CS5011: Assignment 1 – Search - Marine Navigation Planner

Student ID: 220029176

Date of Submission: 08/02/2023

Word Count: 1852

Table of Contents

Table of Figures

1. Introduction

The objective of this assignment was to implement, run and evaluate several search algorithms for the given scenario, Marine Navigation Planner. Various maps and configurations have been provided for testing and evaluation.

Uninformed and informed search algorithms have been implemented. The uninformed algorithms included depth-first and breadth-first. The informed search algorithms included best-first and A* search. Additionally, the uninformed search, bi-directional, was implemented.

The instructions to run the search algorithms can be found in section 1.2. The design and implementation have been documented in section 2. The testing and evaluation of the search algorithms can be seen in section 3 and 4.

1.1 Parts Implemented

Agent	Status
Basic: Breadth-First (BFS)	Attempted, Tested, Fully Working.
Basic: Depth-First (BFS)	Attempted, Tested, Fully Working.
Intermediate: Best-First (BestF)	Attempted, Tested, Fully Working.
Intermediate: A* (AStar)	Attempted, Tested, Fully Working.
Advanced: Bi-Directional (BiDir)	Attempted, Tested, Fully Working.
Advanced: Weather Forecast Management	Attempted, Tested, Fully Working.

1.2 Compilation and Execution Instructions

1.2.1 Running the Code

To execute the code, please do the following:

1. Download the zip file “CS5011-Assignment1-220029176” and unzip it to a chosen folder location.
2. Open a terminal prompt from within the src folder location.
3. Enter the following command into the terminal “javac A1main.java”
4. Enter the following command into the terminal “java A1main” following by the desired search algorithm, configuration ID and optional coordinates, please see structure within 1.2.2.

1.2.2 The command structure

The navigator runs on the following structure:

java A1main <DFS|BFS|AStar|BestF|BiDir> <ConfID> <Coord>

<DFS|BFS|AStar|BestF|BiDir>: The search algorithm to be used by the agent.

<ConfID>: The name of the configuration in the Conf class i.e., CONF20

<Coord>: The coordinate location of the severe weather conditions. (**Optional**)

2. Design and Implementation

2.1 Problem Definition

A ferry must cross a given map from the starting coordinate to a goal coordinate. The map is represented by a two-dimensional triangular grid system with cells marked either as '0' for free cells

or '1' for land. The ferry can only move from one cell to another along the grid, with each step costing 1, and can move in 3 directions (Left, Right, Up/Down) depending on the triangle shape. The ferry cannot cross any land cells or go beyond the map boundaries. Within the context of this assignment, the goal is to implement and evaluate search algorithms based on their solution quality (path cost) and efficiency (number of search states visited).

2.2 The Performance, Environment, Actuators and Sensors (PEAS) Model

The performance measure used for this problem was the total path cost across the graph. The characteristics of the environment were identified as; fully observable, deterministic, episodic, static, discrete, and single agent. Due to the nature of this program, there are no actuators as the subsequent actions are determined based on the frontier of the search. The sensor used within this model include the cost of the neighbouring grid cells.

2.3 Search Components

The search components include the state space, actions, initial state, goal state, search and cost. The state space for this model is the reachable coordinates on the grid, indicated by a “0”. The actions are limited to the adjacent coordinates of the current agent’s location. The initial state is the coordinate within the triangular grid provided by the user. The goal state is a coordinate within the triangular grid provided by the user. For the purposes on this assignment the search varies based on the user input (BFS, DFS, BestF, AStar and BiDir). The cost component was based on the Manhattan distance and the Euclidian distance.

2.4 System Implementation

2.4.1 The Search Algorithm

The general search algorithm (GSA) was used within this assignment. This was based on the model provided in Lecture 2 of CS5011, Artificial Intelligence Practice (Rahman, 2023) and Russell & Norvig’s model (Russell & Norvig, 2010). The pseudo code can be seen in Appendix A and B, respectively. GSA performs a search by creating a frontier of the node to be explored. The next node is chosen from the frontier, depending on the search algorithm, and then tested against the goal state. If it is not equal to the goal state, then the node is evaluated for successors. If successors are found, they are added to the frontier and the initial node is added to the expanded list. The next node is then chosen from the frontier. This continues until a successor has the same state as the goal (success) or there are no more nodes in the frontier i.e., all possible options have been explored (failure).

The implementation of the various search algorithms differ only regarding how the frontier manages adding, removing, and organising the order of the nodes. A Deque and a LinkedList were chosen for the frontier within the uninformed and informed search respectively. BFS utilises a First-In First-Out (FIFO) approach to frontier organisation, this is commonly represented as a queue. Whilst DFS utilises a Last-In First-Out approach (LIFO) which is commonly represented as a stack. A Deque offered the ability to add the given node to the back and the front of the collection which suited the required functionality for BFS and DFS. Informed search requires the frontier to be continually updated and evaluated for the next best option when a new node is added. This functionality would not be possible with a Deque. Therefore, a LinkedList was chosen to represent informed search.

Heuristics were required to implement Best-First and A* search. For Best-First, the heuristics chosen were Manhattan distance over the triangular grid to the goal coordinate. A* incorporated this heuristic plus the Euclidian distance between the agent location and the given node.

For the Bi-Directional implementation, the GSA was used. However, two search trees are initiated, one rooted in the start coordinate and the other rooted in the goal coordinate. The two search trees use the BFS implementation as recommended by Russell and Norvig (2010). The goal test is replaced with a check for an intersection between the search tree node history. If an intersection is found, then a solution can be achieved. To attain this goal testing functionality, the expanded lists are checked after the frontier has been added.

2.4.2 Successor Function

The successor function returns a set of all the legal coordinates. A coordinate is legal if it is within the grid and does not contain an island (displayed as 1). Additionally, the direction of the current triangle determines if the upward or downward coordinate is legal i.e., if the triangle is pointing upwards then the downward coordinate is legal and vice versa. The legal coordinates are added to the return set based on the following directional hierarchy; Right, Down, Left, Up.

