# CS 540: Introduction to Artificial Intelligence
# Homework Assignment 4

Matthew Klebenow
CSL: klebenow
Section 1
0 Late Days Used

November 30, 2012

# Contents



# List of Tables



# List of Figures

# 1 Search Algorithms

Table 1 is a $5 \times 5$ maze filled with mouse traps ($\tau$). There is a poor hungry mouse ($\mu$) at one corner of the maze which smells the presence of cheese ($\xi$) somewhere in the maze but does not know the exact location. Your task is to help the mouse find its food (fixed in square S) without getting trapped anywhere. Assume that the mouse can sense the presence of a mouse trap in neighboring cells. The mouse can only move in vertical and horizontal directions. It cannot move diagonally. Assume the successor function will cause *legal moves to be examined in a clockwise order*: up, right, down, left. Note that not all of these moves may be possible from a given square.

Table 1: Maze

| | | | | |
|---|---|---|---|---|
| A | B | C | D | E |
| $\mu$ | | $\tau$ | | |
| F | G | H | I | J |
| | $\tau$ | | | |
| K | L | M | N | O |
| | | | | $\tau$ |
| P | Q | R | S | T |
| $\tau$ | | $\tau$ | $\xi$ | |
| U | V | W | X | Y |
| | | | | |

## 1.1 Depth-First Search

Using Depth-First Search, list the squares in the order they are expanded (including the goal node if it is found). Square A is expanded first (hint: State B will be examined next). Assume cycle checking is done so that a node is not generated in the search tree if the grid position already occurs on the path from this node back to the root node (i.e., Path Checking DFS). Write down the list of states you expanded in the order they are expanded. Write down the solution path found (if any), or explain why no solution is found.

### 1.1.1 List of expanded states

A complete list of expanded states for the Path Checking Depth First Search is depicted in Table 2.

Table 2: Expanded states for Section 1.1

| Iteration | State |
|---|---|
| 1 | A |
| 2 | AB |
| 3 | AF |
| 4 | AFK |
| 5 | AFKL |
| 6 | AFKLM |
| 7 | AFKLMH |
| 8 | AFKLMHI |
| 9 | AFKLMHID |
| 10 | AFKLMHIDE |
| 11 | AFKLMHIDEJ |
| 12 | AFKLMHIJ |
| 13 | AFKLMHIJE |
| 14 | AFKLMHIJED |
| 15 | AFKLMN |
| 16 | AFKLMNI |
| 17 | AFKLMNID |
| 18 | AFKLMNIDE |
| 19 | AFKLMNIDEJ |
| 20 | AFKLMNIJ |
| 21 | AFKLMNIJE |
| 22 | AFKLMNIJED |
| 23 | AFKLMNIH |
| 24 | AFKLMNS |

## 1.2 Iterative Deepening Search

Using Iterative Deepening Search, for each depth list the squares in the order they are expanded until a solution is reached. Use the same cycle checking as in §1.1.

### 1.2.1 List of expanded states

1. A
2. ABF
3. ABFK
4. ABFKL
5. ABFKLMQ
6. ABFKLMQHNV
7. ABFKLMQHNVISUW

In the last step, $I$ will be expanded to include $D$ and $J$. Next $S$ will be expanded and we will have reached the goal state.

## 1.3 Manhattan Distance heuristic function

Let each move (up/down/left/right) of the mouse have cost 1. Consider the heuristic function $h(n) = \left|x_n - x_g\right| + \left|y_n - y_g\right|$, where the grid square associated with node $n$ is at coordinates $(x_n, y_n)$ on the board, and the goal node is at coordinates $\left(x_g, y_g\right)$. That is, $h(n)$ is the Manhattan distance between $n$ and the goal. Is $h(n)$ admissible? Why?

### 1.3.1 Answer

We define an admissible heuristic function to be one that never over-estimates the true cost from any given state to the goal state. The best possible situation we can place ourselves in is one where our current state has a clear shot to the goal; in this case, the heuristic would be equivalent to the true cost. In a worse situation where the mouse must avoid a mousetrap that is directly on its path to the goal, the heuristic under-estimates the true cost (the heuristic cost would be equivalent to charging straight through the mousetrap). As such, the heuristic function $h(n)$ is admissible.

## 1.4 A* Search

Regardless of your answer to 1.3, perform A* Search using the heuristic function $h(n)$ with a slight modification that $h(n) = \infty$ if node $n$ has a mouse trap. In the case of ties, expand states in alphabetical order. List each square in the order they are added to the OPEN list, and mark it with $f(n) = g(n) + h(n)$ (show $f$, $g$, and $h$ separately). Also, list the squares in the order they are expanded (including the goal node) and list the solution path found (if any), or explain why no solution is found.

### 1.4.1 Solution

The process followed to reach the solution is shown explicitly in Table 3. The final solution path found is AFKLMNS.

Table 3: Solution Steps for 1.4

| Step | OPEN | $f = g + h$ | $g$ | $h$ | Expanded |
|---|---|---|---|---|---|
| 1 | A | 6 | 0 | 6 | |
| 2 | | | | | A |
| 3 | B | 6 | 1 | 5 | |
| 4 | F | 6 | 1 | 5 | |
| 5 | | | | | B |
| 6 | | | | | F |
| 7 | K | 6 | 2 | 4 | |
| 8 | | | | | K |
| 9 | L | 6 | 3 | 3 | |
| 10 | | | | | L |
| 11 | M | 6 | 4 | 2 | |
| 12 | Q | 6 | 4 | 2 | |
| 13 | | | | | M |
| 14 | H | 8 | 5 | 3 | |
| 15 | N | 6 | 5 | 1 | |
| 16 | | | | | Q |
| 17 | V | 8 | 5 | 3 | |
| 18 | | | | | N |
| 19 | I | 8 | 6 | 2 | |
| 20 | S | 6 | 6 | 0 | |
| 21 | | | | | S |

## 1.5 A* Search, heuristic function $h_2(n)$

Repeat 1.4 using a new heuristic function $h_2(n)$, which is defined using the old heuristic $h(n)$ as shown in Table 4. A new mapping of the maze is shown in Table 5 with each

Table 4: New heuristic function $h_2(n)$

| h(n) | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | infty |
|---|---|---|---|---|---|---|---|---|---|---|
| h2(n) | 0 | 1 | 3 | 1 | 3 | 2 | 3 | 2 | 4 | infty |

cell labeled with its name, attribute (mousetrap, cheese, mouse), and heuristic function values $h(n); h_2(n)$.

