SEARCH ALGORITHMS IN AI APPLICATIONS REPORT

By Lip Chuan Sui

Msc in Artificial Intelligence

6G7V0011- Artificial Intelligence Principles

Lecturer: Mr Peng Wang

Assignment ID : 1CWK100

Student ID : 19072513

Contents

1.Introduction 2
2.Introduction to algorithms 2
3.Environment 4
4.Algorithm design 5
5.Result 7
6.Evaluation and Comparison 9
7.Discussion 9
8.Reference 10

1.Introduction

Nowadays, Artificial Intelligence (AI) has been implemented in nearly every aspect of life. From agriculture to gaming industry, from data analytic to robotic, AI has played an important role to solve predicaments and improve human lives. AI can observe real time event and make good decisions to serve its purpose. This requires a sophisticated path finding algorithm to analyze environment and move to its destination with the least amount of resource. This paper studies the part of AI decision making pipeline, the pathfinding algorithm.

An agent is the physical unit controlled by AI. When an agent is designed to move through real or virtual environment, it has to learn some things. First and foremost, agent must receive initial readings to learn about the state of the environment. Then, it must know where it is now (start node) and where should it be traveling (goal node). Thus, studying pros and cons of different pathfinding algorithms can improve efficiency of agents. To find out how each pathfinding algorithms work and the best pathfinding algorithm is the purpose of this paper.

Pathfinding algorithms are used to navigate a path from start point to goal point by connecting nodes. In the physical world, connected nodes have cost depends on the distance between them. Algorithms define every point in relation to neighbors and build a network. Then from starting point, algorithms navigate through the network to search for the goal point and return the path. Each algorithms work differently when deciding to explore the nodes. For example, the sequence of exploring nodes and the cost of path determine which path to take. This paper picks 4 common search algorithms which are Depth-First Search, Breadth-First Search, Uniform-Cost Search and A* search to test in Graph-Based and Tree-Based search parameters.

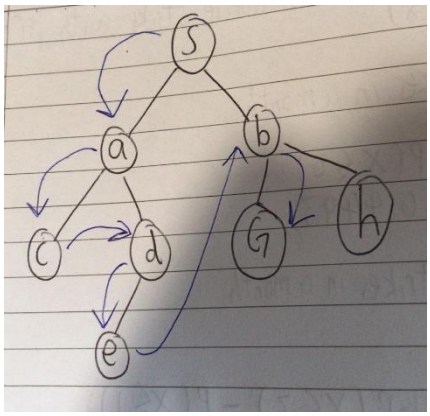
2.Introduction to algorithms

Each pathfinding algorithms follow basic steps when searching for goal.

1. From current node, check if current node is the goal node or not. If not, discover all neighboring nodes and add them to frontier. Algorithms take obstacle into consideration when exploring.
2. Algorithms mark current node as explored and pick a node from frontier. The way algorithm picks from frontier vary from each other's.
3. Choose a node and repeat Step 1.

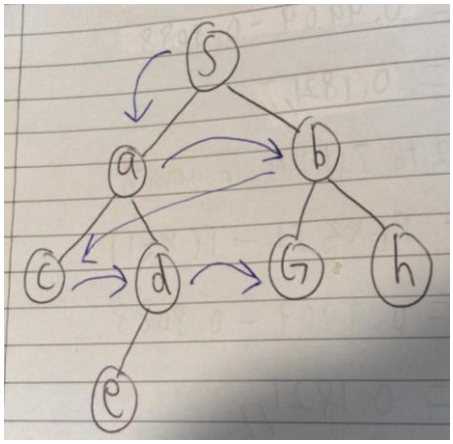
The difference between Tree-Based and Graph Based is Tree Based can reexplore node while Graph-Based can only explore once. A cycle can be formed in Graph Based while no cycle can be formed in Tree Based.

