AI Assignment 1: Pathologic

October 2018

1 Python AIMA

1.1 Question 1

In order to perform a search, what are the classes that you must define or extend? Explain precisely why and where they are used inside a tree search. Be concise! (e.g. do not discuss unchanged classes).

Ans: In order to perform a search, the classes that represent an object must be well defined and extended because they contain some specific data that are generalized for all objects. In the tree search, The "Problem" class should be extended because it refers to the instances, which serves as the initial state of the search problem. Furthermore, the "Problem" class is sent to the tree search algorithm and thus initiates the search algorithm. Essentially, the "Problem" class defines the common skeleton for all search problems. Since different search problems have their distinct features and properties, when solving practical problems, we should extend the "Problem" class and add problem-specific features to it. For instance, in the pathological problem, the "goal test" function should be modified because it is hard to predict the final state in advance. Instead, we test the goal by counting whether it still exists while circle tile.

1.2 Question 2

Both breadth first graph search and depth first graph search are making a call to the same function. How is their fundamental difference implemented?

Ans: Both breadth first graph search and depth first graph search are making a call to the graph search function by the "fringe" so that it can add or remove a node in the position according to the given parameter. In breadth first graph search, a FIFO queue is created so that it removes node from the beginning of the queue. In comparison, a "Stack" which is using

LIFO queue is created when solving the depth first graph search problem so that it allows to remove the last one. To be more concrete, in breadth first graph search, when expand a node n in the graph, n's frontiers n_1, ..., n_M will be generated and saved in the "fringe" list. After all the frontiers are generated, n_1's frontiers will then be expanded, which guarantees that all the node in the same level of the search tree will be expanded.

1.3 Question 3

In graph search and tree search, what is the effect of the instruction fringe.extend(node.expand(problem)). What are the classes and methods involved?

Ans: This instruction will exploit all the possible states from the current state and add these nodes into the fringe list, as it shown we have the node class and his method expand .

1.4 Question 4

What is the difference between the implementation of the graph search and the tree search methods and how does it impact the search methods?

Ans: The difference between the implementation of the graph search function and the tree search function is that the last one add the node to the fringe when his state is in the goal where as the graph search function doesn't add the node to the fringe even if it's in the goal only if his state is in the closed list, So there is just The "explored set" of the graph search method is the main difference between the implementation of the graph search and the tree search methods. In the graph search, an "explored set" is maintained in the internal state. In every search epoch, the graph search will check whether current state exists in the "explored set" to avoid repeated searching, which saves the trouble of redundant paths.

1.5 Question 5

What kind of structure is used to implement the closed list? What are the methods involved in the search of an element inside it? What properties must thus have the elements that you can put inside the closed list?

Ans: The closed list is structured as a dictionary of 2D array that returns a Boolean result. The state is saved as a unique key in the closed list. Given a key of one node state, the graph search check if the key has already existed in the closed list. In that, only those states that have never occurred in the closed list can be added to it and create a new (key, value) pair and we can

search an element inside it using the graph search methods by his key and value.

1.6 Question 6

How technically can you use the implementation of the closed list to deal with symmetrical states? (hint: if two symmetrical states are considered by the algorithm to be the same, they will not be visited twice)

Ans: Using the implementation of closed list allows to give even the symmetrical states a different key so that we can able to differ them, because when we add a state into the closed list, one can also add the symmetrical version of the state into the closed list.

2 The Pathologic Problem

2.1 Question 1

Describe the set of possible actions your agent will consider at each state. Evaluate the branching factor.

Ans: In general, our agent have four actions: turn up, turn down, turn left and turn right. To be more specific, particular grids may have less than four actions so the set of possible actions of a state can be 0,1,2,3,4 according to the node state in the boundaries of each state. For instance, in a $3 * 3$ chessboard, only the central grid have exactly four possible actions. Other grids along the sides have two or three possible actions. In that, the branching factor is 4.

2.2 Q2: Problem analysis

2.2.1 part (a)

Explain the advantages and weaknesses of the following search strategies on this problem (not in general): depth first, breadth first

Ans: For the pathological problem, depth first search may outperform breadth-first-search. Depth-first-search selects only one possible path from the node list in till there is no further actions to be performed. Furthermore, depth first search only maintain one path in memory, which greatly saves memory resources. However, one main drawback of depth first search is that if the correct path is select at last, the whole search process may be slow. In comparison, breadth first search exploits all the possible path at every step. When the chessboard becomes larger, the node will grow definitely and

become extremely long, which makes the search process slow and consumes great memory resources.

