TP S (Search)
Techniques of AI [INFO-H-410]
Correction

v1.0.0

Source files, code templates and corrections related to practical sessions can be found on the UV or on github (`https://github.com/iridia-ulb/INFOH410`).

Fundamentals for search

Question 1. Suppose you have a data structure D with the push and pop operation. Suppose you have the following sequence of push operations: push(1), push(3), push(2).

a) Which elements are returned by two consecutive pop operations if D is a stack, if D is a queue?

b) Now suppose that D is a priority queue and that the priority of each element is the value of the element itself. Which elements are returned by two consecutive pop operations now?

Use python to validate your answers.

Answer: (a) If D is a stack, the first pop operation returns 2, the second one returns 3. If D is a queue, the first pop operation returns 1, the second one returns 3.

(b) If D is a priority queue, the first pop operation returns 3, the second one returns 2.

General search algorithms

Question 2. The outline of a general tree search algorithm is sketched below. How can this outline be turned into:

a) depth first search

b) breadth first search

c) best first search

d) A* search

```
q = [start]
while len(q) != 0:
    current = q.pop()
    if current == goal:
        print("Found goal")
        break
    for n in neighbors(current):
        q.append(n)
```

For each of those search algorithms, start from the general algorithm above and add more details to it, such that it becomes the desired search algorithm.

Answer: (a) for depth first search, the agenda should operate as a stack: new nodes are added to the front of the agenda and taken from the front.

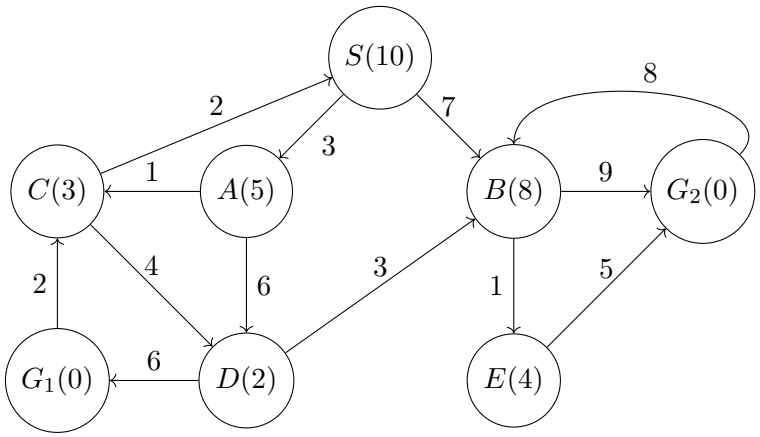
(b) for breadth first search, the agenda should operate as a queue

(c) for best first search, the agenda should be a priority queue with priorities f(n) = h(n)

(d) for A*, the agenda should be a priority queue with priorities f(n) = g(n) + h(n)

Search spaces

Question 3. Consider the following state space:



- S is the initial state
- G1 and G2 are goal states
- Arcs show actions between states (e.g., the successor function for state S returns A, B).
- Arcs are labelled with actual cost of the action (e.g. the action from S to A has a cost of 3).
- The numeric value in parentheses in each state is the states h-value (e.g. $h(A) = 5$).

You should assume the following on the operational details of the algorithms:

- The algorithm does not check if a state is revisited, so there may be several nodes with the same state in the search tree.
- The algorithm terminates only when it selects a goal node for expansion, not when it is generated by the successor function.

- The successor function always orders states alphabetically.

a) Give the sequence of states that will be visited, together with the total cost of reaching the goal state, for (1) depth first search, (2) breadth first search and (3) A*.

b) In python what data structure could be used to store this graph ? Implement one of the 3 algorithms in python and verify your previous answers (you can use the code template (see page 1)).

Answer: (a) Depth first search, path found: S – A – C – D – B – E – G2 = 17 Breadth first search, path found: S – B – G2 = 16 A*, path found: S – A – C – D – G1 = 14. (b) We can use an adjacency list or adjacency matrix, using python lists (for implementation see code on github).

The Missionaries and Cannibals Puzzle

Question 4. There are three missionaries and three cannibals on the west bank of a river. There is a boat on the west bank that can hold no more than two people. The missionaries wish to cross to the east bank. But they have a problem: If on either bank the cannibals ever outnumber the missionaries, the outnumbered missionaries will be eaten. Is there a way for the missionaries to get to the east bank without losing anyone?

a) Represent this puzzle as a search problem by defining the states and actions.

b) Investigate your initial representation of the puzzle as you did in part (a): is there any redundant information in your representation? Try to remove any redundant information from it.

c) Try to solve the puzzle by enumerating possible states.

d) Implement and solve the puzzle using python.