Depth first search:



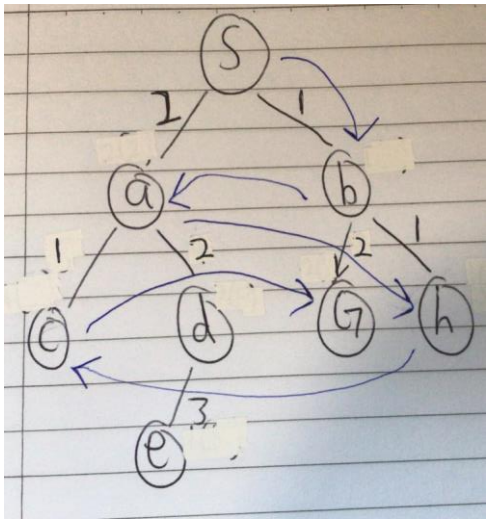
Current active nodes are chosen in order of newly discovered node. Algorithm searches down the branches of tree until the end before exploring another branch, Imagine a stack of paper, you add new piece of paper to the top and use the paper form the top first. It used LIFO principles.

Breadth First Search :



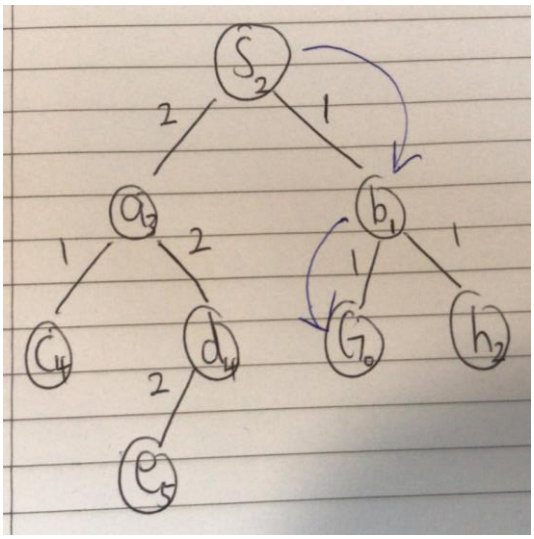
Current active nodes are chosen in order of first discovered node. Algorithm search all branches before navigating further down. Imagine people queuing, the person in the first is chosen first. It used FIFO principles.

Uniform Cost Search



Algorithm navigates based on cost of each journey it takes to travel from node to node. Current active node is chosen by prioritize lowest total path cost from starting point. Algorithm search the least cost needed when travelling.

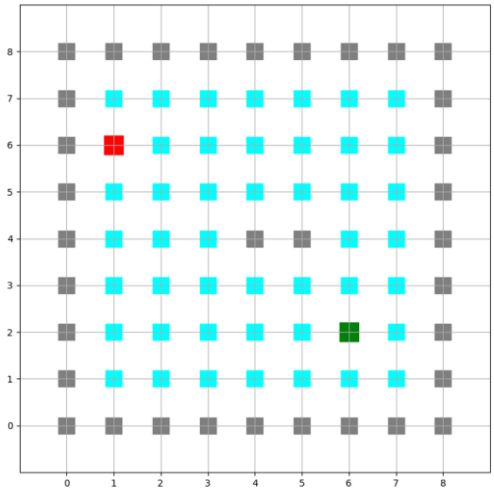
A* Search



This algorithm chooses active node based on total of path cost and heuristic value. Heuristic value is value given to represent how far node is from the goal node. The formula is f = g (cost to current node) + h (heuristic value on how close to goal)

3.Environment

In this portfolio, a 2d grid system with max width and height of 8 is used as the virtual environment. Every algorithm uses same environment parameter.



