AU 332 Artificial Intelligence: Principles and Techniques

By: Zhengbao He(517030910157)

HW#: 1

September 23, 2019

I. INTRODUCTION

A. Purpose

In homework 1 we are going to do exercises about:

- Depth-first search algorithm
- Breadth-first search algorithm
- Uniform cost search algorithm
- A* algorithm

The first three algotithms are uninformed-search strategies. The UNINFORMED term means that the strategies have no additional information about states beyond that provided in the problem definition. All they can do is generate successors and distinguish a goal state from a non-goal state. These search strategies are distinguished by the order in which nodes are expanded.

A* algorithm is an informed-search strategy, which has a heuristic fucntion. It uses problem-specific knowledge beyond the definition of the problem itself—can find solutions more efficiently than can an uninformed strategy.

All of the algorithms can search from a start node to a goal node and return the path on a tree or a connected graph. And the exercises in homework 1 are all based on graphs.

B. Equipment

I finished the homework with:

- A laptop with Windows 10
- Anaconda 3
- Pycharm2019
- TexLive

II. QUESTIONS AND ANSWERS

This section will consist of the questions in homework 1 and corresponding answers.

A. Graph Traversal

Question:

In Figure 1, the start node is A. Please draw a expanding tree stricture for the graph using depth-first search algorithm and the breadth-first search algorithm. If the graph has n nodes and the maximum degree for each node is d, what is the complexity of BFS and DFS?

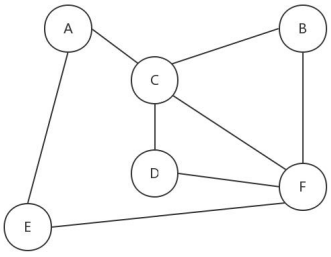
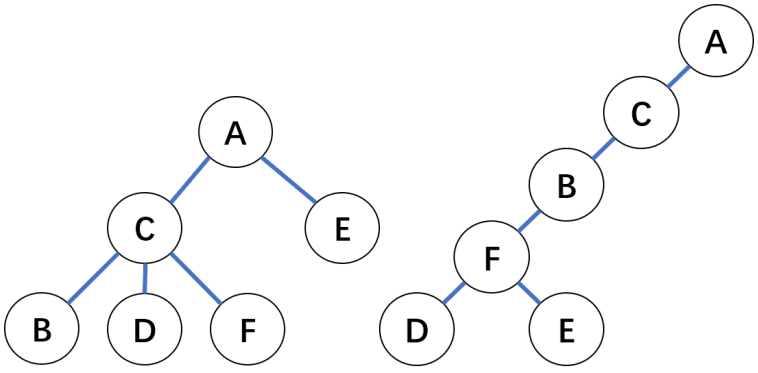


FIG. 1: Solve DFS and BFS based on the given graph. Node A is the start node

Answer:

The expanding trees of BFS and DFS are below(provided that the nodes are expanded alphabetically):



(a) BFS (b) DFS

FIG. 2: The expanding trees of BFS and DFS

If a graph has n nodes and the maximum degree for each node is d, in the case of only considering fringe:
the time complexity of BFS is $O(n)$ and the space complexity of BFS is $O(\frac{n}{2})$=$O(n)$;
the time complexity of DFS is $O(n)$ and the space complexity of DFS is $O(\log_d n)$.
Considering the time to query neighbors in the adjacency list and the space to store the closed set:
the time complexity of BFS is $O(\frac{nd}{2}+n) = O(nd)$ and the space complexity of BFS is $O(n)$;
the time complexity of DFS is $O(\frac{nd}{2}+n) = O(nd)$ and the space complexity of DFS is $O(n)$.

B. Uniform Cost Search Algorithm

Question:

In Figure 3, the start node is A and the goal node is E, calculate the shortest path from node A to E using UCS method. Write step by step update of the fringe list and closed list.

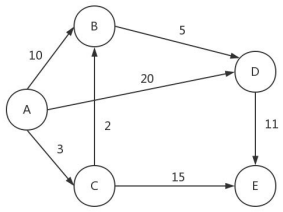


FIG. 3: Uniform cost graph search problem

Answer:

Use a pair of parentheses to distinguish the basic elements in the fringe. The first letter in parentheses represents the parent, the second letter represents the child which is the node in the fringe to be expanded, and the number after it represents its cost. An example is (Parent-Child,Cost of child). The progress is below:

Step	Expanding node	Fringe	Closed set
1	A	{(A-B,10),(A-C,3),(A-D,20)}	{}
2	C	{(C-B,5),(A-D,20),(C-E,18)}	{A}
3	B	{(B-D,10),(C-E,18)}	{A,C}
4	D	{(C-E,18)}	{A,C,B}
5	E	{}	{A,C,B,D}

C. A* algorithm

Question:

Write out the complete path finnding process(fringe list and close list) from the green grid to the red grid using A* algorithm (blue grids are obstacles). The green grid is the start location and the red grid is the goal location. Write out your choice of of h heuristic function. use the row and column number as reference to the location of the cell. The action space are up, down, left and right. The cost for each action is 1.

