# ICS 271 - Solutions Homework 2

1. Trace the operation of A* search applied to the problem of getting to Bucharest from Lugoj using the straight-line distance heuristic.

   **Answer:**

   Using the single-letter labels from the map on page 100 and omitting simple back-tracking state expansions due to space limitations.

| | Action | Open Nodes | $f(n)$ | $g(n)$ | $h(n)$ |
|---|---|---|---|---|---|
| 1 | Initial State | (L, 244) | 244 | 0 | 244 |
| 2 | Expand Lugoj | | | | |
| | Add Timisoara | (T, 440) | 440 | 111 | 329 |
| | Add Mehadia | (T, 440), (M, 301) | 301 | 70 | 241 |
| 3 | Expand Mehadia | | | | |
| | Add Drobeta | (T, 440), (D, 387) | 387 | 145 | 242 |
| 4 | Expand Drobeta | | | | |
| | Add Craiova | (T, 440), (C, 425) | 425 | 265 | 160 |
| 5 | Expand Craiova | | | | |
| | Add Rimmicu Vilcea | (T, 440), (R, 604) | 604 | 411 | 193 |
| | Add Pitesti | (T, 440), (R, 604), (P, 503) | 503 | 403 | 100 |
| 6 | Expand Timisoara | | | | |
| | Add Arad | (R, 604), (P, 503), (A, 595) | 595 | 229 | 366 |
| 7 | Expand Pitesi | | | | |
| | Add Bucharest | (R, 604), (A, 595), (B, 504) | 504 | 504 | 0 |
| 8 | Expand Bucharest | | | | |
| | At Goal State | (R, 604), (A, 595) | - | - | - |

2. The heuristic path algorithm is a best-first search in which the objective function is $f(n) = (2 - w)g(n) + wh(n)$. For what values of $w$ is this algorithm guaranteed to be optimal? What kind of search does this perform when $w = 0$? When $w = 1$? When $w = 2$?

   **Answer:**

   The algorithm is guaranteed to be optimal for $0 \leq w \leq 1$, since scaling $g(n)$ by a constant has no effect on the relative ordering of the chosen paths, but, if $w > 1$ then it is possible the $wh(n)$ will overestimate the distance to the goal, making the heuristic inadmissible. If $w \leq 1$, then it will reduce the estimate, but it is still guaranteed to underestimate the distance to the goal state.

| $w$ | $f(n)$ | Algorithm |
|---|---|---|
| $w = 0$ | $f(n) = 2g(n)$ | Uninformed best-first search |
| $w = 1$ | $f(n) = g(n) + h(n)$ | A* search |
| $w = 2$ | $f(n) = 2h(n)$ | Greedy best-first search |

3. Propose an admissible h function for this problem that is better than $h \equiv 0$.

**Answer:**

One of the possible admissible heuristics is:

$$h(state) = 3 - |(number\ of\ discs\ on\ peg\ B) - (number\ of\ discs\ on\ peg\ C)|$$

Obsiously $h(goal) = 0$ for both goals. Moreover $h(stat) \leq number\ of\ steps\ to\ achieve\ goal$.

4. Algorithms A* does not terminate until a goal node is selected for expansion. However, a path to a goal node might be reached long before that node is selected for expansion. Why not terminate a soon as a goal node has been found? Illustrate your answer with an example.

**Answer:**

One cannot stop after the a goal node is found while expanding its predecessor since that can lead to a suboptimal solution. For instance, consider the simple "diamond" search graph with the given costs and heuristics in Figure 4.

Consider the trace of A* for this problem shown in the table below. If we stopped searching after encountering the goal state (D) for the first time after expanding node C in step 3, then we would have been lead to believe that the best path to the goal had a cost of 7. However, by waiting until the the goal state has the lowest cost of nodes in the frontier, we guarantee that the lowest cost path through node B is found.

| | Action | Open Nodes | $f(n)$ | $g(n)$ | $h(n)$ |
|---|---|---|---|---|---|
| 1 | Initial State | (A, 6) | 6 | 0 | 6 |
| 2 | Expand A | | | | |
| | Add B | (B, 6) | 6 | 3 | 3 |
| | Add C | (B, 6), (C, 5) | 5 | 2 | 3 |
| 3 | Expand C | | | | |
| | Add D | (B, 6), (D, 7) | 7 | 7 | 0 |
| 4 | Expand B | | | | |
| | Replace D | (D, 6) | 6 | 6 | 0 |
| 5 | Expand D | | | | |
| | At Goal State | | - | - | - |

5. For the sliding block puzzle, specify a heuristic function $H$, and show the search tree produced by algorithm A* using this function. Show the first 10 nodes expanded.

