Assignment3 – Search

Problem 3.1 (Search Algorithms)

Consider the following *directed graph*:



Every *node* is labeled with n : h(n) where n is the identifier of the node and h(n) is the *heuristic* for estimating the cost from n to a goal node. Every edge is labeled with its actual cost.

1. Assume that *I* is the goal node. Argue whether or not the *heuristic* is *admissible*.

Solution: It is not admissible: The cost from *D* to the goal is 1 + 5 = 6 < 7 = h(D), and a *heuristic* must not overestimate that cost.

Now assume you have already expanded the node *A*. List the **next** 4 **nodes** (i.e., excluding *A*) that will be expanded using the respective algorithm. If there is a tie, break it using alphabetical order.

2. depth-first search

Solution: B, C, F, H

3. breadth-first search

Solution: B, C, E, F

4. uniform-cost search

Solution: C, F, B, E

5. greedy-search

Solution: B, E, C, G

6. A^* -search

Solution: C, F, G, B

Problem 3.2 (Formally Modeling a Search Problem)

Consider the Towers of Hanoi for 7 disks initially stacked on peg A.

Is this problem deterministic? Is it fully observable?

Formally model it as a *search problem* in the sense of the mathematical definition from the slides. Explain how your *mathematical* definition models the problem.

Note that the formal model only defines the problem — we are not looking for solutions here.

Note that modeling the problem corresponds to defining it in a programming language, except that we use *mathematics* instead of a programming language. Then explaining the model corresponds to documenting your *implementation*.

Solution: We need to give (S, A, T, I, G).

Because the problem is deterministic, we know $\#(T(a, s)) \le 1$. Because the problem is fully observable, we know #(I) = 1.

Let $D = \{1, 2, 3, 4, 5, 6, 7\}$ (the set of disks) and $P = \{A, B, C\}$ (the set of pegs). We put:

• $S = D \rightarrow P$, i.e., a state *s* is a function from disks to pegs.

Explanation: In state *s*, the value s(d) is the peg that disk *d* is on. Because disks must always be ordered by size, we do not have to explicitly store the order in which the disks sit on the pegs.

• $\mathcal{A} = \{(A, B), (B, A), (A, C), (C, A), (B, C), (C, B)\}$, i.e., an action *a* is a pair of different pegs.

Explanation: (p,q) represents the action of moving the top disk of peg p to peg q.

• For $s \in S$ and $p \in P$, we abbreviate as top(s, p) the smallest $d \in D$ such that s(d) = p.

Explanation: top(s, p) is the top (smallest) disk on peg p in state s.

Then $T : \mathcal{A} \times S \to \mathcal{P}(S)$ is defined as follows:

- If top(s,q) > top(s,p), we put $T((p,q),s) = \{s'\}$ where $s' : D \to P$ is given by * s'(d) = q if d = top(s,p)* s'(d) = s(d) for all other values of d
- otherwise, $T((p,q)), s) = \emptyset$

Explanation: In state *s*, if the top disk of *q* is bigger than the top disk of *p*, the action (p, q) is applicable, and the successor state *s'* of *s* after applying (p, q) is the same as *s* except that the top (smallest) disk on peg *p* is now on peg *q*.

• $I = \{i\}$ where the state *i* is given by i(d) = A for all $d \in D$.

Explanation: Initially, all disks are on peg A.

• $G = \{g\}$ where the state g is given by g(d) = B for all $d \in D$.

Explanation: There is only one goal state, described by all disks being on peg *B*.

Note that there are many different correct solutions to this problem. In particular, you can use different definitions for S (i.e., model the state space differently), in which case everything else in the model will be different, too. Often a good model for the state space can be recognized by how straightforward it is to define the rest of the model formally.

Even if you used a different state space, a good self-study exercise is to check that the above (a) is indeed a *search problem* and (b) correctly models the Towers of Hanoi. Continuing the above analogy to programming languages, (a) corresponds to compiling/type-checking your *implementation* and (b) to checking that your *implementation* is correct.

Problem 3.3 (Heuristic Searches)

Consider the graph of Romanian cities with edges labeled with costs c(m, n) of going from *m* to *n*. c(m, n) is always bigger than the straight-line distance from *m* to *n*. c(m, n) is infinite if there is no edge.

Our search algorithm keeps:

- a list *E* of expanded nodes *n* together with the cost *g*(*n*) of the cheapest path to *n* found so far,
- a fringe F containing the unexpanded neighbors of expanded nodes.

We want to find a cheap path from Lugoj to Bucharest. Initially, *E* is empty, and *F* contains only Lugoj. We terminate if *E* contains Bucharest.

Expansion of a node *n* in *F* moves it from *F* to *E* and adds to *F* every neighbor of *n* that is not already in *E* or *F*. We obtain g(n) by minimizing g(e) + c(e, n) over expanded nodes *e*.

As a *heuristic* h(n), we use the straight-line distance from n to Bucharest as given by the table in the *lectures*.

Explain how the following algorithms choose which node to expand next:

1. greedy search with *heuristic h*

Solution: The search expands the node n that minimizes the function h(n).

2. A* search with path cost g and *heuristic* h

Solution: The search expands the node *n* that minimizes the function g(n) + h(n)

3. Explain what h^* is here and why *h* is *admissible*.

Solution: $h^*(n)$ is the cost of the shortest path from *n* to Bucharest (which we do not know unless we expand all nodes). Because c(m, n) is always bigger than the straight-line distance, every path is longer than the straight-line distance between its end points. Thus $h(n) \le h^*(n)$.

4. For each search, give the order in which nodes are expanded. (You only have to give the nodes to get the full score. But to get partial credit in case you're wrong, you may want to include for each step all nodes in the fringe and their cost.)

Solution: Lugoj (244), Mehadia (241), Drobeta (242), Craiova (160), Pitesti (100), Bucharest (0)

A* search: Lugoj (0+244), Mehadia (70+241), Drobeta ((70+75)+242), Craiova (70+75+120)+160), Timisoara (111+329), Pitesti ((70+75+120+138)+100), Bucharest ((70+75+120+138+101)+0)

Problem 3.4 (Heuristics)

Consider *heuristic search* with *heuristic h*.

1. Briefly explain what is the same and what is different between *A*^{*} search and greedy search regarding the decision which node to expand next.

Solution: Both choose the node that minimizes a certain function. As that

function, A^* uses the sum of path cost and *heuristic* whereas greedy only uses the *heuristic*.

2. Is the constant function h(n) = 0 an *admissible heuristic* for A^* search?

Solution: Yes. (But it's a useless one.)