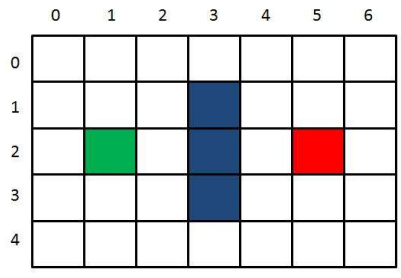


FIG. 4: Use A* to solve for the maze

Answer:

The table of the search progress is below:

STEP	EXPANDING NODE	FRINGE			CLOSED SET
		NODE	H+F	COST	
1	(2,1)	(1,1)	8.0	1	{}
		(2,2)	8.0	1	
		(3,1)	8.0	1	
		(2,0)	10.0	1	
2	(1,1)	(0,1)	8.0	2	(2,1)
		(1,2)	8.0	2	
		(2,2)	8.0	1	
		(3,1)	8.0	1	
		(1,0)	10.0	2	
		(2,0)	10.0	1	
3	(0,1)	(0,2)	8.0	3	(2,1) (1,1)
		(1,2)	8.0	2	
		(2,2)	8.0	1	
		(3,1)	8.0	1	
		(0,0)	10.0	3	
		(1,0)	10.0	2	
		(2,0)	10.0	1	
4	(0,2)	(0,3)	8	4	(2,1) (1,1) (0,1)
		(1,2)	8	2	
		(2,2)	8	1	
		(3,1)	8	1	
		(0,0)	10	3	
		(1,0)	10	2	
		(2,0)	10	1	
5	(0,3)	(0,4)	8	5	(2,1) (1,1) (0,1) (0,2)
		(1,2)	8	2	
		(2,2)	8	1	
		(3,1)	8	1	
		(0,0)	10	3	
		(1,0)	10	2	
		(2,0)	10	1	
6	(0,4)	(0,5)	8	6	(2,1) (1,1) (0,1) (0,2) (0,3)
		(1,4)	8	6	
		(1,2)	8	2	
		(2,2)	8	1	
		(3,1)	8	1	
		(0,0)	10	3	
		(1,0)	10	2	
		(2,0)	10	1	

Continued:

STEP	EXPANDING NODE	FRINGE			CLOSED SET
		NODE	H+F	COST	
7	(0,5)	(1,5)	8	7	(2,1) (1,1) (0,1) (0,2) (0,3) (0,4)
		(1,4)	8	6	
		(1,2)	8	2	
		(2,2)	8	1	
		(3,1)	8	1	
		(0,6)	10	7	
		(0,0)	10	3	
		(1,0)	10	2	
		(2,0)	10	1	
8	(1,5)	(2,5)	8	8	(2,1) (1,1) (0,1) (0,2) (0,3) (0,4) (0,5)
		(1,4)	8	6	
		(1,2)	8	2	
		(2,2)	8	1	
		(3,1)	8	1	
		(1,4)	10	8	
		(1,6)	10	8	
		(0,6)	10	7	
		(0,0)	10	3	
		(1,0)	10	2	
		(2,0)	10	1	

And the path is shown in the form of a table as follows:

	0	**1**	**2**	**3**	**4**	**5**	**6**
0		**2**	**3**	**4**	**5**	**6**	
1		**1**		**WALL**		**7**	
2		**0**		**WALL**		**8**	
3				**WALL**			
4							

There are two points to note about this specific algorithm used in the solution:

First, because of the existence of the 'wall', the Manhattan distance does not work well as a heuristic function, although it satisfies consistency. In this question, I use another heuristic function, specifically:

when the current node is in the upper part of the graph(its row value is less than or equal to 2), its h *value is The sum of the Manhattan distances of it to node (0,3) and node (0,3) to the goal node. Correspondingly, when it is in the lower half of the graph, its* h *value is The sum of the Manhattan distances of it to node (4,3) and node (4,3) to the goal node.*

It is easy to prove that this heuristic function also satisfies the consistency.