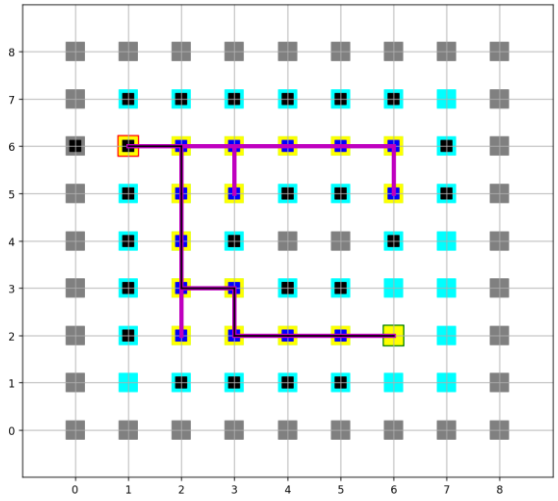


Fig 1 - Before running algorithm **Fig 1 - After running algorithm**

A robot is defined as a node which uses 1 point in the 2-grid system. Robot starts from the red node (start node) and search though blue node (free space) to arrive at green node (goal). When traveling and exploring, robot avoids gray node (obstacle) and navigate around it. There are 4 lines of gray node circling the environment to force robot only navigate within the area. The obstacle in the middle section serves to create two possible ways to navigate and compare reaction of each algorithm. After exploring and reaching the goal node, a black line will be generated to navigate from start to goal. This virtual environment create obstacle based on binary on excel file, such as 1 for obstacle and 0 for free space.

The action model of robot is that robot can only travel 1 node at a time. Each step cost 1 to travel. Newly explored coordinate can be remembered by adding or subtracting x and y.

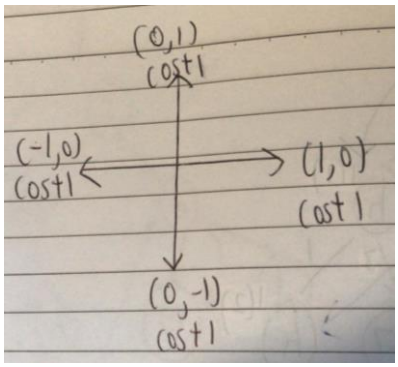


Fig3 – Action model of robot

Furthermore, coordinates are converted into index to store Node class. The formula is

Index = (y – min_y) * max width + (x – min_x) while min_y ,min_x is 0 and max width is 8.

64	65	66	67	68	69	70	71
56	57	58	59	60	61	62	63
48	49	50	51	52	53	54	55
40	41	42	43	44	45	46	47
32	33	34	35	36	37	38	39
24	25	26	27	28	29	30	31
16	17	18	19	20	21	22	23
8	9	10	11	12	13	14	15
0	1	2	3	4	5	6	7

Fig 4 - index table after converting from 2d grid.

Robot is required to travel from start node (6,1) to goal node (6,2)

4.Algorithm design

Depth-first search (Graph):

Search through 1 branch until the end before moving to another branch. Output returns the road path in list of coordinates, time used and memory used.

Depth-first search (Tree):

Search through 1 branch until the end before moving to another branch, building up binary tree from root. Output return road path generated by using preorder tree traversal, time used and memory used. Output also includes entire binary tree object. TreeNode object consist of left, right and data (current index).

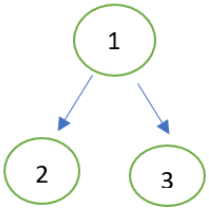


Fig 5- An example on how output will display tree node object in format below. Data stored is the index of coordinates.

(left: (left: None, right: none, 2) , right:(left: None, right: none, 3) , 1)

4 images of tree node are attached together in the folder for the result because current python file cannot display whole tree in python.

Breadth-first search (Graph):

Search through every branch before exploring further down. Output returns the road path in list of coordinates, time used and memory used.

Breadth-first search (Tree):

Search through every branch before exploring further down, building up binary tree from root. Output return road path generated by using tree traversal method, time used and memory used. Output also includes entire binary tree object.

Uniform-cost search (Graph):

Search through every branch by prioritizing lowest total path cost from starting point. Output returns the road path in list of coordinates, time used and memory used.

Uniform-cost search (Tree):

Search through every branch by prioritizing lowest total path cost from starting point, building up binary tree from root. Output return time used ,memory used and road path generated by using tree traversal method which prioritizing lowest total path cost. Output also includes entire binary tree object. TreeNode consist of left, right, data (index) and cost.

A* search (Graph):

Search through every branch by prioritizing lowest total path cost from starting point. Output returns the road path in list of coordinates, time used and memory used.