**Answer:**

A simple heuristic is taking half the sum of the number of white tiles to the right of each black tile, $w_i$ since it will take at least one move to move past each black tile. Dividing by half ensures that we never overestimate since it's possible for a white tile to hop two black tiles in a single move.

$$h(n) = \frac{1}{2} \sum_i w_i$$

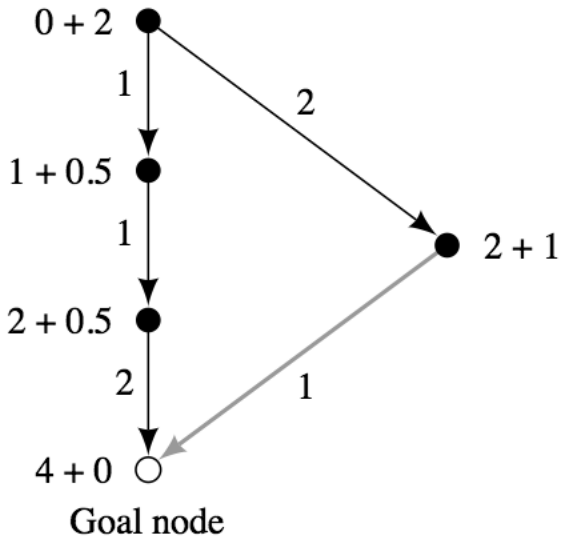


Figure 1: Example where stopping after finding a goal state for the first time fails for A$^*$ search.

| | Action | Open Nodes | $f(n)$ | $g(n)$ | $h(n)$ |
|---|---|---|---|---|---|
| 1 | Initial State | (BBBWWWE, 4.5) | 4.5 | 0 | 4.5 |
| 2 | Expand BBBWWWE | | | | |
| | Add BBBWWEW | (BBBWWEW, 5.5), (BBBWEWW, 5.5) | 5.5 | 1 | 4.5 |
| | Add BBBWEWW | (BBBEWWW, 5.5) | 5.5 | 1 | 4.5 |
| | Add BBBEWWW | | 6.5 | 2 | 4.5 |
| 3 | Expand BBBWWEW | | | | |
| | Add BBEWWBW | (BBBWEWW, 5.5), (BBBEWWW, 6.5) | 6.5 | 3 | 3.5 |
| | | (BBEWWBW, 6.5) | | | |
| 4 | Expand BBBWEWW | | | | |
| | Add BBEWBWW | (BBBEWWW, 6.5), (BBEWWBW, 6.5) | 6 | 2 | 4 |
| | Add BEBWBWW | (BBEWBWW, 6.0), (BEBWBWW, 7.0) | 7 | 3 | 4 |
| 5 | Expand BBEWBWW | | | | |
| | Add BBWEBWW | (BBBEWWW, 6.5), (BBEWWBW, 6.5) | 7 | 3 | 4 |
| | Add BBWWBEW | (BEBWBWW, 7.0), (BBWEBWW, 7.0) | 7.5 | 4 | 3.5 |
| | Add EBBWBWW | (BBWWBEW, 7.5), (EBBWBWW, 7.0) | 7 | 3 | 4 |
| 6 | Expand BBBEWWW | | | | |
| | Add BBEBWWW | (BBEWWBW, 6.5), (BEBWBWW, 7.0) | 7.5 | 3 | 4.5 |
| | Add BEBBWWW | (BBWEBWW, 7.0), (EBBWBWW, 7.0) | 7.5 | 3 | 4.5 |
| | Add EBBBWWW | (BBWWBEW, 7.5), (BBEBWWW, 7.5) | 8.5 | 4 | 4.5 |
| | | (BEBBWWW, 7.5), (EBBBWWW, 8.5) | | | |
| 7 | Expand BBEWWBW | | | | |
| | Add BEBWWBW | (BEBWBWW, 7.0), (BBWEBWW, 7.0) | 7.5 | 4 | 3.5 |
| | Add EBBWWBW | (EBBWBWW, 7.0), (BBWWBEW, 7.5) | 7.5 | 4 | 3.5 |
| | Add BBWEWBW | (BBEBWWW, 7.5), (BEBBWWW, 7.5) | 7.5 | 4 | 3.5 |
| | Add BBWWEBW | (EBBBWWW, 8.5), (BEBWWBW, 7.5) | 7.5 | 4 | 3.5 |
| | | (EBBWWBW, 7.5), (BBWEWBW, 7.5) | | | |
| | | (BBWWEBW, 7.5) | | | |

6. Prove the following properties on algorithm A$^*$.

   (a) A heuristic function is monotone if for every node $n$ and its child node $n'$

   $$h(n) \leq h(n') + c(n, n')$$

   Prove that is $h_1$ and $h_2$ are both monotone, so also is $h = \max(h_1, h_2)$
   **Answer:**

   $$\begin{aligned} h(n) &= \max(h_1(n), h_2(n)) \\ &\leq \max(h_1(n') + c(n, n'), h_2(n') + c(n, n')) \\ &\leq \max(h_1(n'), h_2(n')) + c(n, n') \\ &\leq h(n') + c(n, n') \end{aligned}$$

   (b) Prove that if $h$ is monotone then it is also admissible.
   **Answer:**
   At the goal node, $h(n^*) = 0$, so one step away from the goal, $h(n) \leq h(n') + c(n, n') \leq c(n, n^*)$. By induction, $h(n) \leq c(n, n^*)$ from any node, thus it is admissible since it always underestimates the path cost to the goal state.