Second, when two nodes have the same g value($g = f + h$,f represents the cost of a node), it's obvious that you should first expand the node with a larger cost, that is, a node with a smaller h value. Because the smaller value of h(the larger cost) means this node may be closer to the goal node.

Based on the above two points, I solved this problem using the A* algorithm.

D. Programming Assignment

Please check the code in *BFSvsDFS.py, UniformCostSearch.py* and *AStarSearch.py.*

It is worth noting that in the code editing area of the search algorithm in each file, I have called the function in *E.1* to check if the goal and start state are valid nodes in the graph. So the output may be a little different.You can learn more in *E.1.*

E. Extra credit

Question:

For the programming assignment part, extra credits will be given if you completed the following cases.

1. Check if the goal and start state are valid nodes in the graph, return error handling message.
2. Check whether the graph satisfies the consistency of heuristics.

For E.1, the function is below:

```
def check_valid(graph, start, goal):
  print('Begin to check the validity of the start node %s and the goal node %s in the graph:' % (start,
       goal))
  flag_of_start = False # True means that start is a valid node, while False means not.
  flag_of_goal = False # True means that goal is a valid node, while False means not.
  for i in graph.edges.keys():
    if i==start or start in graph.edges[i]:
      flag_of_start = True
    if i==goal or goal in graph.edges[i]:
      flag_of_goal = True
    if flag_of_start and flag_of_goal:
      break
  print("\tThe start node is%s a valid node." % '' if flag_of_start else ' not')
  print("\tThe goal node is%s a valid node." % '' if flag_of_goal else ' not')
  return flag_of_start, flag_of_goal
```

The first return value of the function represents whether the start state is a valid node, and the second return value represents whether the goal state is a valid node.

I also call the function in the code editing area of search algorithms *BFSvsDFS.py, UniformCostSearch.py* and *AStarSearch.py.*

For E.2, I also wrote a function as following:

```
  def check_graphs_consistency_of_heuristics(graph,goal):
      not_satisfied=[]
      for i in graph.edges.keys():
          for j in graph.edges[i]:
              if (heuristic(graph,i,goal)-heuristic(graph,j,goal))>graph.get_cost(i,j):
                not_satisfied.append((i,j))
      if not_satisfied == []:
          print('The graph satisfies the consistency of heuristics.')
          return True
      else:
          print('The graph doesn\'t satisfi y the consistency of heuristics because:')
          for i,j in not_satisfied:
              print('The h value of %s is %2.4f and the h value of %s is %2.4f but the cost from %s to %s
                   is %d'%
              (i,heuristic(graph,i,goal),j,heuristic(graph,j,goal),i,j,graph.get_cost(i,j)) )
        return False
```

The return value and the printed message both indicate whether the graph satisfies the consistency of heuristics. I called this function in *AStarSearch.py*, and the printed message is below:

```
'''
The small graph doesn't satisfiy the consistency of heuristics because:
  The h value of B is 7.2111 and the h value of D is 2.8284 but the cost from B to D is 4
  The h value of C is 8.0000 and the h value of A is 5.6569 but the cost from C to A is 2
The large graph doesn't satisfiy the consistency of heuristics because:
  The h value of S is 10.0000 and the h value of B is 7.8102 but the cost from S to B is 2
  The h value of B is 7.8102 and the h value of H is 5.0000 but the cost from B to H is 1
  The h value of C is 8.0000 and the h value of L is 5.0990 but the cost from C to L is 2
  The h value of G is 3.1623 and the h value of E is 0.0000 but the cost from G to E is 2
'''
```

The results show that the heuristic function I designed does not satisfy the consistency in both figures. This is not only related to functions, but also to the nature of the graph. It is possible that the graphs in *AStarSearch.py* are not in the European space, so they do not satisfy the consistency. If we multiply the Euclidean distance in the heuristic function by a sufficiently small constant, we can make these graphs consistent. But when this constant is too small, the heuristic function loses its meaning, and the A* algorithm becomes the UCS algorithm. Therefore, it is very important to design different heuristic functions for different graphs.

III. DISCUSSION & CONCLUSION

After studying these search algorithms, it's obvious that the A* algotithm is the most efficient algotithm. But when we having no additional information about states beyond that provided in the problem definition, we just can use the uninformed search strategies.

And based on the A* algorithm, we can still make a lot of improvements. In addition to the small trick mentioned in *II.C*, we can use dynamic heuristics, or two-way search, etc. I hope that in the future study, I can continue to deepen the study of these algorithms.