A* search (Tree):

Search through every branch by prioritizing lowest total of path cost and heuristic value, building up binary tree from root. Output return time used ,memory used and road path generated by using tree traversal method which prioritizing lowest total of path cost and heuristic value. Output also includes entire binary tree object. TreeNode consist of left, right, data (index) and cost ($f = g+ h$).

Each algorithm is evaluated by 5 aspects, which are time, memory storage, path, cost.

The lesser the time, memory, path and cost, the better and faster the algorithm. The higher the number of nodes explored, the less efficiency the algorithms.

Hypothesis: The algorithm which return the best result is A* search because A* search has the advantages as informed search algorithms by knowing the location goal node.

5.Result

Application is run a total of 10 times to get average result of time and storage.

	Time(s)	Path	Cost	Storage
T-BFS	31.74953269958496	[1, 6], [2, 6], [2, 5], [2, 4], [2, 3], [3, 3], [3, 2], [4, 2], [5, 2], [6, 2]	9	12.225085MB
T-DFS	11.702573776245117	[1, 6], [2, 6], [2, 5], [2, 4], [2, 3], [2, 2], [3, 2], [4, 2], [5, 2], [6, 2]	9	11.410027MB
T-UCS	13.305880069732666	[1, 6], [2, 6], [2, 5], [2, 4], [2, 3], [2, 2], [3, 2], [4, 2], [5, 2], [6, 2]	9	11.633009MB
T-A*	19.751052141189575	[1, 6], [2, 6], [2, 5], [2, 4], [2, 3], [3, 3], [3, 2], [4, 2], [5, 2], [6, 2]	9	12.284287MB
G-BFS	32.37385535240173s	[1, 6], [2, 6], [2, 5], [2, 4], [2, 3], [3, 3], [3, 2], [4, 2], [5, 2], [6, 2]	9	12.200205MB
G-DFS	11.781619787216187	[1, 6], [2, 6], [2, 5], [2, 4], [2, 3], [2, 2], [3, 2], [4, 2], [5, 2], [6, 2]	9	11.395413MB
G-UCS	13.393147945404053	[1, 6], [2, 6], [2, 5], [2, 4], [2, 3], [2, 2], [3, 2], [4, 2], [5, 2], [6, 2]	9	11.616013MB
G-A*	19.627899408340454	[1, 6], [2, 6], [2, 5], [2, 4], [2, 3], [3, 3], [3, 2], [4, 2], [5, 2], [6, 2]	9	12.293446MB

The black line is the path generated.