2.2.2 part (b)

What are the advantages and disadvantages of using the tree and graph search for this problem. Which approach would you choose

Ans: Using the tree search , we don't have the closed list so we don't use much of memory that's why is too fast to solve the problem but the graph search maintain a closed list so we memorize all the states what it took long time to solve the problem.

2.3 Q6: Conclusion and further work

Ans: We would like to choose tree search for this problem. Graph search maintains a closed list which memorizes all the states that has existed in this former search. This practice effectively avoids repeating redundant paths. In comparison, tree search does not have a closed list, which may speed up the searching speed because no more state checking will be carried out.

2.3.1 part (a)

Are your the experimental results consistent with the conclusions you drew based on your problem analysis (Q2)?

Ans: By carrying out some experiments, we found that tree search is faster than graph search and depth-fist-search outperforms breadth-first-search, which is consistent with our former conclusions.

2.3.2 part (b)

Which algorithm seems to be the more promising? Do you see any improvement directions for this algorithm? (Note that since were still in uninformed search, were not talking about informed heuristics).

Ans: From our experience ,The depth-first-search tree search is more promising.

3 Experiments

Note that we print the epoch number of the search algorithm in order to record how many nodes have been expended, which will absolutely make

the search algorithm a little bit slower. In that, the actual time of the search algorithm will perfectly outperform the recorded times.

Our experiment results are recorded as follows:

Table 1: Instance 1

Property / Algorithm	Time(s)	Explored nodes	Moves
depth-first tree-search	0.7	8	4
breadth-first tree-search	0.5	6	4
depth-first graph-search	0.5	8	4
breadth-first graph-search	0.4	6	4

Table 2: Instance 2

Property / Algorithm	Time(s)	Explored nodes	Moves
depth-first tree-search	39.8	86892	46
breadth-first tree-search	180	251639	time out
depth-first graph-search	43.4	86892	46
breadth-first graph-search	0.4	207255	time out

Table 3: Instance 3

Property / Algorithm	Time(s)	Explored nodes	Moves
depth-first tree-search	30.4	79520	36
breadth-first tree-search	63.0	117645	36
depth-first graph-search	36.8	79520	36
breadth-first graph-search	54.6	117645	36

Table 4: Instance 4

Algorithm \ Property	Time(s)	Explored nodes	Moves
depth-first tree-search	165.3	375533	59
breadth-first tree-search	180	265108	time out
depth-first graph-search	182.5	375533	59
breadth-first graph-search	180	258681	time out

Table 5: Instance 5

Algorithm \ Property	Time(s)	Explored nodes	Moves
depth-first tree-search	5.7	11741	46
breadth-first tree-search	24.7	47063	44
depth-first graph-search	6.6	11741	46
breadth-first graph-search	23.1	47063	44

Table 6: Instance 6

Algorithm \ Property	Time(s)	Explored nodes	Moves
depth-first tree-search	24.7	79520	36
breadth-first tree-search	47.7	117645	36
depth-first graph-search	29.4	79520	36
breadth-first graph-search	42.6	117645	36

Table 7: Instance 7

Algorithm \ Property	Time(s)	Explored nodes	Moves
depth-first tree-search	0.4	16	15
breadth-first tree-search	0.5	607	13
depth-first graph-search	0.4	16	15
breadth-first graph-search	0.7	607	13

Table 8: Instance 8

Algorithm \ Property	Time(s)	Explored nodes	Moves
depth-first tree-search	4.5	9706	41
breadth-first tree-search	14.8	33073	37
depth-first graph-search	5	9706	41
breadth-first graph-search	16.2	33073	37

Table 9: Instance 9

Algorithm \ Property	Time(s)	Explored nodes	Moves
depth-first tree-search	2.3	4016	43
breadth-first tree-search	22.8	42453	43
depth-first graph-search	2.3	4016	43
breadth-first graph-search	21.3	42453	43

Table 10: Instance 10

Algorithm \ Property	Time(s)	Explored nodes	Moves
depth-first tree-search	6.3	12235	46
breadth-first tree-search	180	234606	time out
depth-first graph-search	6.5	12235	46
breadth-first graph-search	180	234307	time out