IDA - Department of Computer and Information Science

LINKÖPING UNIVERSITY

TDDC17 Artificial Intelligence

Lab 2: Search

Aim

The aim of this lab is for you to understand how different search strategies work, and to gain experience with a domain-specific application of search algorithms. You will explore the strengths and weaknesses of different search strategies and acquire deeper hands-on knowledge to complement your knowledge about search theory.

Preparation

Read chapters 3 and 4 in the course book. All parts of ch. 3 and 4 are not required for completing the lab, but it is recommended that you do this in any case.

About the lab

This lab consists of two parts, one practical and one theoretical. The first part is a programming assignment where you are to implement search algorithms for an agent in a vacuum cleaner domain. Contrary to the first lab this domain will contain lots of maze-like obstacles which require the agent to plan its movements in advance. You will implement search algorithms to make this work. The second part of the lab is to study and compare different search algorithms. The analysis should include the time and space complexity of the algorithms, and whether they are optimal or complete. To help you in the analysis, there are a number of questions for you to answer.

Part 1 - Vacuum Cleaner Agent

Lab system

The lab system consists of an interactive vacuum cleaner domain simulator with a simplified graphical representation suitable for debugging search algorithms. Contrary to the first lab, the agent has perfect knowledge of where all the dirt and obstacles are. This implies that when the search is performed, no percepts are needed to act since complete information about the world is assumed and no dynamic changes occur other than those caused by the agent. The agent picks the closest dirt by euclidian distance and then searches for the best path. Several different search algorithms can be used and we encourage experimentation! Your task is to implement your own version of breadth-first search and depth-first search according to the directions below.

GUI

The provided vacuum domain simulator application is presented in Figure 1.

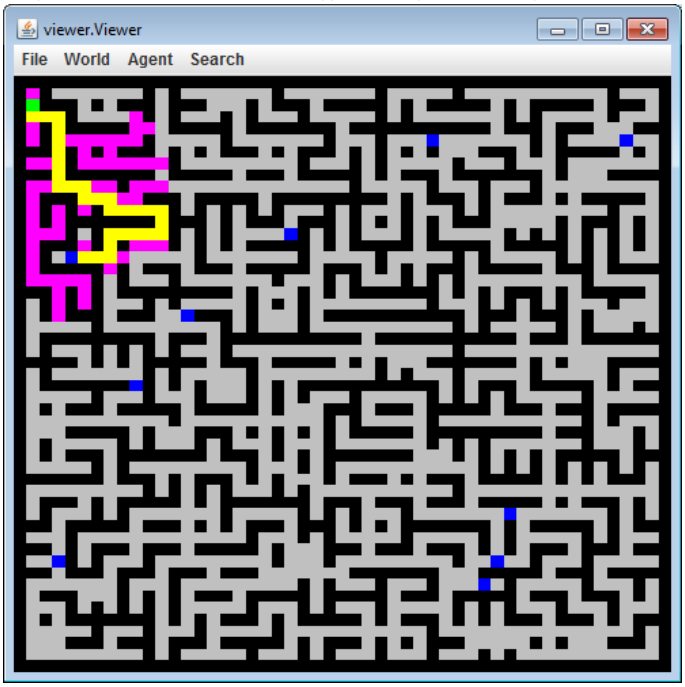

Figure 1: *Example of an A* plan in the vacuum agents' world.*

The vacuum domain simulator shows a maze-like environment with black walls and gray passages. The green dot is the position of the agent, and blue dots are dirt. The yellow path is the currently active plan the agent is following to the nearest dirty spot. As a state is equal to (agent) position in this simplified domain, states that are expanded by a search algorithm will be colored pink. This allows you to see better how different algorithms work. As the agent moves along a corridor it will leave light gray trails to indicate where it has been. Some relevant information is also outputed to the console.
An example output for the A* algorithm looks like this:

```
Agent: planning from (01,01) to (15,17)
Agent: starting AStar Search Method
        Needed 4 msec , PathLength: 50, NumExpNodes: 257
Agent: Vacuuming Dirt
World: Dirt 4 vacuumed at (15,17)
Agent: planning from (15,17) to (09,30)
Agent: starting AStar Search Method
        Needed 3 msec , PathLength: 23, NumExpNodes: 81
```

```
Agent: Vacuuming Dirt
World: Dirt 3 vacuumed at (09,30)
```

The following drop down menus are available to control the agent:

1. `File` Allows you to exit the application.
2. `World` Allows you to reset the world and list the map in ascii form.
3. `Agent` Manually takes an action for the agent, moving or sucking dirt.
4. `Search` Allows you to select a search algorithm and tells it to plan on its own.