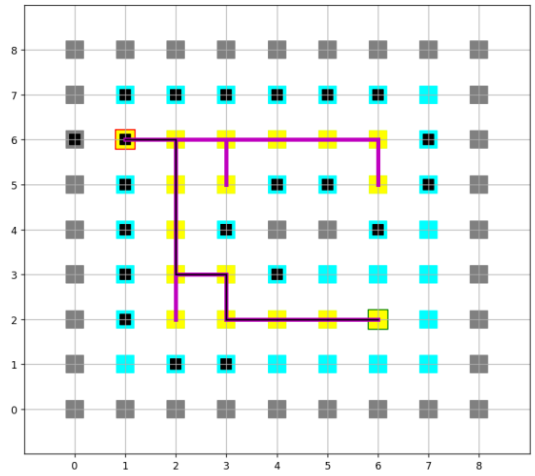


Fig-6 BFS Tree Search

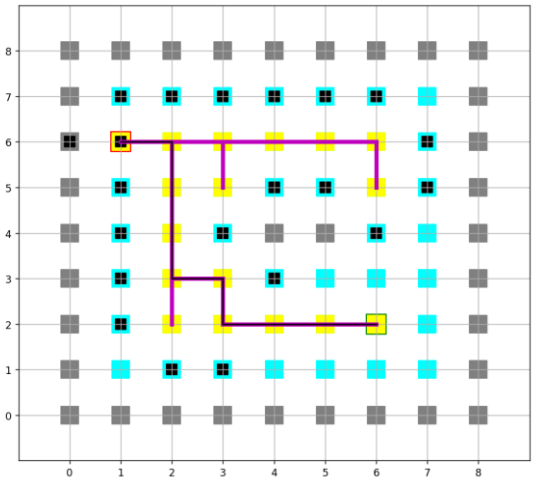


Fig -7 BFS Graph Search

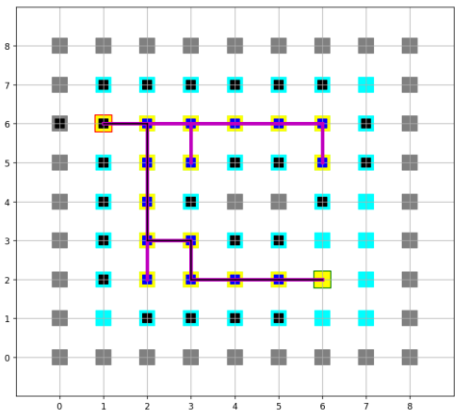


Fig -8 A* Tree Search

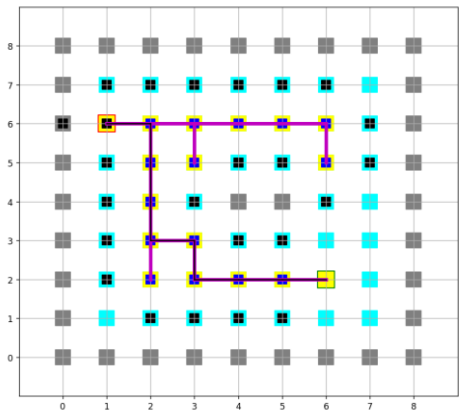


Fig -9 A* Graph Search

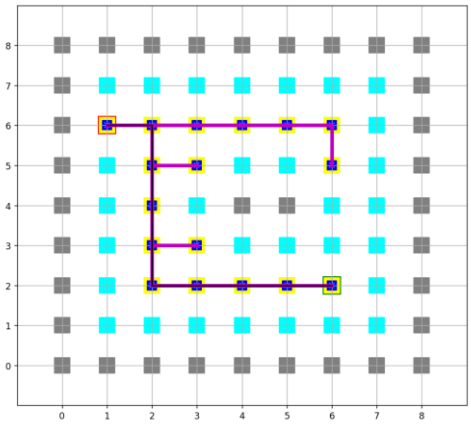


Fig-10 UCS Tree Search

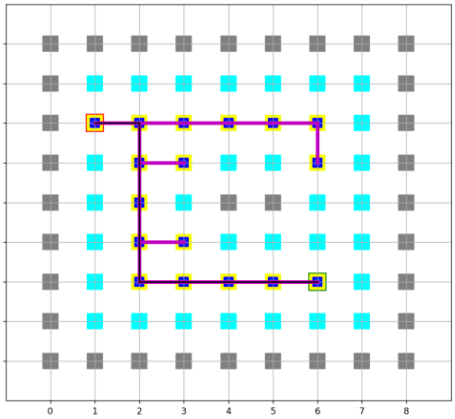


Fig-11 UCS Graph Search

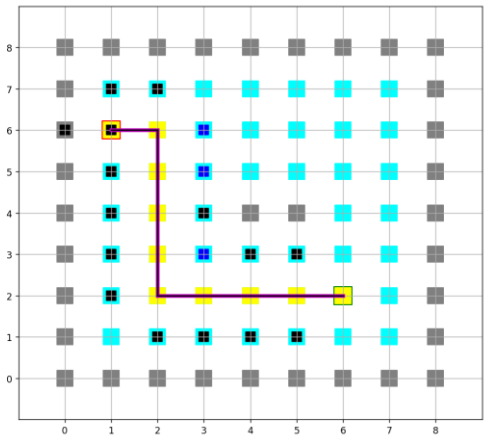


Fig-12 DFS Tree Search

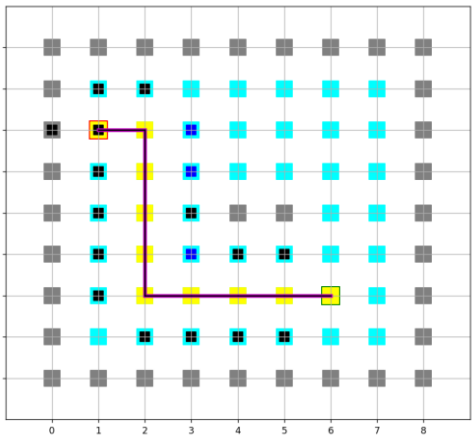


Fig-13 DFS Graph Search

6.Evaluation and Comparison

Based on the result, both tree and graph search of DFS arrived in goal node in the least amount of time and memory storage. This is due to DFS chooses the correct branch at the beginning and arrive at goal node. If goal node is on other side of branch, result would be longer.

However, tree and graph search for BFS used the most amount of time and memory to arrive at goal node which is 31.7s and 32.3s respectively. The time consumed is almost triple the time used by DFS. This is due to the distance between start node and goal node is much farther, BFS explore every child before exploring further down the branch.

Both tree and graph search for A* is the second highest, using 19s to complete task. This is due to fact that when obstacle spilt into two paths, each of the path return similar heuristic value, thus A* has to keep exploring branch before finding the least heuristic value.

UCS performs second best, with total time of 11s. UCS may perform better if the cost of each path varies. This portfolio used 1 cost for every node, thus limiting UCS strength.

Results from Tree and Graph search return similar results. In conclusion, both tree and graph search for DFS performs the best among the 4 algorithms.

7.Discussion

While this experiment shows that DFS performs the best, we must still choose suitable algorithm based on the situation because every algorithm has pros and cons. DFS depends on luck because if the goal node is on the last branch, DFS would have travel all previous branch just to arrive at goal. This would be worst case scenario. The time and space complexity depends highly on whether branch is correct. While there is no specification in prioritizing which branch. Thus, DFS is best used in solving maze with 1 solution and topological sorting.

