

University  
of  
St Andrews

**First Practical-CS5011**

**A1 - Search - Flight route planner**

**Submitted by:**

**180030908**

**Submission Date: October 11, 2019**

## 1. Introduction:

In the given report, I have implemented several Search Algorithms to find a route for an aeroplane that flies from a starting point to a destination point. Below is the list of all the algorithms that I have implemented. Each search output, shows status of frontier at each stage of search, list of explored nodes, path followed by the search, number of steps taken by a particular search and execution time.

- **Un-Informed Search**

These search algorithms do not make use Heuristic function to decide the next move at any stage of the search. They do not have any information about distance between any two nodes. [1]

- **Best First Search (BFS)**

This is a blind search which search starts from a node, explore all its successor nodes on one depth and then moves to the next depth and explore all nodes present on that depth. This will continue until the goal node is discovered. [1]

- **Depth First Search (DFS)**

Here search starts from a node and explores one node at a depth and go as deep as possible down with that node. If goal node is found at the end of depth, this search backs up to the previous depth to try another un explored node. This continues until the goal node is found. [1]

- **Uniform Search**

This search makes use of path cost to decide which node to explore next. Path cost is the cost of all actions that is taken by search to travel from start of node to a node. So, on each this stage search calculates the path cost and then the cost with least path cost is explored. Here Heuristic function is  $f(n) = g(n)$  [2]

- **Informed Search**

These searches make use of Heuristic function to calculate their next move. They are more efficient and less expensive as compared to the uninformed search. [3]

- **Best First Search (BestF)**

This search expands the node whose calculated distance to the goal is the smallest. It does not the past cost into consideration like the cost from start to that current node.[3]

Here Heuristic function is  $f(n) = h(n)$

- **A\* Search**

This search is a combination of both Best First and Uniform Search and hence provides the efficient solution in terms of path cost. It finds the shortest path from start node to goal node by choosing that path which has the shortest path cost.[3]

Here Heuristic function is  $f(n) = g(n) + h(n)$

Where  $g(n)$  is the cost from start node to current node and  $h(n)$  is cost from current node to goal node.

- **Bidirectional Search**

This search is a modification of Breadth First Search where search occurs in two directions parallelly. One start from the start node and one start from the goal node. The solution is found when both searches intersect at a point. [4]

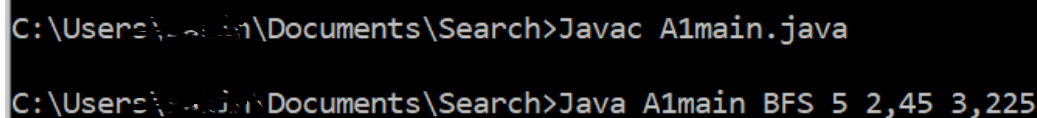
However, I was not able to print the path of this search and was not able to determine the number of steps as the program was throwing exception while I was trying to print the path.

### Instructions to Run the Program:

In command prompt, path of the folder should be given in which A1main.java file is present. After that following two command should be given:

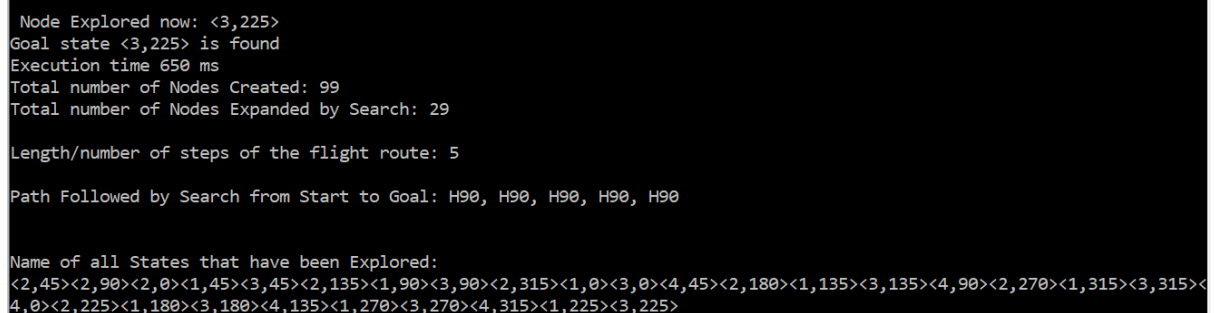
1. `Javac A1main.java`
2. `Java A1main <Search Name> <N><Node1 distance, Node2 angle><Node2 distance, Node2 angle>`