Answer: (a) A straightforward approach would be to represent the state by an array containing the following:

- number of missionaries at the left,
- number of cannibals at the left,
- number of missionaries at the right,
- number of cannibals at the right,
- number of missionaries in the boat,
- number of cannibals in the boat,
- position of the boat,

(b) This representation is overcomplete: if you know how many missionaries and cannibals we have at the one bank and in the boat, we know how many there are at the other bank, as we always have 3 missionaries and 3 cannibals. So, a first improvement might be the following representation:

- number of missionaries at the left,
- number of cannibals at the left,
- number of missionaries in the boat,
- number of cannibals in the boat,
- position of the boat.

However, as the boat can contain at most 1 missionary and 1 cannibal, the cannibals can never outnumber the missionaries in the boat, so it is of no use to represent the number of each in the boat. So, we could simply store:

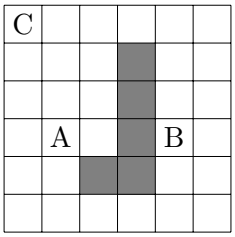
- number of missionaries at the left,
- number of cannibals at the left,
- position of the boat.

For instance <3,3,1> represent the initial state, with 3 missionaries and 3 cannibals at the left bank, with the boat at the left bank. The state <3,1,0> can then be reached from this state, by the action "transfer two cannibals to the right bank". As a result, we have 3 missionaries at the left bank, one cannibal at the left bank and the boat at the right bank.
(c) A state is represented by <x,y,z>, with x varying between 0 and 3, y varying between 0 and 3 and z being either 0 or 1. So, we have $4 \times 4 \times 2 = 32$ states at most. For the boat being at left, we can for instance list all 16 possible states: 331/321/311/301/231/221/211/201/131/121/111/101/031/021/011/001. From this list, we can eliminate the states 231/211/201/131/121/101 because we have more cannibals than missionaries at one bank. The same holds of course for the boat being at the other side. So we have $3266 = 20$ states. The states <0,0,1> and <3,3,0> are not valid (because it implies that you move an empty boat), the states <3,0,1> and <0,3,0> are not reachable, because all preceding states are invalid: <3,0,1> can only be reached from <2,0,0> or <1,0,0>. However, in both states the cannibals do outnumber the missionaries. In total we have hence 16 states.
(d) One solution is the following: <3,3,1>, <3,1,0>, <3,2,1>, <3,0,0>, <3,1,1>, <1,1,0>, <2,2,1>, <0,2,0>, <0,3,1>, <0,1,0>, <0,2,1>, <0,0,0>,

see github for implementation.

Path planning

Question 5. Imagine a maze with some obstacles, you can only move 1 cell at a time in any of the four directions (no diagonals). We have to find the shortest path from A to B.



a) Discuss whether depth first search and breadth first search will always find a path from A to B. Will they find the shortest path?

b) Suppose that we want to use informed search in order to guide our search. What heuristic could you use ? Is this heuristic admissible ?

c) Implement the A* algorithm for this maze using python (you can use the code template (see page 1)).

d) Now suppose that there is a direct underground connection from C to B. Hence, the shortest path from A to B now is over C (it is only five steps). Is your heuristic still admissible ?

Answer: (a) If it is ensured that the algorithms do not get stuck in loops, then they will find a solution. Depth first search will return the first path to the goal state it happens to encounter, which is not guaranteed to be the shortest one. Breadth first search, will find the shortest path.
(b) We will use the Manhattan distance as a search heuristic, it is admissible. (c) see github for implementation.
(d) The heuristic is not admissible anymore, the A* will not find the shortest path through C.

Question 6. The optimality condition of A* is expressed as follows: "If the heuristic h(n) is admissible, then the solution returned by A* is optimal". In order to prove this, we assume that the optimal solution n* has an optimal cost f(n*) = C*.

a) Suppose that there is a solution G2 which is not optimal in the agenda. What does this teach us about f(G2)?

b) Suppose that there is a node n in the agenda which is on the path to the optimal solution G. What does this learn about f(n)?

c) Proove that if h(n) is admissible, then the solution returned by A* is optimal.

Answer: A heuristic is admissible if it never overestimates the cost to the goal. (a) Suppose A* returns a solution which is not optimal. This means that at some point a non optimal solution G2 must be in the agenda. If G2 is not optimal, then $f(G2) > f(n*) = C*$.
(b) If h(n) is admissible, then h does not overestimate the cost to the goal, so we can put $f(n) < C*$.
(c) Combining a and b results in the inequality: $f(n) < C* < f(G2)$. Hence, every node on the path towards the optimal solution has a cost less than the cost of the non optimal solution G2. Hence, G2 will never be expanded from the priority queue. So, G2 cannot be the solution of the A* algorithm.

Found an error? Let us know: `https://github.com/iridia-ulb/INFOH410/issues`