TDDC17 Artificial Intelligence :

Lab 2 : Search Algorithms

Arnaud Pecoraro

September 2015

1 Implementing CustomGraphSearch

1.1 Aim

This second lab session explores the concept of graph search algorithms. The agent is still a Vacuum cleaner evolving in a simple grid. Each square of that grid is either clean filled with dirt, or a wall. The vacuum cleaner uses search algorithm to locate the dirt squares. The purpose of this lab is to write our own implementation of *Breadth First Search* and *Depth First Search* algorithms which inherit from a *CustomGraphSearch* class.

Using the Java GUI included in the project files, it is possible to compare the efficiency of differents search algorithms : BFS, DFS, IDS, A*.

1.2 Implementation

Given that the course book and the slides provide us with pseudo code of those algorithms, *figures 3.7 and 3.11*, it is almost only necessary to translate it in Java.

The CustomDepthFirstSearch and CustomBreadthFirstSearch both inherit from CustomGraphSearch. In terms of pseudo code the only difference between them is the handling of the *frontier*. Indeed this collection is a *LIFO* in DFS while a *FIFO* in BFS. The skeleton provide us with a very simple way of dealing with this issue. Thanks to the boolean *insertFront*, it is possible to add nodes in the front or in the back of the queue depending on the algorithm, and therefore simulate a LIFO and FIFO.

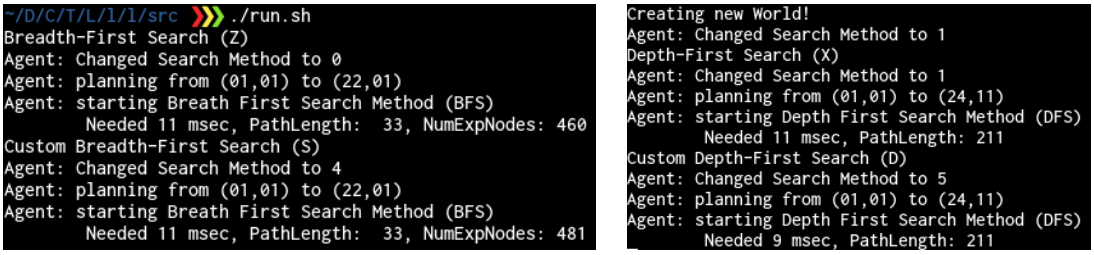
Thus, to create a CustomBreadthFirstSearch object, the constructor just call the superclass with *insertFront* being false :

```
super(false); // The BFS uses a fifo, insertFront is false
```

In the main loop while expanding the currentNode, this boolean is tested before adding the resulting nodes to the frontier.

Each turn the choosen node is tested, if its state match the goalState, the full path is returned. If the frontier is empty an empty path is returned, the algorithm failed. Please refer to the code and comments for more details.

The *Figure 1* presents the result of the two implementations compared to the built-in DFS and BFS algorithms. The results are almost similar in terms of time and space complexity.



(a) (b)

FIGURE 1 – Comparison between built-in and custom implementation (a) BFS - (b) DFS

2 Theory : questions

1. In the vacuum cleaner domain in part 1, what were the states and actions ? What is the branching factor ?

In the domain part 1, each square of the grid is a state, which can be clean or with dirt.The initial state is *In(1,1)*, the starting point in the grid.The goal state is the position of a square filled with dirt. In a given state, the possible actions are *suck* and *move* (up, down, right or left). The branching factor **b** is 4 in this domain.

2. What is the difference between Breadth First Search and Uniform Cost Search in a domain where the cost of each action is 1 ?

In a domain where the cost of each action is 1, the difference between breadth-first search and uniform-cost search is that BFS stops as soon as it find the goal. On the other hand, uniform-cost search still examines all the other nodes at the goal's depth to find a lower cost solution.

3. Suppose that h1 and h2 are admissible heuristics (used in for example A*). Which of the following are also admissible ?

An admissible heuristic never over estimate the cost to reach the goal from n. Given that h1(n) and h2(n) are admissible, **(h1+h2)/2** and **max (h1,h2)** are also admissible.

4. If one would use A* to search for a path to one specific square in the vacuum domain, what could the heuristic (h) be ? The cost function (g) ? Is it an admissible heuristic ?

An admissible heuristic never overestimates the cost to reach the goal. Therefore a possible **h(n)** could be the Manhattan distance between the current node and the goal node. The cost function **g(n)** could be the total ammount of step from the start position to the current node. Given the definition, it would be an admissible heuristic.

5. Draw and explain. Choose your three favorite search algorithms and apply them to any problem domain (it might be a good idea to use a domain where you can identify a good heuristic function). Draw the search tree for them, and explain how they proceed in the searching. Also include the memory usage. You can attach a hand-made drawing.

6. Look at all the offline search algorithms presented in chapter 3 plus A* search. Are they complete ? Are they optimal ? Explain why !

7. Assume that you had to go back and do lab 1 once more, but this time with obstacles. Remember that the agent did not have perfect knowledge of the environment but had to explore it incrementally. Could you still use the search algorithms you have learned to guide the agent's execution ? What would you search for ? Give an example.

If we had to redo lab 1 but with obstacles, we could implement a search algorithm. However, given that the agent has no knowledge of the environment when it starts, it would not be possible to use heuristic search. In fact we would have to use uninformed search algorithms.

The objective of this search would be to cross the world until every square is explored, and

suck the dirt when necessary.

Finally to reach start position, after the environment is fully discovered, we could use an heuristic algorithm, for example A*.