Search Name should be one from the following Search Names: BFS, DFS, AStar, BestF, BiDirect and Uniform. Otherwise, program will not execute and give an error message of invalid input. Also, when giving the details of coordinates, a comma (,) should be given between distance and angle for valid execution.



```
C:\Users\...Documents\Search>Javac A1main.java
C:\Users\...Documents\Search>Java A1main BFS 5 2,45 3,225
```

Figure 1: Commands to execute the program



```
-----
Node Explored now: <3,225>
Goal state <3,225> is found
Execution time 650 ms
Total number of Nodes Created: 99
Total number of Nodes Expanded by Search: 29

Length/number of steps of the flight route: 5

Path Followed by Search from Start to Goal: H90, H90, H90, H90, H90

Name of all States that have been Explored:
<2,45><2,90><2,0><1,45><3,45><2,135><1,90><3,90><2,315><1,0><3,0><4,45><2,180><1,135><3,135><4,90><2,270><1,315><3,315><4,0><2,225><1,180><3,180><4,135><1,270><3,270><4,315><1,225><3,225>
```

Figure 2: Output of each search have these headings

## 2. Design and Implementation

The first step for implementing these algorithms is designing of PEAS Model for the given problem. The PEAS model for the problem of route planning is given below:

**Problem:** Reaching destination point from Source point where plane can never fly or reach over the coordinates at the pole of the grid.

**Agent:** Software Program

**Performance:** This determines the level of success of a search algorithm. Here we have Number of steps taken by search to reach goal, Number of Nodes Created (which is directly proportional to execution time). Number of Nodes Explored (memory used), completeness (whether goal node is found or not).

**Environment:** Oedipus airspace which is represented by the help of two-dimensional coordinate system. In the program different data structures (Array List and Priority Queue) are used to store and represent values of this coordinate system.

**Actuators:** These are the actions performed by the agent. In our case, these includes going to four different directions, i.e. going north (360), south (180), east (90) and west (180). Other than that, several actions are performed by the agent on the environment which include adding, deleting and searching from data structures. It also includes calculating heuristic function and path costs for some algorithms.

**Sensors:** This is something perceived by the agent and on this basis, different actions are performed. Here path cost and heuristics function will be one of the sensors and when it returned for each node, program decides the next action which it has to take.

### Implementation Strategy:

Before starting with the given problem, I implemented different search algorithms on a simple tree as shown below to understand the steps taken by each search to find the goal node.

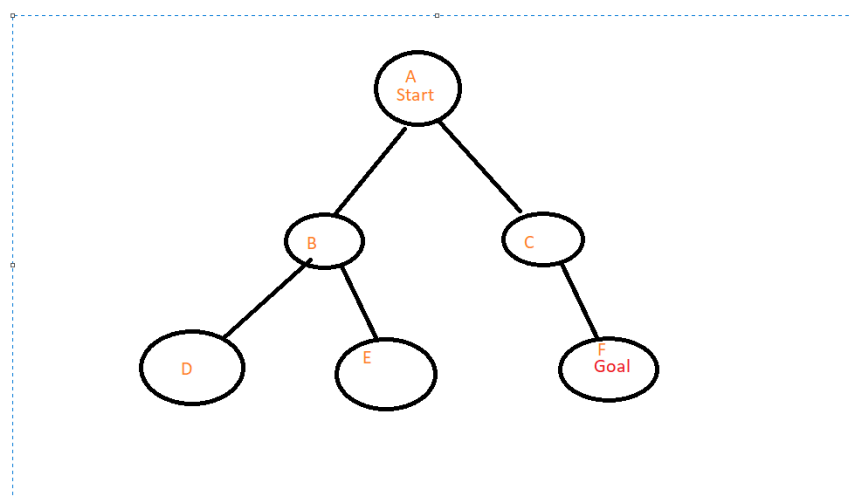


Figure 3: Simple Tree for Algorithm understanding

### Basic Algorithm:

Below image shows the underlying general search algorithm I used in all of my searches.

```
Add node to frontier
if frontier.size>0
if node!=goal node
    if node!=alreadyexplored
        -remove node from frontier
        -add node to exploredlist
        -add child nodes to frontier ( it differs in differnt algorithms)|
        -move to next node on frontier
repeat untill goal node is found
```

