Fundamentals of Artificial Intelligence: Assignment 1

Cho Changhee(chch0474)
Hinako Oshima (hios0003)

Comparing the performance of the different algorithms

In terms of code, the logic of all the problems is similar, but the detailed conditional statements and the data structure used are different. And each algorithm is specialized for a particular situation.

DFS vs BFS

In an unweighted graph, it is a task to find a maze such as pacman, and DFS is expected to be most effective if there is one target point. This is because BFS finds width first, so if the target point is deep, it is slower than DFS. However, DFS has irregular problem-solving times, and if the wrong solution is deep, it can take a very long time. In addition, BFS may be more efficient if there are more than two target points.

UniformCostSearch vs A* search

UniformCostSearch and A* search are suitable for weighted graphs. Unlike DFS and BFS using stacks and queues, priority queues are used to write weights to the queues. The difference between uniform and A* is that if uniformCostSearch only stores weights to specific nodes, A* uses a heuristic function to estimate the distance from a specific node to a target point and stores the added value. In other words, A* even considers future situations. The performance of the A* function is expected to be determined by the performance of the heuristic function. If the heuristic function performs well, A*'s performance will be better, and if not, uniformCostSearch will show more consistent results. On the code, a factor to be stored was added to accumulate and store the weights up to the search node. In addition, the Counter() data structure in the util.py was used to compare it because the weights may be different even with the same node. This is similar to the dictionary data structure in the existing Python.

Corners problem

The key to Q5 is how to save this form of visit when visiting each corner. Do not modify the existing BFS function while changing the storage method. Because it has to work with the problem of Q2. Therefore, one dimension was added to the class and returned. In addition, the visited corners were stored in the added dimension to store where all nodes visited and where they did not visit. And when visiting all corners, 1 was returned from the goal state function in corner class.

Trouble shooting (Issues)

There are Issues I faced when I toke an assignment.

1. I had a problem about the Python version. My python version was originally 3.8, but pacman.py didn't run in that version. Upgrading the version to 3.11 fixed this issue.
2. When I solved q2, at first, I used the logic of DFS as it was and applied it to BFS by transforming only the data structure but doing so did not pass all autograder. The problem was with visited. In the same way as DFS, we had re-checked nodes that have already been verified. So I changed the order about checking the node like this. In DFS, visited processing is performed after the current node is popped from the stack, while in BFS, visit processing is performed before the successor's node is queued.
3. At first, local variables within the class were used to indicate whether a corner was visited. But it didn't work properly. Therefore, it was changed to return by adding a list to the return value without specifying a variable separately. In addition, when adding a value to the list related to whether to visit or not, the value should be added to another local variable. Otherwise, you'll be marked as visiting every corner, even though you haven't actually visited every corner.

Discussion of the Chat-GPT report

Chat GPT's generated text is generally accurate. However, there are three areas where the explanations could be more comprehensive.
Firstly, in the explanation of DFS, it mentions, "[It is often used in applications such as maze-solving, where finding any valid path is more important than finding the shortest path.]" While this is true, it could also include that DFS requires minimal memory to find the destination in maze-solving scenarios. However, if the objective is to find the shortest path to the destination, BFS is superior. The choice between BFS and DFS in maze-solving depends on the specific goal.
Secondly, in the explanation of BFS, it states, "[This is particularly problematic in graphs with a large branching factor or depth.]" While it is true that BFS can be memory-intensive

in graphs with a large branching factor, the depth may not have a significant impact on memory usage if the target node is not located deep within the structure. Therefore, whether depth is a disadvantage for BFS depends on the depth of the target node.
Thirdly, in the explanation of A* Search, it mentions, "[A* Search combines elements of both DFS and BFS while introducing an additional heuristic to guide the search process.]" While this makes A* seem like the superior choice among the three algorithms, considering memory usage tells a different story. DFS only requires memory proportional to the depth, whereas A* needs additional memory to compute the cost function for each node. Therefore, in terms of memory capacity, A* requires more than DFS, and in terms of efficiency, the order is DFS, A*, BFS.

Roles

- Cho Changhee : I solved pacman problem from q1 to q5. I discussed with my group members if I had any question during the process of solving them, and I asked assistant during the supervision class. In the report, I wrote about comparing the performance of the different algorithms and trouble shooting.
- Hinako Oshima : I solved pacman problem with the help from Cho Changhee. I wrote about the discussion about Chat GPT.