Report Assignment #1: Labyrinth

Authors: Simon Bilgeri (422852), Rui Lopes (31689) – Group 29

Python Version: 3.4.2

Code usage: python assign1.py <path to labyrinth input file>, default: inputTest7.txt

Answers to the questions (a)-(e):

(a) The problem state (labyrinth) is read from a file and converted into a 2D matrix (lists in a list). This matrix is then converted into a graph representation in the function *create_graph(labyrinth)*. The possible operators are (UP,DOWN,LEFT,RIGHT,PUSH) and the current state of each node is represented by the tuple (x,y,doorstate), where x,y is the field position in the matrix and doorstate defines if each door is currently open or closed. The full state space is created by iterating through all possible fields and doorstates of the given labyrinth. At a switch field, the agent can go to another branch of the graph with another doorstate and the same field position. The initial state is the labyrinth position with value 2 and the original doorstate. The goal states are defined by the labyrinth position (x,y) with value 3 and any doorstate. The path cost increases by 1 for each node traversed. Furthermore, the environment is observable, discrete, known and deterministic.

(b) Uniformed methods:

- Breadth First Search (BFS): It uses a FIFO queue and thus is optimal and complete in our case (path costs do not decrease with depth, are constant). Hence, the shortest path to a goal is found with that method. But: Time and Space complexity is high. Both: O(b^d), with b=branching factor and d=depth of closest goal node. In this problem, the Uniformed Cost Search is the same as BFS, because path costs are increasing by one for each traversal.
- Depth First Search (DFS): Implementation similar to BFS, but it uses a LIFO stack! Thus, is not complete and not optimal. (e.g. first path has infinite depth, or higher cost than optimal path). Nevertheless, it is more efficient in terms of memory O(b*m) with m=maximum length of any path.
- Iterative deepening DFS (IdDFS): It tackles the difficulties of DFS in iteratively performing limited DFS searches. The maximum depth of any path is increased until a goal state is found or the maximum depth is reached. In that way IdDFS is complete and optimal if the path costs do not decrease with depth. The time complexity is only slightly larger than DFS but it still has a smaller space complexity than BFS. O(b*d).

Informed methods:

- Greedy best-first search: Given a heuristic function h(n), this search type sorts the queue according to their heuristic value and selects the best one, i.e., the node with the lowest value is expanded first. It is complete if repeated nodes are not allowed, but the result is not optimal (always goes into the direction of the goal, regardless costs). Time and space complexity are O(b^m).
- A*: Improvement of Greedy approach. Evaluation function f(n)=path_cost+heuristic_estimate. The node with the best heuristic value is taken from the queue. If the heuristic function is consistent (also admissible) then A* is optimal and complete.
- Iterative Deepening A*(IDA): Similar to IdDFS, but not the maximum depth of the paths is iteratively increased, the maximum evaluation value f is changed. It is set to the smallest f that exceeded the value

in the previous iteration. In that way, the memory used is bounded.

(c) Each of the 6 search algorithms is completely independent of the labyrinth problem. The functions expect a domain-independent graph representation (dictionary), a start and goal nodes (can be more than one in a list) and maybe a cost and a heuristic function. In that way, the algorithms can be used for any type of search problem.
The domain dependent part is already described in answer (a). The graph is built with the function *create_graph(labyrinth)* and the cost and heuristic functions are defined with the functions *n_func(node)* and *h_func(node,goal)*. These functions are also passed to the specific domain-independent search methods.

(d) The heuristic function used in the informed search methods is the L1-norm of the given labyrinth, which represents the number of steps in vertical and horizontal direction to reach the goal if there were no borders and doors. Hence the heuristic addresses a relaxed labyrinth problem and thus h(n) <= r(n), where r(n) represents the real value, or number of actions. Thus, this heuristic is admissible. This heuristic is also consistent, as the f value cannot get smaller in each iteration. h(n)<=1+h(n'). h(n') can only decrease by one in each step.
Another heuristic function h_2 could be the Euclidean L-2 norm, but h_2<=h_1 and thus the L-1 norm is the better heuristic.

(e) The following numbers and comparisons are based on the test labyrinths provided with the assignment. Computational Time (Computer dependent, blue: fast, green: fast and optimal) and shortest Path:

	BFS	DFS	IdDFS	Greedy	A*	IDA	Shortest Path
Lab_1	0.02s	0.02s	0.02s	0.02s	0.02s	0.02s	4 actions
Lab_2	0.02s	0.02s	0.02s	0.02s	0.02s	0.02s	4 actions
Lab_3	0.02s	0.02s	0.03s	0.02s	0.02s	0.03s	20 actions
Lab_4	0.02s	0.02s	0.03s	0.02s	0.02s	0.03s	26 actions
Lab_5	0.02s	0.02s	0.03s	0.03s	0.03s	0.03s	15 actions
Lab_6	0.04s	0.02s	1.3s	0.03s	0.02s	0.04s	30 actions
Lab_7	0.04s	0.04s	3.9s	0.05s	0.05s	1.6s	140 actions
Lab_Ex	0.04s	0.04s	0.04s	0.07s	0.07s	0.44s	50 actions
41x41	0.03s	0.03s	0.13s	0.03s	0.03s	0.14s	74 actions

This comparison shows that DFS is always the fastest algorithm for the given labyrinth problems and the used computer. Nevertheless, the result may not be the shortest Path through the labyrinth, as DFS is not optimal. A* with a consistent heuristic and BFS provide optimal solutions and are also fast.
Furthermore, for different problems, with a larger branching factor, BFS can be very inefficient as it explores all nodes in the current depth. A*, which makes use of a-priori knowledge in the heuristic function, should be implemented instead. In applications which demand low memory consumption, IdDFS or IDA can be implemented, which leads to a decrease in computational performance. The Greedy approach should not be used, as it does not consider the path costs and it is not faster than A*.