*Figure 4: Pseudocode of Search Algorithm*

### Creation of Node and Child nodes:

After that I used the same implementation on the given problem. I replaced the child nodes of this tree with four directions each node when explored, results in 4 child nodes, each in different directions i.e. H90, H180, H270, H360. The number of these four nodes of a node are dependent on the number of parallel that node is currently on. Let's say in a given input if total number of parallels are N and the node present on N-1 parallel will have 3 child nodes H90, H180, H360 as from that node, agent is not allowed to fly over the last parallel that is N. similarly if agent is on 01 parallel, it can not go further towards Magnetic North and hence here node will have child nodes in only these H90, H180, H270 directions.

During the creation of a node, its direction and distance from starting node as well are saved in memory. This distance is used to calculate heuristic function and path cost in case of informed and uniform search.

The stored direction is used at the time of printing the path from start node to goal node.

### Use of Data Structures:

For both BFS and DFS I used a simple Array List known as frontier in my code for adding and removing nodes instead of stack or queue. In case of BFS, I used **ArrayList.add(node)** to add the child nodes of the explored list at the back of the frontier so that the node that is added first in frontier is explored first. For DFS, I used the **ArrayList.add(0,node)** [5] to add the nodes in the frontier. This add nodes in the front of the frontier so that the node added last is explored first.

During implementation, the code for BFS and DFS is same except the way a node is inserted in a frontier as mentioned above.

This is the reason I used Array List for BFS and DFS so that I don't have to declare stack and queue separately and can use a single data structure for both searches.

For storing the explored nodes, I used ArrayList as well. I did this to have a check that agent doesn't expand a node which is already explored. In all searches, I used the same ArrayList for explored nodes.

For Bidirectional Search, I again used the Array List as frontier and used same logic of Breadth First Search but here I made two frontiers and two explored list. One for each direction of search that is one for start node to goal node and one from goal node to start node.

For Uniform, AStar, Best First Search, I used Priority Queue. In a priority queue, the elements are ordered according to their natural ordering, or by a Comparator provided at time when element is added in the queue [6]. I override this comparator in a way that it compares heuristics of each added node with the heuristic of the node that is already present in the queue and then places the node with the least heuristics in the front of the priority queue so that when poll() or peek() function of priority queue are called, the node with the least heuristic value is explored first.

Now for these searches, I used Priority Queue instead of any other data structure because Priority Queue provides built-in data sorting mechanism which was required for these searches, otherwise I had to develop a sorting mechanism myself if I used any other data structure.

### Mathematical Calculations:

In order to calculate the heuristic function (distance between two points), I used Euclidian Distance. "Euclidean distance or Euclidean metric is the "ordinary" straight-line distance between two points in Euclidean space" [7].

$$\text{dist}((x, y), (a, b)) = \sqrt{(x - a)^2 + (y - b)^2}$$

*Figure 5: Euclidian Distance Formula*

I used the above-mentioned formula for calculating distance between two coordinates at each node. I used **Math.sqrt** and **Math.cos** to perform the trigonometric requirement of the formula.

### Java Practices and Code Structure:

For node, I have simply used Java Class and then in its constructor I have passed the values of coordinates and parent node reference.

I have tried to implement the concept of data abstraction and encapsulation in my code. Code Abstraction can be seen from my main class where I have just added function calls and no other logic implementation. I have created separate classes for all main functionalities and then called their functions from different classes in order to provide maximum code re-use.

In some cases, I have to rewrite the code like for Bidirectional Search as the code for BFS/DFS can not be used for it even though the logic was same.

I have also used Getters and Setters [8] in my implementation which again supports data encapsulation. I have used it so for setting and getting directions and path cost of my child nodes at run time.