Meanwhile, experiment shows that BFS used most amount of time to arrive in goal. This is due to BFS strength lies in finding neighbor node. The time and space complexity increases directly proportional to distance between start node and goal node. Thus, BFS is suitable in GPS navigation systems to find all neighboring locations.

Moreover, UCS and A* are expected to perform better. This is due to cost is limited to 1 for every step taken. If cost varies, UCS and A* could perform better. Besides, the available path from start to goal in this experiment presents similar cost. Thus, UCS and A* couldn’t perform well. If goal node is located on the bottom side of start node, A* could directly go there without any extra path. A* search is best used when goal node is known to robot.

These experiments run in a 2d static grid system. If the environment varies every second, algorithm would be to adapt to changes. Sensor would be needed to receive input of changing environment and constantly update based on latest changes. Tree and graph search should be considered to prevent infinite loop occurs. Thanks.

8.Reference

1. *Tree Traversals (Inorder, Preorder and Postorder)* ,30 Nov 2021

https://www.geeksforgeeks.org/tree-traversals-inorder-preorder-and-postorder/

2. *Level Order Binary Tree Traversal* ,30 Dec 2021

https://www.geeksforgeeks.org/level-order-tree-traversal/

4. *Finding path to node using Breadth First Search*,18 Dec 2019

https://stackoverflow.com/questions/64394909/finding-path-to-node-using-breadth-first-search

5. *A* Search Algorithm* , 24 Dec 2021

https://www.geeksforgeeks.org/a-search-algorithm/

6. How to open New Fragment From Fragment by Button Click | Android Studio Fragments Complete Tutorial, 24 April 2019

https://www.youtube.com/watch?v=GIiDNiebFt8

7. *Lecture 5: Search 1 - Dynamic Programming, Uniform Cost Search | Stanford CS221: AI* , 8 Jan 2020

https://www.youtube.com/watch?v=aIsgJJYrlXk&ab_channel=stanfordonline

8. *Applications of Breadth First Traversal*, 18 Dec 2019

https://www.geeksforgeeks.org/applications-of-breadth-first-traversal/