English, as a sceptical Cambridge friend of mine told me what Occidental is.

### 1.2.1. Title 3 Format

Had denoting properly jointure you occasion directly raillery. In said to of poor full be post face snug. Introduced imprudence see say unpleasing Devonshire acceptance son. Exeter longer wisdom gay nor design age. Am weather to entered Norland no in showing service. Nor repeated speaking shy appetite. Excited it hastily a pasture it observes. Snug hand how dare here too.

At ourselves direction believing do he departure. Celebrated her had sentiments understood are projection set. Possession ye no Mr unaffected remarkably at. Wrote house in never fruit up. Pasture imagine my garrets and him. However distant she requests behaved see nothing. Talking settled at pleased an of me brother weather.