### 3. Evaluation

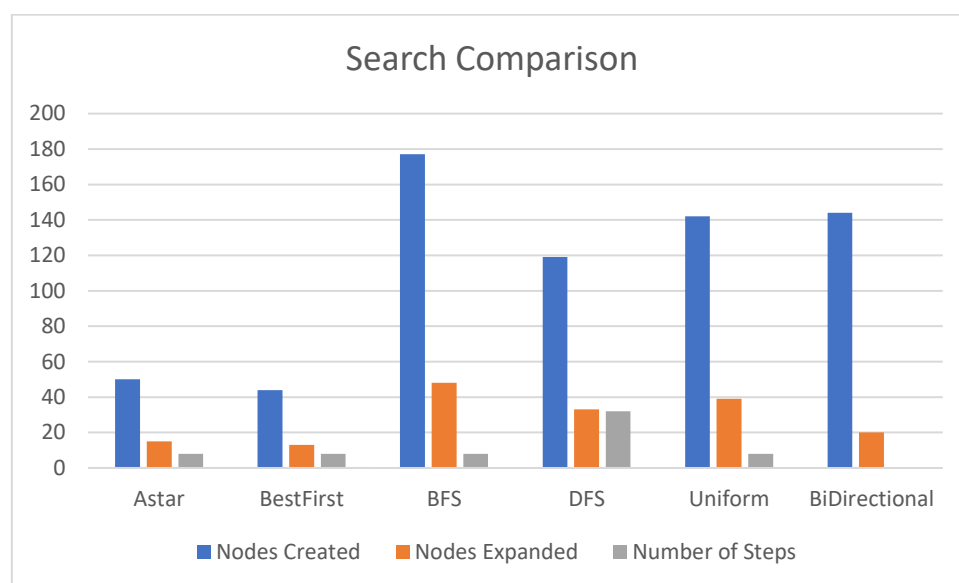


Figure 6: Comparison Between Different Search

- Based on the above shown graph and the tabular data in Test Summary part of this report, I have deduced following statements.
- All the searches were able to find the goal node but in most of the cases, BFS created the maximum number of nodes and hence used highest memory.
- Results of DFS depends on the side of the tree it starts the search from. It can sometimes perform better than the BFS and use less memory.
- Uniform Cost in most cases is better than BFS/DFS in terms of performance as it uses path cost instead of searching blindly.
- Bidirectional Search Takes more time than BFS but in most cases, expands less nodes than BFS.
- Performance of Best First and A\* is more or less is the same but depends on different scenarios, it can differ. Although A\* expands and creates more nodes than Best First in most of the cases but the overall path cost by A\* is always better in terms of total path cost of the search.

- In A\* and Best First sometimes two coordinates have same heuristic value. In this case agent expands the node which is added in front of the other in the frontier.

#### 4. Test Summary

Following tables shows the output result of different searches with different input data.

##### 1. N=5 S= (2,0) G= (2,135)

Search Name	Solution Found?	Execution Time (Milli Seconds)	Total Nodes Created	Total Nodes Explored	Total Number of Steps	Any Intersection Node?
BFS	Yes	78	44	13	3	Not Applicable
DFS	Yes	152	74	22	21	Not Applicable
A*	Yes	54	19	7	5	Not Applicable
BestF	Yes	33	16	6	5	Not Applicable
BiDirect	Yes	898	381	55	-	(4,270)
Uniform	Yes	62	28	10	5	Not Applicable



**2. N=5 S=(1,180) G=(4,180)**

Search Name	Solution Found?	Execution Time (Milli Seconds)	Total Nodes Created	Total Nodes Explored	Total Number of Steps	Any Intersection Node?
BFS	Yes	110	53	16	3	Not Applicable
DFS	Yes	31	11	4	3	Not Applicable
A*	Yes	31	11	4	3	Not Applicable
BestF	Yes	28	11	4	3	Not Applicable
BiDirect	Yes	1125	478	67	-	(4,0)
Uniform	Yes	293	109	32	3	Not Applicable