2.4.3 Weather Forecast Management

Weather forecasting plays an important role in marine navigation and is vital in maintaining the health and wellbeing of the vessel and crew. In a real navigation system, the dynamic nature of weather within naval routes needs to be captured and evaluated. To implement this functionality, the user must add additional coordinates to the initial command prompt to simulate hazardous grid locations that cannot be entered i.e., Java A1main BFS CONF1 (1,1) (3,4). These coordinates are added to the map that will be used by the search algorithm and will be avoided by the agent.

3. Testing

The testing for this project involved manually transcribing each search algorithm against configuration JCONF03. This configuration places the start state at position (0,0) and the goal position at (2,2). The map is a 3x3 triangular grid.

Test	**Passed?**	**Proof**
BFS	Yes	See Figures 1 & 2
DFS	Yes	See Figures 3 & 4
Best-First	Yes	See Figures 5 & 6
A*	Yes	See Figures 7 & 8
Bi-Directional	Yes	See Figures 9 & 10
BFS With Weather on (0,1) & (1,2)	Yes	See Figures 11 & 12

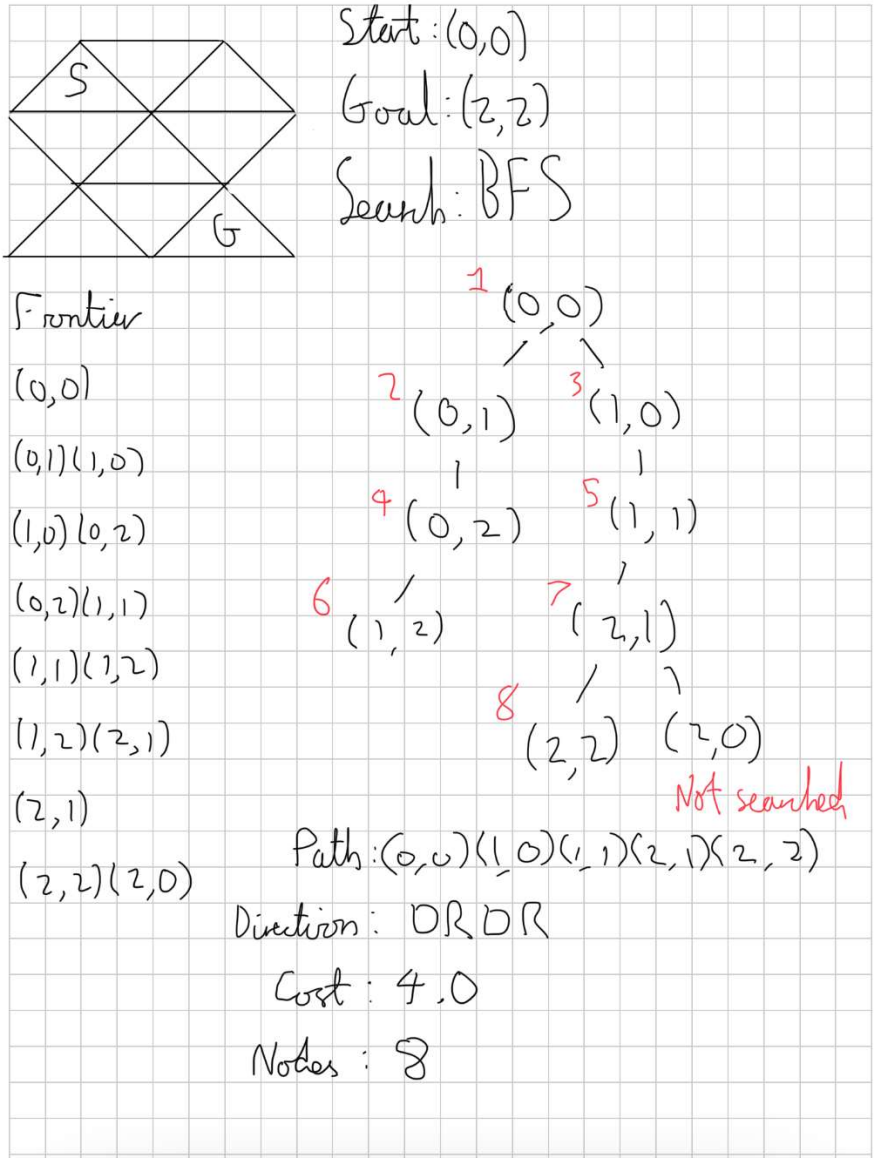

Figure 1: BFS Handwritten Implementation of JCONF03

```
[(0,0)]
[(0,1),(1,0)]
[(1,0),(0,2)]
[(0,2),(1,1)]
[(1,1),(1,2)]
[(1,2),(2,1)]
[(2,1)]
[(2,2),(2,0)]
(0,0)(1,0)(1,1)(2,1)(2,2)
Down Right Down Right
4.0
8
```