When told to search and follow path it will first search if it has no active plan, and then follow that planned path until completion. After the target dirt has been removed it will search for a path to another dirt square.

The agent can preferrably be controlled by the keyboard shortcuts indicated in the drop-down menus. Try holding down 'A' and alternating search algorithms with 'Z','X','C','V'. In addition you can move it around manually with the arrow keys to test the search algorithms from different positions. If you move the agent manually it will re-plan the path the next time it takes an action on its own.

Running the Lab Environment

You can choose either of the two following lab setups, depending on if you want to use a pre-packaged Eclipse environment or your own favorite editor. If you are unsure, go with Eclipse. A .zip is also provided for running it on your personal computer. Note, the general information about working on this lab remotely can be found **here**.

Eclipse version (RDP, ThinLinc or on campus)

1. To prepare the local files run the following commands in the terminal:
   - `mkdir TDDC17_lab2; cd TDDC17_lab2`
   - `cp -r /courses/TDDC17/labs/lab2_workspace/ .`
2. Setup and start the Eclipse editor:
   - To add the Eclipse editor to your environment execute the following command in the terminal: `module add prog/eclipse` or alternatively `module initadd prog/eclipse` to always add it when logging in - note that you will need to open a new terminal window for the command to take effect the first time.
   - `eclipse`
3. When the eclipse editor asks for a path to the workspace choose the lab2_workspace folder which you copied in step 1.
4. Click *Continue* if you get an *Older Workspace Version* warning.
5. Right-click on the project and select refresh.
6. The Eclipse editor will compile the java files automatically. You can start the lab by using the dropdown menu on the run button (green circle with a white triangle/play symbol). You may have to select "Main" in the drop-down menu. If Eclipse asks, choose Java Application.

Console version (RDP, ThinLinc or on campus)

1. To prepare the local files run the following commands in the terminal:
   - `mkdir TDDC17_lab2; cd TDDC17_lab2`
   - `cp -r /courses/TDDC17/labs/lab2_workspace/lab2_part1/* .`
   - `cd src`
2. To compile the required local classes:
   - `./compile.sh`
3. Now you can run the simulation environment GUI:
   - `./run.sh`

Running it on your personal computer

You can download the Eclipse compatible workspace here: **.zip file**. Simply extract the file and point Eclipse to it during the start-up ("Select a directory as workspace") and follow the general instructions above. If you do not have Eclipse installed it is recommended that you download a version suitable to your platform **here**.

Tasks

Play around a bit with different search algorithms in the simulator. The full source for the above vacuum domain minus a few search related classes is provided in the lab environment above. You are to implement the missing parts of the three files "CustomGraphSearch.java", "CustomBreadthFirstSearch.java" and "CustomDepthFirstSearch.java". The CustomGraphSearch class is the foundation of both the others so you only need to change **one line** in each of the latter. A code skeleton is provided for the graph search with some hints and comments. You can run these from the simulator in the same way as other search algorithms.

We strongly recommend you to have read at least the first half of chapter 3 before starting. In particular pseudo-code for the graph search can be found in **Figure 3.7 (3rd edition only)** and a bit more detailed (from the BFS perspective, note how BFS and DFS are variants of graph search) in **Figure 3.11 (3rd edition, Figure 3.9 in 4th edition)**. As a guideline our reference implementation is less than one page of extra code. When the goal is found remember to always set the member variable **"path"** before returning from the function, this is later re-used by the agent.

Part 2 - Theory

In your report, you will have to answer all the questions in the following list:

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

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

3. Suppose that h1 and h2 are admissible heuristics (used in for example A*). Which of the following are also admissible?
a) (h1+h2)/2
b) 2h1
c) max (h1,h2)

4. If one uses A* to search for a path to one specific square in the vacuum domain, what could the heuristic function (h) be? What could the cost function (g) be? Is your choice of (h) an admissible heuristic? Explain why.

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 (i.e. Best-first search, Breadth-first search, Uniform-cost search, Depth-first search, Iterative deepening search, Bidirectional search, Greedy best-first search and A* search). Are they complete? Are they optimal? Explain why!

7. Assume that you had to go back and do Lab 1/Task 2 once more (if you did not use search already). Remember that the agent did not have perfect knowledge of the environment but had to explore it incrementally. Which of the search algorithms you have learned would be most suited in this situation to guide the agent's execution? What would you search for? Give an example.

Examination

Demonstrate part 1 to a lab assistant. Write a report where you present what has been done in part 1 and the answers to the questions in part 2. Also include the source code of part 1 (electronic submission only). **Submission instructions**.

Page responsible: **Fredrik Heintz**
Last updated: 2023-08-18