(c) Prove that is the heuristic function is monotone then A* will never reopen any nodes.
**Answer:**

$$\begin{aligned} f(n) &= g(n) + h(n) \\ &\leq g(n) + h(n') + c(n, n') \\ &\leq g(n) + c(n, n') + h(n') \\ &\leq g(n') + h(n') \\ &\leq f(n') \end{aligned}$$

Thus, we have proven that $f(n)$ is always greater at a successor node, so once a node has been visited, it cannot be visited again at a smaller cost and thus will never be reopened.

(d) Prove or give a counter example: if for every node $n$, $h_1(n) \geq h_2(n)$, then A* with $h_1$ always expands less nodes than A* with $h_2$
**Answer:**
Well, since $h_1(n) \geq h_2(n)$ we could trivially say that if the two heuristics are equal then $h_2$ will not result in expanding less nodes than $h_1$, but I assume that is not the intent of the question, so let's consider $h_1(n) < h_s(n)$.
For any given node $n$, it will be placed in the open list with a values of $f(n) = g(n) + h(n)$. Since $f_1(n) \geq f_2(n)$, every node evaluated by $f_1$ functions will have an estimated at least as large as $f_2$. Since the node with the smallest value is chosen for expansion, one of two possibilities exit

- $f_1(n) \geq f_2(n)$ and $n$ has the smallest value. In this case the same node is expanded by both heuristics.
- $f_1(n) \geq f_2(n)$ and there exits a node $n'$ in the open list such that $f_1(n) \geq f_1(n')$ and $f_2(n) \leq f_2(n')$. In this case, A* will select node $n'$ first, and, if that node lead to a solution state, node $n$ will not be expanded. Note that the inequlity for $f_2$ is valid since it just shows that $f_2$ under estimated the path cost to the goal from node $n'$.

Thus we have shown that A* with heuristic $h_1$ con only expand as many or less nodes than $h_2$.

(e) Let $h$ be an admissible function and let $f(n) = w \cdot g(n) + (1 - w)h(n)$, $0 \leq w \leq 1$. Will A* with $f$ find an optimal solution when $w = \frac{1}{4}$?, $w = \frac{1}{2}$?, $w = \frac{3}{4}$? Can you provide a general rule? (note, that $f$ here denotes an arbitrary evaluation function, not necessarily an exact one).
**Answer:**
It is not guaranteed to find an optimal solution for $w = \frac{1}{4}$ and is guaranteed for $w = \frac{1}{2}$ and $\frac{3}{4}$. The general rule is, for constants $a$ and $b$ where $f(n) = a \cdot g(n) + b \cdot h(n)$, if $\frac{a}{b} \geq 1$ then A* is guaranteed to find an optimal solution.
For this problem, $a = w, b = 1 - w$, so, solving the ratio,

$$\begin{aligned} \frac{a}{b} &\geq 1 \\ \frac{w}{1-w} &\geq 1 \\ w &\geq 1 - w \\ w &\geq \frac{1}{2} \end{aligned}$$

7. Extra-credit problem, no solution.

8. On page 108, we defined the relaxation of the 8-puzzle in which a tile can move from square A to square B if B is blank. The exact solution of this problem defined **Gaschnig's heuristic**. Explain

why Gashnig's heuristic is at least as accurate as $h_1$ (misplaced tiles), and show cases where it is more accurate than both $h_1$ and $h_2$. Can you suggest a way to calculate Gaschnig's heuristic efficiently?

**Answer:**

Since the misplaced tile heuristic can place any tile in any other position in one move and Gaschnig's heuristic can only move a tile to the black spot, Gaschnig's will always take *at least* one move to get a tile to its proper position, and may require two moves if the blank location is in its final position and tiles are still misplaced. Thus, Gaschnig's heuristic *dominates* $h_1$ since it is always returns more or an equal number of moves.

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 8 | | 4 |
| 7 | 6 | 5 |

Target

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 6 | | 4 |
| 8 | 7 | 5 |

State

True Shortest Path Cost: 4
Gashnig's: 4
Misplaced Tiles: 3

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 8 | | 4 |
| 7 | 6 | 5 |

Target

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 7 | | 4 |
| 8 | 6 | 5 |

State

True Shortest Path Cost: $> 3$
Gashnig's: 3
Manhatten: 2

Some possible pseudo-code for computing Gaschnig's heuristic is:

```
moves = 0
while not in goal state:
  if blank in goal position:
    swap blank with any mismatch
  else
    swap blank with matched tile
  moves++
return moves
```