Figure 2: BFS Execution on JCONF03

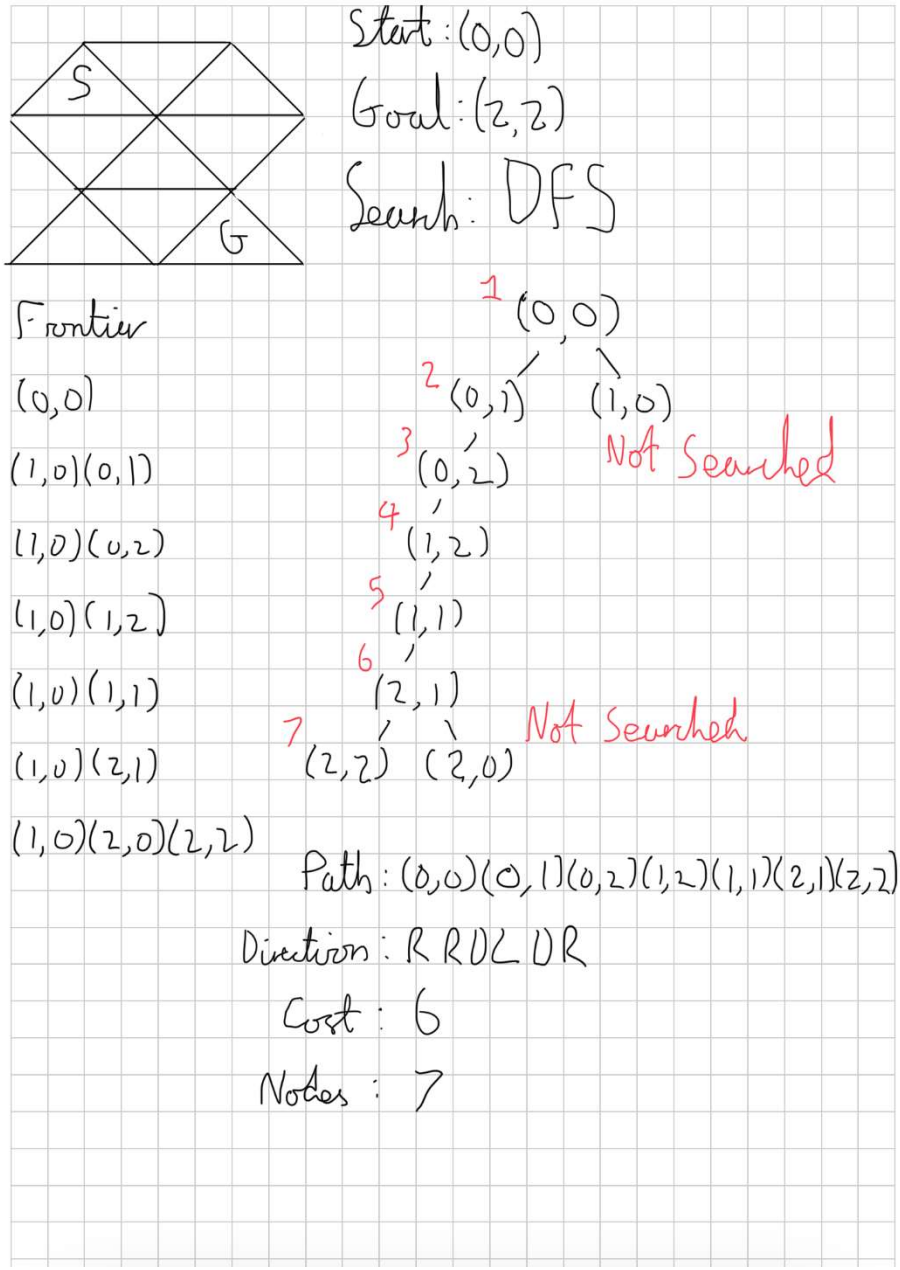

Figure 3: DFS Handwritten Implementation of JCONF03

```
[(0,0)]
[(1,0),(0,1)]
[(1,0),(0,2)]
[(1,0),(1,2)]
[(1,0),(1,1)]
[(1,0),(2,1)]
[(1,0),(2,0),(2,2)]
(0,0)(0,1)(0,2)(1,2)(1,1)(2,1)(2,2)
Right Right Down Left Down Right
6.0
7
```

