Introduction to Artificial Intelligence

UNIZG FER, AY 2021/2022

Exercises, v1

2 State space search

1 (T) One component of a state space search problem is the successor function, $succ : S \to \wp(S)$, which defines the transitions between the states. This function can be defined either explicitly or implicitly. **What is the advantage of an implicit definition of the successor function over an explicit definition?**

- A An explicit definition is not possible if the set of states is infinite or if the state space has cycles, while this is not the case for an implicit definition, provided the state space has at least one goal state
- B An implicit definition relies on a set of operators to define the successor states, which ensures that the algorithm is complete, whereas with an explicit definition the algorithm may get trapped in a loop
- C An explicit definition is infeasible for problems with a large number of states, and it is then easier to implicitly (algorithmically) define all the ways in which a state can be altered to obtain the next state
- D An implicit definition can seamlessly be extended with a pointer to the parent node, which is what we need to reconstruct the solution, wheras an explicit definition requires us to additionally store all open nodes

2 (C) Function *succ* defines the transitions between states $\{a, b, c, d, e, f\}$ as follows: $succ(a) = \{b, c\}$, $succ(b) = \{c, d\}$, $succ(c) = \{d, e\}$, $succ(d) = \{a, e\}$, $succ(e) = \{f\}$, $succ(f) = \{d\}$. Let a be the initial state, and f the goal state. Assume lexicographic ordering between the nodes. **For which algorithm will the sequence of tested nodes be** $\dots, b, c, a, b, c, \dots$**?**

- A Breadth-first search
- B Depth-first search
- C Iterative deepening search
- D Depth-limited search with $k = 3$

3 Heuristic search

3 (T) The best-first search algorithm is not optimal and we refer to it as a “greedy algorithm”. However, depth-first search is also not an optimal algorithm, but we don’t call it greedy. **Why do we say that the best-first search algorithm is ”greedy”, but we don’t say so for the**

depth-first search algorithm?

- A Because depth first search uses stack to store open nodes and also closes every expanded node, whereas best-first search uses a priority queue and never closes a node, hence its space complexity is larger than that of depth-first search
- B Because best-first search always selects the node that currently seems to be the best one according to the heuristics, regardless of the total path cost, whereas depth-first search is selecting nodes not by heuristics but by depth
- C Both algorithms are suboptimal because they don't search level by level, however best-first search is greedy as it doesn't consider the heuristic values but only the total path costs, and hence yields better (shorter) solutions than depth-first search
- D Because best-first search uses a heuristic function which, if not optimal, need not result in an optimal path, whereas depth-first seach selects nodes based on depth rather than a heuristic, and hence can get stuck in an infinite loop

4 (C) Let the set of states be $S = \{a, b, c, d, e, f\}$. The successor function is defined as $succ(a) = \{(b, 6), (d, 2)\}$, $succ(b) = \{(c, 3), (d, 5)\}$, $succ(c) = \{(d, 4), (f, 1)\}$, $succ(d) = \{(e, 2)\}$, $succ(e) = \{(b, 1), (f, 6)\}$, $succ(f) = \varnothing$. The initial state is a and the goal state is f. The values of the heuristic function are $h(a) = 9$, $h(b) = 2$, $h(c) = 1$, $h(d) = 7$, $h(e) = 2$, and $h(f) = 0$. Run the A^* algorithm, in each iteration writing down the list of open nodes (O) and the list of closed nodes (C). In zeroth iteration, let $O = [(a, 0)]$ and $C = \varnothing$. **How do the lists O and C look like after the fifth iteration of A^*?**

- A $O = [(c, 8), (f, 10)]$, $C = \{(a, 0), (b, 5), (d, 2), (e, 4)\}$
- B $O = [(e, 4), (c, 9)]$, $C = \{(a, 0), (b, 6), (d, 2)\}$
- C $O = [(f, 10)]$, $C = \{(a, 0), (b, 6), (d, 2), (c, 9)\}$
- D The algorithm terminates before the fifth iteration

5 (C) We're using the A^* algorithm to solve a 3×3 sliding puzzle problem. The costs of all transitions is set to 1. The heuristic function is computed as the number of displaced pieces (the unoccupied position is not counted, hence the maximum value of the heuristic is 8). While searching for the path from the initial state s_0 to the goal state s_g, at some point in time the A^* algorithm generates a node with a state s_x. The path cost $g(n)$ from the node with state s_0 to the node with state s_x is 15. The state s_x and the goal state s_g are the following:

$$s_x = \begin{array}{|ccc|}\hline 4 & 1 & 3 \\ 7 & 2 & 5 \\ 8 & \square & 6 \\ \hline\end{array} \qquad s_g = \begin{array}{|ccc|}\hline 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & \square \\ \hline\end{array}$$

Right before expanding the node with state s_x, the costs $f(n)$ of all other nodes in the list of open nodes is larger by at least 5 than the cost $f(n)$ of the node with state s_x. Run the next three iterations of the A^* algorithm. **What are the costs $f(n)$ of the next three nodes that the algorithm will test and expand?**

A $22, 22, 22$ B $22, 23, 21$ C $23, 22, 20$ D $23, 23, 23$

6 (P) We're working on a 4×4 sliding puzzle problem and have implemented three heuristic functions: h_1, h_2, and h_3. Heuristic h_1 is implemented so that it internally runs a depth-limited search with a depth limit set to $k = 5$, and returns the depth of the solution, or k if no solution has been found. Heuristic h_2 is implemented as iterative deepening search, but with a depth limit of $k = 3$, and also returns the depth of the solution found, or k if no solution has been found. Lastly, heuristic h_3 is implemented as iterative deepening search, but modified so that in each iteration the depth limit is increased by 3 instead by 1, and returning the depth of the solution it has found. We wish for the A^* algorithm to be as informed as possible, but still be complete and optimal. We can achieve this with a suitable combination of heuristics h_1, h_2, and h_3. **Which of the following combinations**

of the three heuristics will give us the most informed, but still complete and optimal A^* algorithm?

A $\min(\min(h_1, h_2), \max(h_2, h_3))$

B $\min(\max(h_1, h_2), \min(h_1, h_3))$

C $\max(h_1, h_2, h_3)$

D $\min(\max(h_1, h_2), h_3)$

7 (P) We're working on finding the shortest route between any point in Zagreb and the St. Mark's Square in the Upper Town. The path cost is the average walking time (in minutes) of a moderately inquisitive tourist. Let's consider one detail on our city map, between Ban Jelačić Square (s_1), Dolac Market (s_2), Zakmardi's Stairs (s_3), and the Blody Bridge (s_4). One can walk from s_1 to s_2 along Splavnica in 2 minutes, and from s_2 to s_4 along Tkalčić street in 5 minutes. From s_1 to s_3 one can get along Radić street in 3 minutes, and from s_3 to s_4 further down Radić street in 4 minutes. To find the route to St. Mark's Square, we use the A^* algorithm with some heuristic function h, which is consistent. Let $h(s_1) = 13$, $h(s_2) = 14$, $h(s_3) = 12$, $h(s_4) = 10$. We'd like to improve the search speed (reduce the number of A^* iterations), but we still want the route to be the shortest possible one in terms of walking time. **Which of the following changes to heuristic function h will result in the greatest speed improvement, but still give us the shortest route to the goal?**

A $h(s_1) = 16$ B $h(s_1) = 12$ C $h(s_1) = 14$ D $h(s_1) = 15$