### 3. N=5 S= (1,315) G= (4,45)

Search Name	Solution Found?	Execution Time (Milli Seconds)	Total Nodes Created	Total Nodes Explored	Total Number of Steps	Any Intersection Node?
BFS	Yes	187	89	26	5	Not Applicable
DFS	Yes	212	95	28	27	Not Applicable
A*	Yes	56	28	9	5	Not Applicable
BestF	Yes	60	26	8	5	Not Applicable
BiDirect	Yes	732	352	50	-	(2,90)
Uniform	Yes	234	95	28	5	Not Applicable

### 5. N=5 S= (3,90) G= (0,0)

Search is not performed as 0,0 is invalid state for all types of Search. Program shows error message.

```
C:\Users\ashwin\Documents\Search>Java A1main BFS 5 3,90 0,0
Input node is invalid state and cannot be explored any further
```

### 6. N=8 S= (1,135) G= (5,315)

Search Name	Solution Found?	Execution Time (Milli Seconds)	Total Nodes Created	Total Nodes Explored	Total Number of Steps	Any Intersection Node?
BFS	Yes	411	177	48	8	Not Applicable
DFS	Yes	257	119	33	32	Not Applicable
A*	Yes	104	50	15	8	Not Applicable
BestF	Yes	99	44	13	8	Not Applicable
BiDirect	Yes	182	144	20	-	(5,315)
Uniform	Yes	390	142	39	8	Not Applicable

**7. N=8 S= (4,0) G= (7,90)**

Search Name	Solution Found?	Execution Time (Milli Seconds)	Total Nodes Created	Total Nodes Explored	Total Number of Steps	Any Intersection Node?
BFS	Yes	399	165	44	5	Not Applicable

DFS	Yes	421	168	46	45	Not Applicable
A*	Yes	109	44	13	11	Not Applicable
BestF	Yes	31	20	6	5	Not Applicable
BiDirect	Yes	1116	460	63	-	(6,180)
Uniform	Yes	625	190	52	11	Not Applicable

### 8. N=8 S=(6,270) G=(0,45)

Search is not performed as 0,45 state is an invalid for all types of Search. Program shows error message.

```
C:\Users\sahin\Documents\Search>Java A1main BFS 5 6,270 0,45
Input node is inavlid state and cannot be explored any further
C:\Users\sahin\Documents\Search>
```

### 9. N=10 S= (9,225) G= (2,45)

Search Name	Solution Found?	Execution Time (Milli Seconds)	Total Nodes Created	Total Nodes Explored	Total Number of Steps	Any Intersection Node?
BFS	Yes	687	259	69	11	Not Applicable

DFS	Yes	414	163	44	43	Not Applicable
A*	Yes	124	54	16	13	Not Applicable
BestF	Yes	93	46	14	13	Not Applicable
BiDirect	Yes	905	398	55	-	(2,45)
Uniform	Yes	333	127	35	13	Not Applicable

**10. N=10 S=(2,45) G=(9,225)**

Search Name	Solution Found?	Execution Time (Milli Seconds)	Total Nodes Created	Total Nodes Explored	Total Number of Steps	Any Intersection Node?
BFS	Yes	683	269	72	11	Not Applicable
DFS	Yes	392	164	44	43	Not Applicable
A*	Yes	174	67	19	13	Not Applicable

BestF	Yes	167	68	19	13	Not Applicable
BiDirect	Yes	5634	1203	160	-	(6,270)
Uniform	Yes	815	255	68	13	Not Applicable

**11. N=10 S=(7,270) G=(7,90)**

Search Name	Solution Found?	Execution Time (Milli Seconds)	Total Nodes Created	Total Nodes Explored	Total Number of Steps	Any Intersection Node?
BFS	Yes	64	93	25	4	Not Applicable
DFS	Yes	312	136	37	36	Not Applicable
A*	Yes	179	75	21	16	Not Applicable
BestF	Yes	141	60	17	16	Not Applicable
BiDirect	Yes	3413	960	125	-	(5,0)

Uniform	Yes	903	243	65	16	Not Applicable
---------	-----	-----	-----	----	----	----------------