Figure 4: DFS Execution on JCONF03

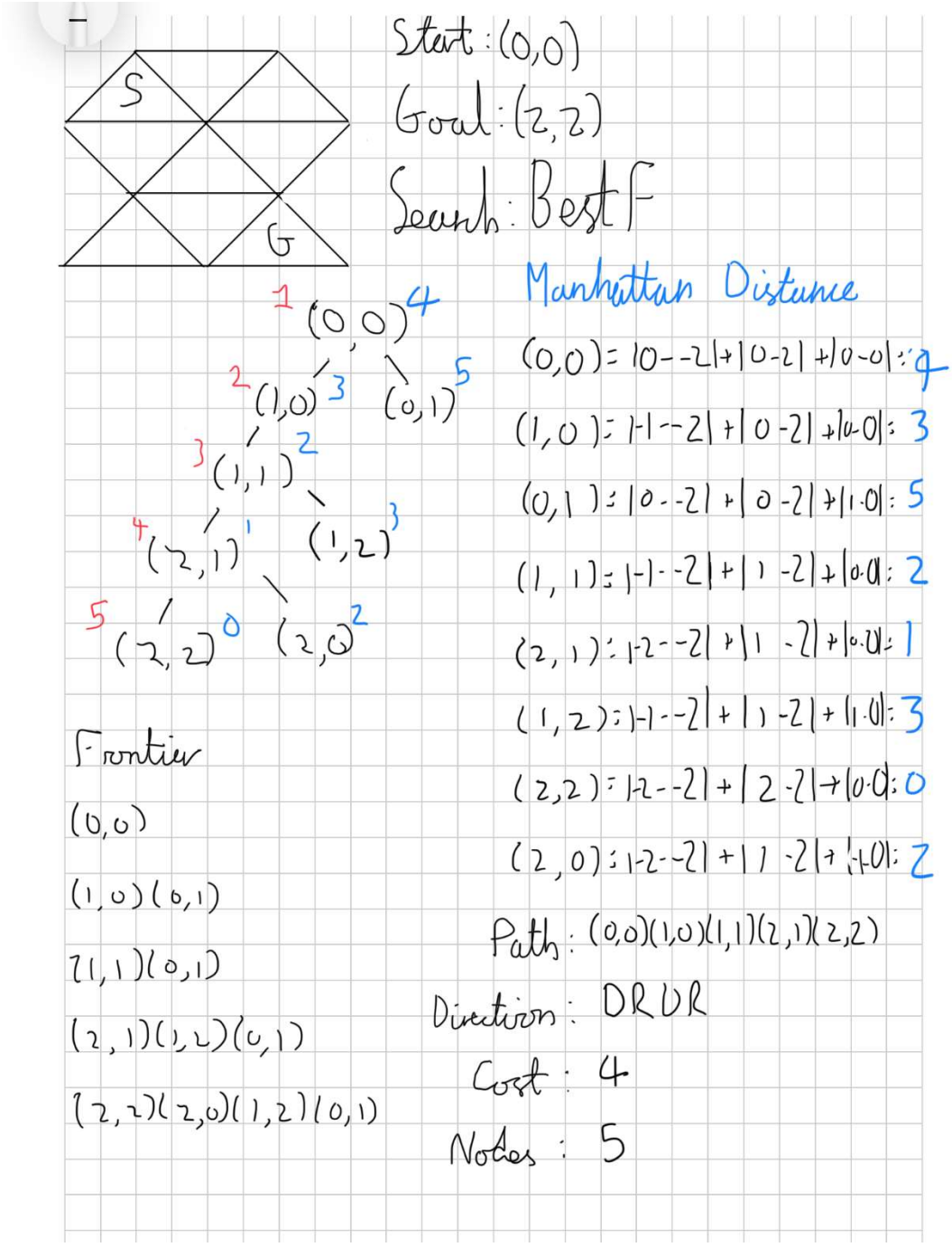

Figure 5: Best-First Handwritten Implementation of JCONF03

```
[(0,0):4.0]
[(1,0):3.0,(0,1):5.0]
[(1,1):2.0,(0,1):5.0]
[(2,1):1.0,(1,2):3.0,(0,1):5.0]
[(2,2):0.0,(2,0):2.0,(1,2):3.0,(0,1):5.0]
(0,0)(1,0)(1,1)(2,1)(2,2)
Down Right Down Right
4.0
5
```

Figure 6: Best-First Execution on JCONF03

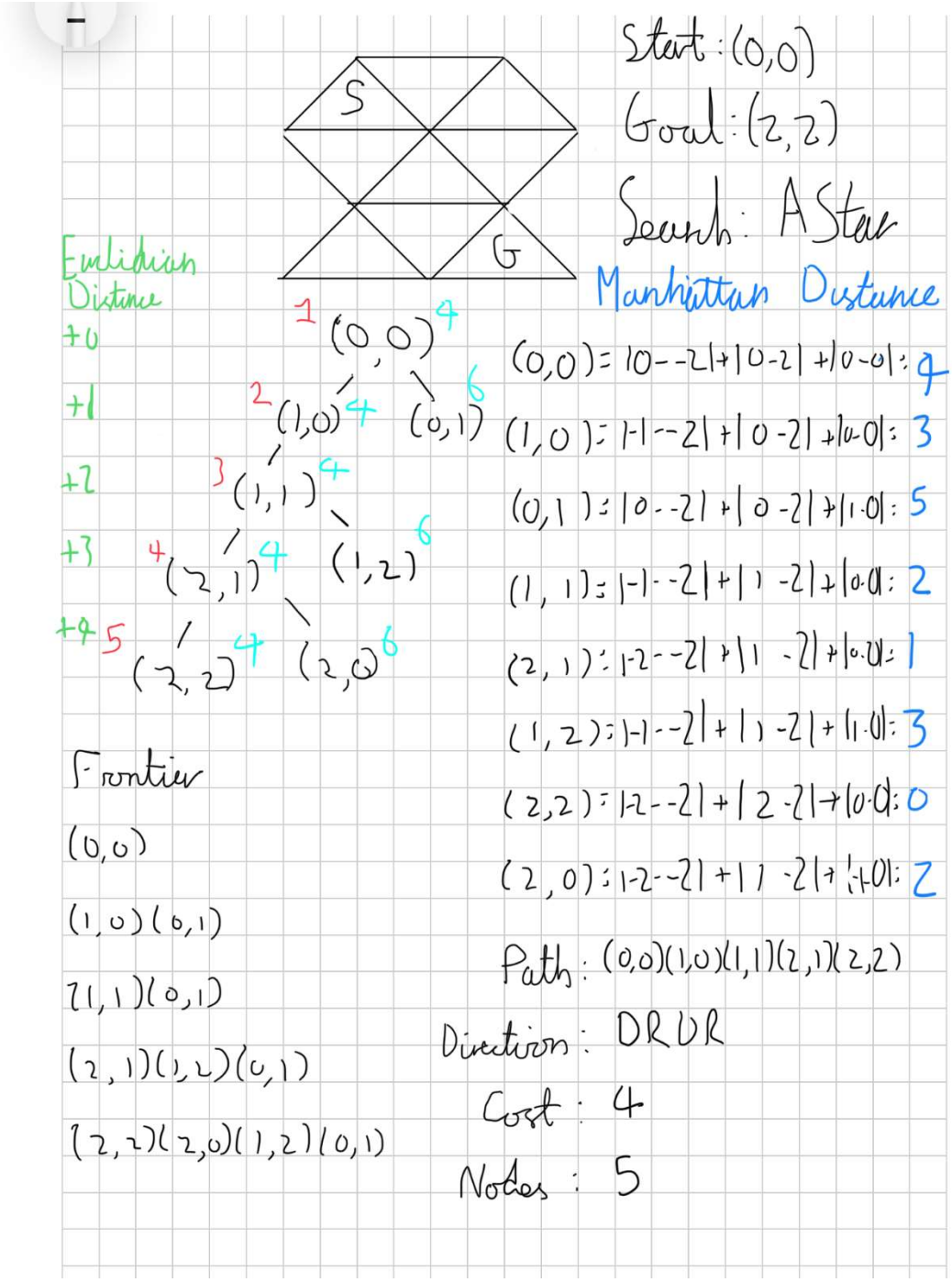

Figure 7: A Handwritten Implementation of JCONF03*

```
[(0,0):4.0]
[(1,0):4.0,(0,1):6.0]
[(1,1):4.0,(0,1):6.0]
[(2,1):4.0,(0,1):6.0,(1,2):6.0]
[(2,2):4.0,(0,1):6.0,(1,2):6.0,(2,0):6.0]
(0,0)(1,0)(1,1)(2,1)(2,2)
Down Right Down Right
4.0
5
```

