

**Homework 1 Solution Sketches**  
**Aleksandr Khasymski**

---

1. Problem 3.15

Environment: Partially observable, deterministic, sequential, static, continuous, single agent

Start Test: Robot at coordinates of the start state

Successor Function:

Moving a small finite distance in any direction

OR

Moving from one obstacle vertex to another (visible) vertex in a straight line

Cost: Distance traveled

Goal State: Robot at the coordinates of the destination

2. Problem 4.4

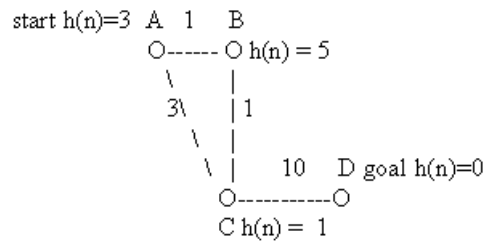


Figure 1: The Environment

$h(n)$  is admissible as it does not overestimate the cost

It is inconsistent because  $h(B) \geq c(B, C) + h(C)$  because  $5 \geq 1 + 1$

Graph-search using A\* with this h returns a suboptimal solution because it expands A, C, D and terminates with solution A, C, D, while the optimal solution is A, B, C, D

You can check that the search would have returned an optimal solution if  $h(B)$  was  $\leq 2$

3. Problem 4.6

An example of an inadmissible heuristic  $h$  is (number of misplaced tiles) \* 3. This definitely works for the goal states (note:  $h(\text{goal})=0$ ) but when there is even one tile misplaced, it overestimates.

Since we are given that  $h(n)$  overestimates by at most  $c$ , we have:

$$h(n) \leq h^*(n) + c, \text{ where } c \text{ is a positive amount.}$$

Let the true solution have path cost  $m$ .

Consider two goal states G1 and G2. Let G1 be the goal that is suboptimal within  $c$ . i.e.,  $g(G1) \leq m + c$ .

Let G2 be the goal that is suboptimal by more than  $c$ . i.e.,  $g(G2) > m + c$ .

We have to prove that A\* with  $h(n)$  will find G1 but not G2. In particular, we must show that A\* with  $h(n)$  will never attempt to expand G2 (since if it did, it would find this as the solution). G2 will never be expanded if we can show that  $f(G2) \geq f(n)$  for any other node on the path to the optimal goal. This is easy to see:

$$\begin{aligned}
f(G2) &\geq g(G2) \\
&= m + c \\
&= g(n) + h^*(n) + c \\
&= g(n) + h(n) \\
&= f(n)
\end{aligned}$$

#### 4. The Knight Problem

The problem is solvable:

It is known that a knight can reach any position of a 5 by 5 board starting from any other (the Knight's tour). Construct a 5 by 5 board around the goal position with the goal position at the center of that board. Travel from the start to the first position you encounter in that board. Move along the Knight's tour until at the goal

States: (x,y) coordinates on the chessboard

Successor function: returns the possible coordinates obtained by a knightly move from current position

Goal test: is arriving at the goal state

Cost: is number of moves performed by the knight

Heuristic: Manhattan distance divided by 3 (why does this work?)

#### 5. 8-Puzzle (courtesy of Mike Overson)

Results:

9 or 16	Tiles Used	Nodes Expanded	Branching Factor
9	0 (BFS)	1872.4	2.721
9	1	632	2.716
9	2	130.8	2.737
9	3	80	2.742
9	4	41	2.678
9	5	29	2.672

(Averaged over 5 problems)

16	0 (BFS)	1465.75	3.210
16	1	204	3.027
16	2	197	2.990
16	3	169	2.963

(Averaged over 4 problems)

As expected, the larger the pattern database that the search uses, the fewer the nodes that have to be expanded (and the faster the search can find the goal). However, the larger pattern databases take an extremely long time to generate; as the book says, that time is amortized by the time saved searching in the long run.

The number of nodes pruned by each heuristic was either approximately the same as or slightly higher than the number of nodes expanded for that heuristic. Smaller pattern databases (BFS included) would prune more than they expanded, and the largest pattern databases had about the same number expanded and pruned.

Of course, the larger the pattern database, the more memory is required to store it. Small pattern databases take up hardly any space, but, for example, the pattern database for the 16-puzzle with 3 numbered tiles takes up 900kb.

As expected, the more effective the heuristic, the more space is going to be required to generate it. If the best heuristics took the smaller amount of space, there would never be a reason to generate the large pattern databases.

As for the time to calculate the heuristic, it is definitely true that you save time by just giving  $h(n) = 0$  (BFS) at every node, and you also don't have to spend the time reading in from the pattern database file. However, the time spent to calculate  $h$  and read the database are relatively very, very small when compared to the time it would take BFS to randomly stumble upon the answer.