## Screenshots of output of Best First and A\*

```
C:\Users\...i\Documents\Search>Java AImain BestF 5 2,25 3,225
Searching Informed Search with BestF Search
Frontier After First Node is Added:
<2,25> 2.6746491171583475
Node Explored now: <2,25>

Status of explored nodes:
<2,25>
child node on east added <2,70>
child node on west added <2,315>
child node on south added <1,25>
child node on north added <3,25>
Status of frontier after a node is expanded: <1,25> 2.6602394534999974 <3,25> 3.0381938466585527 <2,315> 4.286826727726471 <2,70> 4.341190642858258
-----

Node Explored now: <2,225>
Status of explored nodes:
<2,25><1,25><3,25><1,315><1,270><1,225><2,225>
child node on east added <2,270>
child node on west added <2,180>
child node on south added <1,225>
child node on north added <3,225>
Status of frontier after a node is expanded: <3,225> 0.0 <1,225> 2.0 <2,270> 2.587689342673738
<4,25> 3.6479440510830012 <2,315> 4.286826727726471 <3,70> 5.173867440943974 <1,315> 3.562083898
384 <2,70> 4.341190642858258
-----

Node Explored now: <3,225>
Node Explored: <3,225>
Goal state <3,225> is found

Execution time 146 ms
Total number of Nodes Created: 24
Total number of Nodes Expanded by Search: 8

Length/number of steps of the flight route: 6

Path Followed by Search from Start to Goal: H180, H180, H270, H270, H270, H360

Name of all States that have been Explored:
<2,25><1,25><3,25><1,315><1,270><1,225><2,225><3,225>
```

Figure 7: Output of Best First



```

C:\Users\...Documents\Search>Java A\main AStar 5 2,25 3,225
Searching Informed Search with AStar Search
Frontier After First Node is Added:
<2,25> 2.6746491171583475
Node Explored now: <2,25>

Status of explored nodes:
<2,25>
child node on east added <2,70>
child node on west added <2,315>
child node on south added <1,25>
child node on north added <3,25>
Status of frontier after a node is expanded: <1,25> 4.660239453499997 <3,25> 5.038193846658553 <2,315> 8.028788024087538 <2,70> 8.238586742542335
-----

Node Explored now: <3,225>
Node Explored: <3,225>
Goal state <3,225> is found

Execution time 222 ms
Total number of Nodes Created: 27
Total number of Nodes Expanded by Search: 9

Length/number of steps of the flight route: 6

Path Followed by Search from Start to Goal: H180, H180, H270, H270, H270, H360

Name of all States that have been Explored:
<2,25><1,25><3,25><1,315><1,0><1,270><1,225><2,225><3,225>
C:\Users\...Documents\Search>

```

Figure 8: Output of A\*

## 5. Bibliography

- [1] Gao, Y. and Japkowicz, N. (2009). *Advances in artificial intelligence*. Berlin: Springer.
- [2] Artificial Intelligence Stack Exchange. (2018). How does the uniform-cost search algorithm work? [online] Available at: <https://ai.stackexchange.com/questions/8755/how-does-the-uniform-cost-search-algorithm-work> [Accessed 03 Oct. 2019].
- [3] Stuart Russell and Peter Norvig. (1995) *Artificial Intelligence, A Modern Approach*, Prentice Hall.
- [4] Y. Dabachine, B. Bouikhalene and A. Balouki, "Bidirectional Search Algorithm for Airport Ground Movement," 2018 International Arab Conference on Information Technology (ACIT), Werdanye, Lebanon, 2018, pp. 1-9.
- [5] Stack Overflow. (n.d.). Add object to ArrayList at specified index. [online] Available at: <https://stackoverflow.com/questions/7384908/add-object-to-arraylist-at-specified-index> [Accessed 07 Oct. 2019].
- [6] GeeksforGeeks. (n.d.). Implement PriorityQueue through Comparator in Java - GeeksforGeeks. [online] Available at: <https://www.geeksforgeeks.org/implement-priorityqueue-comparator-java/> [Accessed 07 Oct. 2019].
- [7] Cut-the-knot.org. (n.d.). The Distance Formula. [online] Available at: <https://www.cut-the-knot.org/pythagoras/DistanceFormula.shtml> [Accessed 07 Oct. 2019].

[8] Fernandes, R. (2009). *Why use getters and setters/accessors?* [online] Stack Overflow. Available at: <https://stackoverflow.com/questions/1568091/why-use-getters-and-setters-accessors> [Accessed 09 Oct. 2019].