Figure 8: A Execution on JCONF03*

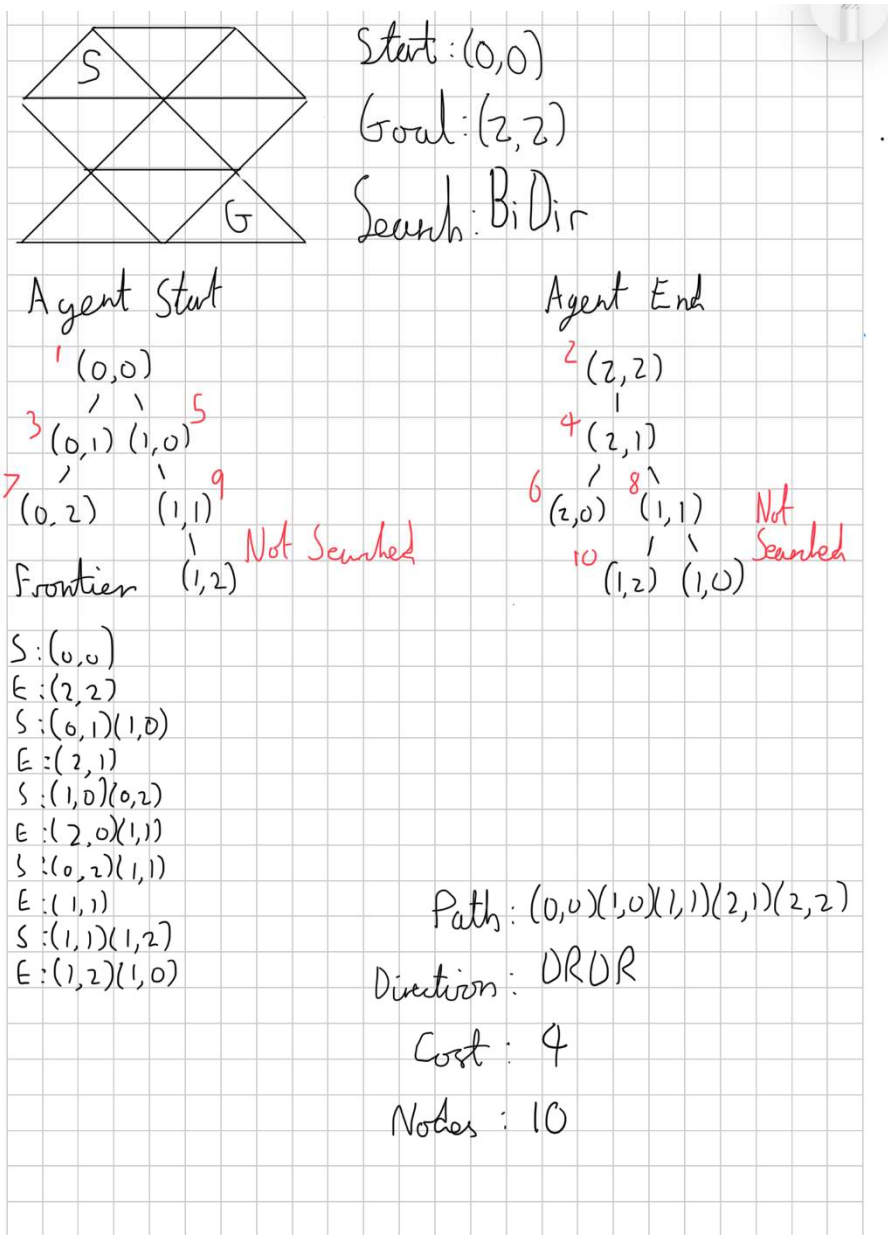

Figure 9: Bi-Directional Handwritten Implementation of JCONF03

```
Agent Start: [(0,0)]
Agent End: [(2,2)]
Agent From Start: [(0,1),(1,0)]
Agent From End: [(2,1)]
Agent From Start: [(1,0),(0,2)]
Agent From End: [(2,0),(1,1)]
Agent From Start: [(0,2),(1,1)]
Agent From End: [(1,1)]
Agent From Start: [(1,1),(1,2)]
Agent From End: [(1,2),(1,0)]
(0,0)(1,0)(1,1)(2,1)(2,2)
Down Right Down Right
4.0
10
```

Figure 10: Bi-Directional Execution on JCONF03

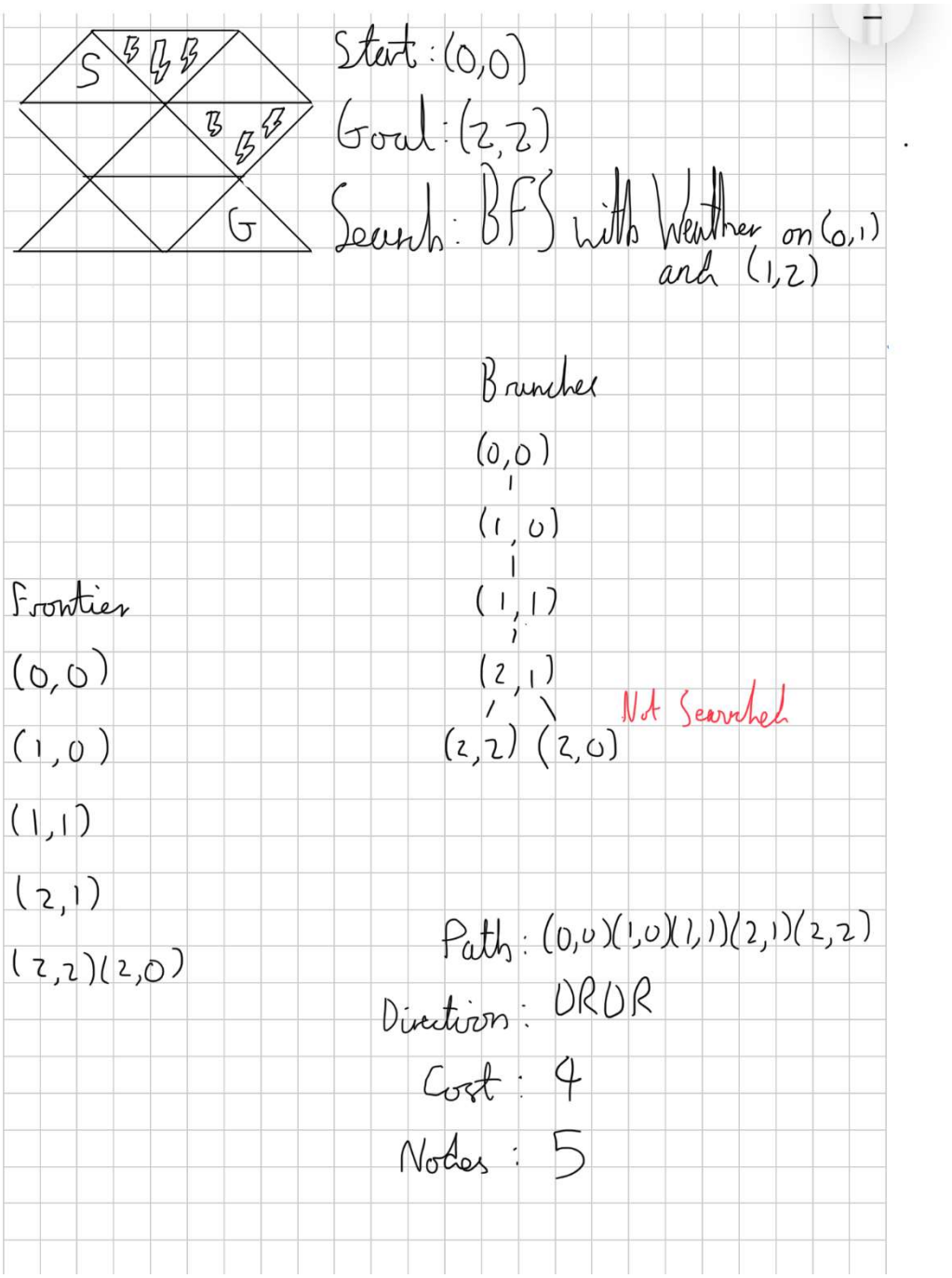

Figure 11: BFS with Weather Conditions on (0,1) & (1,2) Handwritten Implementation of JCONF03

```
[(0,0)]
[(1,0)]
[(1,1)]
[(2,1)]
[(2,2),(2,0)]
(0,0)(1,0)(1,1)(2,1)(2,2)
Down Right Down Right
4.0
5
```

Figure 12: BFS with Weather Conditions on (0,1) & (1,2) Execution on JCONF03

4. Evaluation

The path cost (performance) and the number of explored nodes (efficiency) of each implemented search algorithm was evaluated against all the provided configurations and placed into the table shown in Figure 13.

	Path Cost (Performance)					Number Of Nodes (Efficiency)				
Configuration	BFS	DFS	BestF	A*	BiDir	BFS	DFS	BestF	A*	BiDir
CONF0	10	14	10	10	10	34	16	11	11	32
CONF1	10	20	10	10	10	33	25	11	11	32
CONF2	10	14	10	10	10	33	16	11	11	32
CONF3	8	16	8	8	8	28	21	9	14	30
CONF4	8	12	8	8	8	32	28	9	15	30
CONF5	10	12	10	10	10	22	13	11	13	22
CONF6	10	10	12	10	10	20	12	13	16	22
CONF7	8	12	8	8	8	19	13	11	12	18
CONF8	8	14	8	8	8	17	16	9	9	18
CONF9	7	7	13	7	7	16	10	14	14	16
CONF10	fail	fail	fail	fail	fail	1	1	1	1	2
CONF11	fail	fail	fail	fail	fail	1	1	1	1	2
CONF12	6	6	6	6	6	11	8	8	8	10
CONF13	12	12	12	12	12	21	21	13	20	24
CONF14	11	11	11	11	11	23	20	20	19	24
CONF15	18	24	22	18	18	74	83	26	44	82
CONF16	5	19	5	5	5	14	20	6	7	12
CONF17	11	11	11	11	11	64	13	13	19	34
CONF18	11	41	11	11	11	55	62	14	35	44
CONF19	8	28	8	8	8	51	46	9	16	28
CONF20	16	24	20	16	16	50	28	22	36	48
CONF21	17	23	17	17	17	52	49	21	33	50
CONF22	16	16	22	16	16	42	46	27	35	46
CONF23	15	21	21	15	15	49	46	22	33	44
CONF24	17	19	21	17	17	46	23	23	36	50
No Solutions										
Solutions										
Known Issue										

Figure 13: A Table Showcasing All Possible Results from Each Combination of Configurations with Each Search Algorithm

There are five solutions that do not match the given tests, these are highlighted in Figure 13 as orange. This is caused by a known issue regarding the difference in implementation of the priority queue. The original model established by the lecturer utilises built-in Java PriorityQueue and a comparator, whilst the model used within this report utilises a LinkedList with a separate method called compareTo. Whilst this does not affect the performance or the efficiency, this issue highlights different coding practices.

For uninformed search, Figure 14 indicates that BFS and BiDir have an equal or greater performance than DFS. This is expected behaviour as BFS is optimal and DFS is not. DFS is only optimal when there is only one solution to the problem or when the first solution found is optimal, which is a very lucky scenario. However, Figure 15 illustrates that whilst BFS and BiDir are similar in terms of efficiency, DFS is shown to be consistently more efficient.

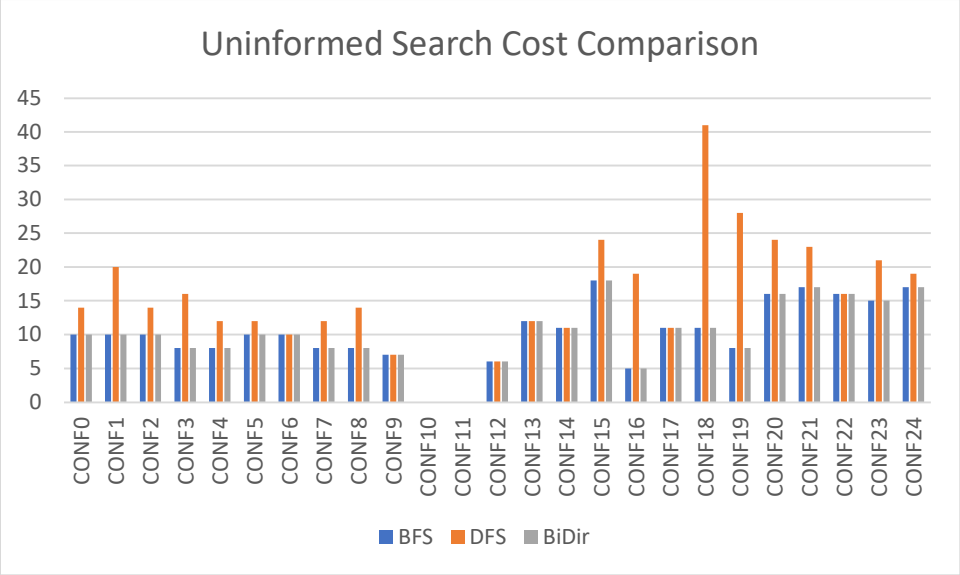

Figure 14: A Graph of All Uninformed Search Algorithms Performance Across Each Configuration

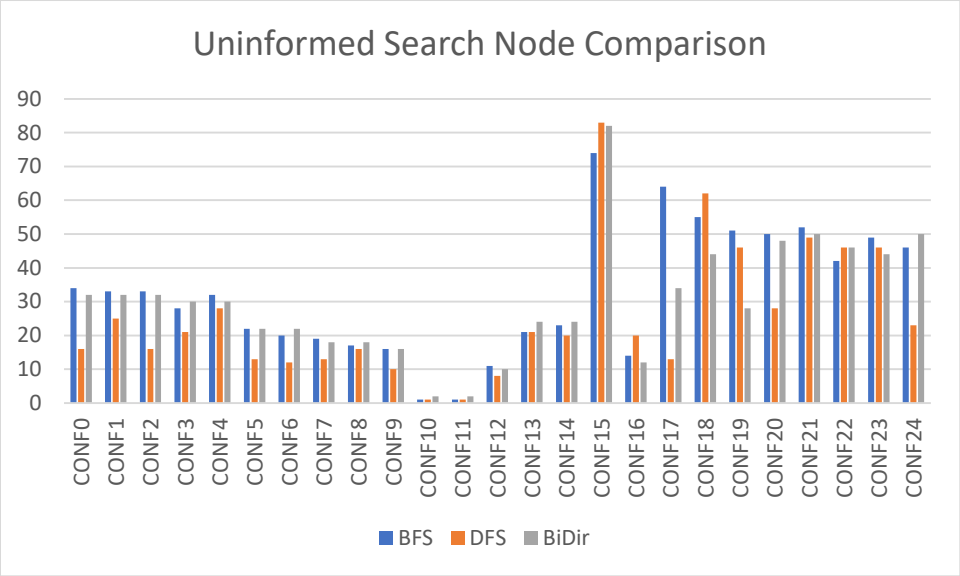

Figure 15: A Graph of All Uninformed Search Algorithms Efficiency Across Each Configuration

For informed search, Figure 16 shows that A* search has a slightly better performance on average when compared to Best-First. Though, they are equal in performance across most configurations. However, there is a trade-off between the performance and the efficiency in A* search as shown in Figure 17 as A* requires more nodes on average than Best-First.

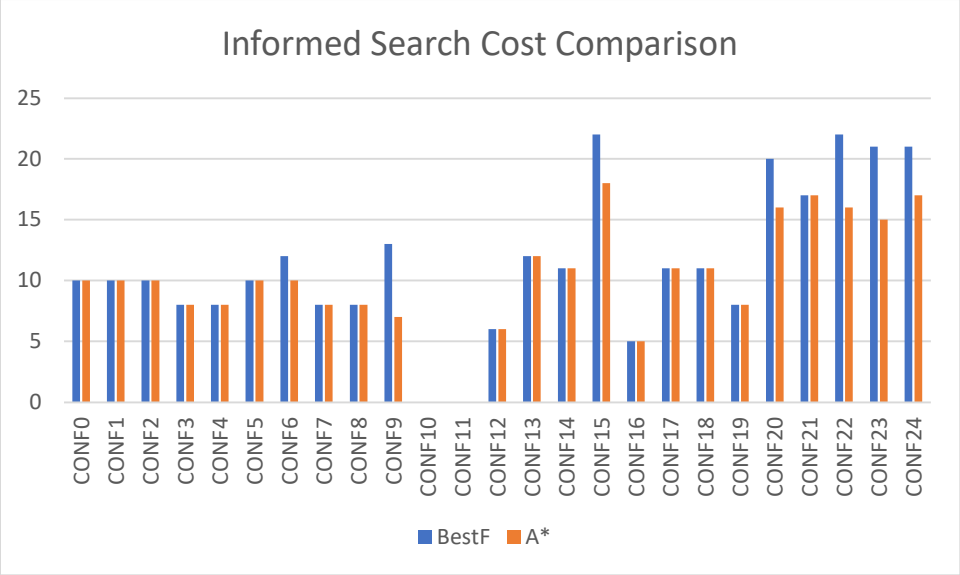

Figure 16: A Graph of All Informed Search Algorithms Performance Across Each Configuration

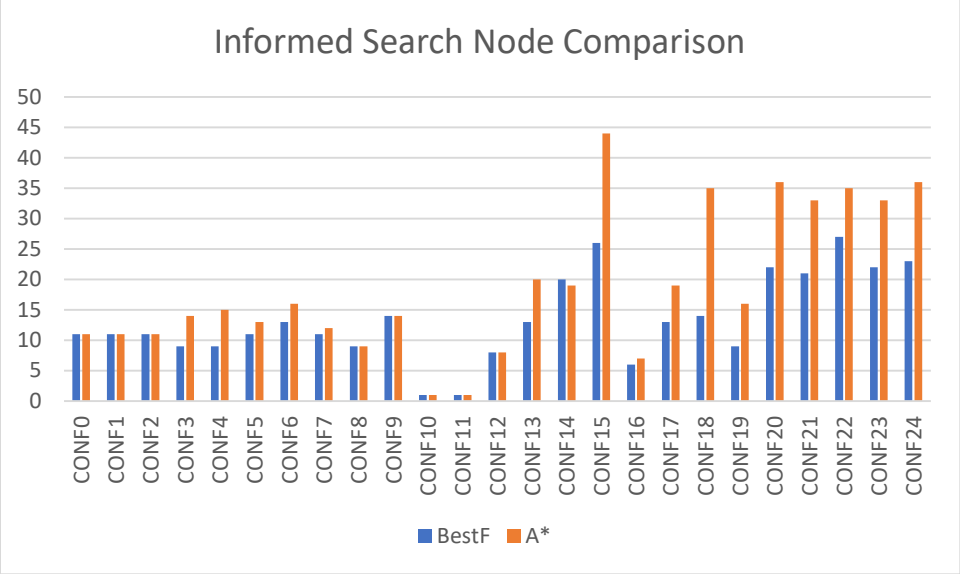

Figure 17: A Graph of All Informed Search Algorithms Efficiency Across Each Configuration

When comparing the performance of the uninformed and informed search, DFS is the worst search algorithm in terms of efficiency and performance as shown in Figures 18 and 19. The other algorithms are similar across the two metrics, the informed algorithms perform better overall when compared to the uninformed.

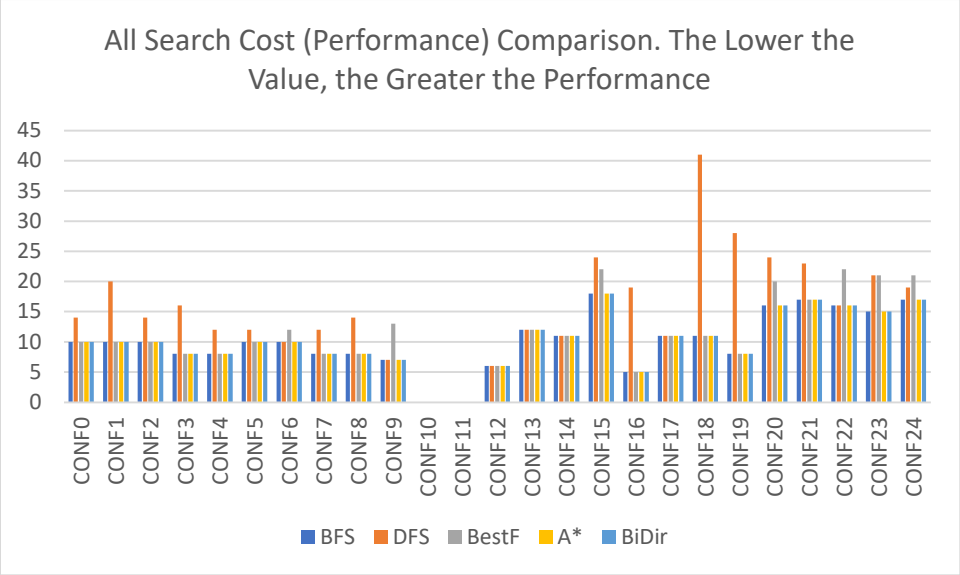

Figure 18: A Graph of All Search Algorithms Performance Across Each Configuration

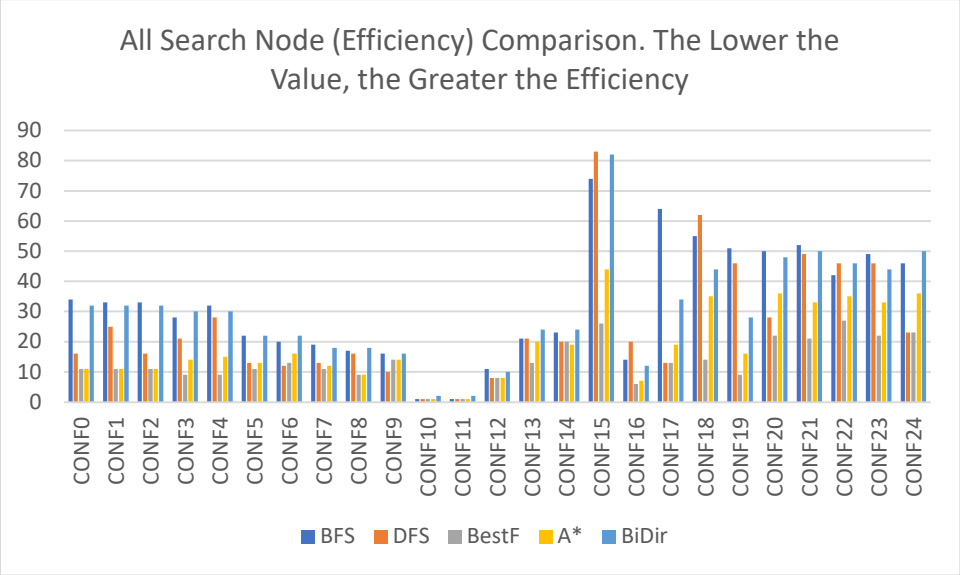

Figure 19: A Graph of All Search Algorithms Efficiency Across Each Configuration

In summary, BFS, BiDir and A* give the best performance, though A* is preferred as the efficiency is greater on average. Best-First is the most efficient search algorithm when compared to the rest. The inclination to use Best-First over A* would be based on prioritising efficiency. However, in a real-world scenario, performance is generally favoured over efficiency and therefore A* would be chosen as this has both an overall great performance and great efficiency.

5. References

Rahman, F. (2023) “CS5011 Introduction to Search,” *University of St Andrews*. St Andrews, January.

Russell, S.J. and Norvig, P. (2010) *Artificial Intelligence A Modern Approach*. Third. New Jersey: Pearson Education Inc (Prentice Hall Series).

6. Appendices

Appendix A: CS5011, Artificial Intelligence Practice GSA Model (Rahman, 2023)

```
function TREE-SEARCH(problem, frontier) returns a solution, or failure
initial_node ← MAKE-NODE(null, initial_state)
frontier ← INSERT(initial_node, frontier)
    loop do
    if IS-EMPTY(frontier) return failure
    nd ← REMOVE(index, frontier)
    if GOAL-TEST(nd.STATE, goal)
        return nd
    else
        frontier ← INSERT-ALL(EXPAND(nd, problem, frontier))
    end loop
end
```

Appendix B: Russell & Norvig’s GSA Model (2010)

```
function TREE-SEARCH(problem) returns a solution, or failure
  initialize the frontier using the initial state of problem
  loop do
    if the frontier is empty then return failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state then return the corresponding solution
    expand the chosen node, adding the resulting nodes to the frontier
```