Table 5: The maze from § 1 with values for both heuristic functions.

| A | B | C | D | E |
|---|---|---|---|---|
| $\mu$ | | $\tau$ | | |
| 6;3 | 5;2 | $\infty;\infty$ | 3;1 | 4;3 |
| F | G | H | I | J |
| | $\tau$ | | | |
| 5;2 | $\infty;\infty$ | 3;1 | 2;3 | 3;1 |
| K | L | M | N | O |
| | | | | $\tau$ |
| 4;3 | 3;1 | 2;3 | 1;1 | $\infty;\infty$ |
| P | Q | R | S | T |
| $\tau$ | | $\tau$ | $\xi$ | |
| $\infty;\infty$ | 2;3 | $\infty;\infty$ | 0;0 | 1;1 |
| U | V | W | X | Y |
| | | | | |
| 4;3 | 3;1 | 2;3 | 1;1 | 2;3 |

### 1.5.1 Solution

The process followed to reach the solution is shown explicitly in Table 6. The final solution path found is AFKLMNS.

Table 6: Solution steps for 1.5

| Step | OPEN | $f = g + h$ | $g$ | $h$ | Expanded |
|---|---|---|---|---|---|
| 1 | A | 3 | 0 | 3 | |
| 2 | | | | | A |
| 3 | B | 3 | 1 | 2 | |
| 4 | F | 3 | 1 | 2 | |
| 5 | | | | | B |
| 6 | | | | | F |
| 7 | K | 5 | 2 | 3 | |
| 8 | | | | | K |
| 9 | L | 4 | 3 | 1 | |
| 10 | | | | | L |
| 11 | M | 7 | 4 | 3 | |
| 12 | Q | 7 | 4 | 3 | |
| 13 | | | | | M |
| 14 | H | 6 | 5 | 1 | |
| 15 | N | 6 | 5 | 1 | |
| 16 | | | | | H |
| 17 | I | 9 | 6 | 3 | |
| 18 | | | | | N |
| 19 | S | 6 | 6 | 0 | |
| 20 | | | | | S |

# 2 Hill Climbing

We would like to solve the n-Queens problem using a greedy hill-climbing algorithm. The n-Queens problem required that we place all n queens on an $n \times n$ board so that none of them can attack any other by the rules of chess. This means there cannot be 2 queens in the same row, column, or diagonal. Here we define each state to correspond to a complete assignment of a row number from 1 to n to each of the n column variables $C_1$ through $C_n$. The successor operator Succ(s) generates all neighboring states of s, which we will define as all total assignments which differ by exactly one variable's row value. So, for example, given $n = 2$ the state with assignments $\{C_1 = 1, C_2 = 2\}$ has two neighboring states $\{C_1 = 2, C_2 = 2\}$ and $\{C_1 = 1, C_1 = 1\}$ corresponding to moving either the queen in the first row or the queen in the second. When breaking ties, order the variables from left to right, and the values for the variables from bottom to top of the board and pick the neighboring state that comes first in the ordered list of states.

## 2.1 Successor function and search space

If you have $n$ rows and $n$ column variables, how many neighboring states does the Succ$(s)$ function produce? What is the total size of the search space?

### 2.1.1 Answer

For any given succession state, we may change any $C_i$ to a value from 1 to n, excluding the current value of $C_i$. We have $n$ columns to work with, and $n-1$ possible positions to change each column to. Therefore, the successor function will produce $n \times (n-1)$ neighboring states. The total size of the search space must then be the aforementioned branching factor times the depth of the problem space. Since the depth is not known a priori, the total search space size is unknown.

## 2.2 State evaluation function

Define an evaluation function for the states such that the goal state has the highest value when the evaluation function is applied to it.

### 2.2.1 Answer

Let us define an evaluation function based on the number of queens attacking each other, $x$. If we evaluate every state with a value of $-x$, the goal state will have a maximum value of 0.

## 2.3 6-Queens Plateau

Consider the 6-Queens problem and come up with a non-goal state that is on a plateau in our hill-climbing space using the evaluation function you defined above.

### 2.3.1 Answer

Similar to the 8-Queens example shown in class that is also on a plateau, the state shown in Figure 1 is a non-goal plateau state in our hill-climbing space using the evaluation function defined in § 2.2.1.

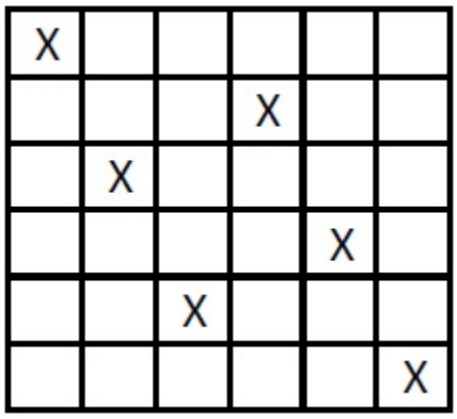

Figure 1: Plateau state

# 3 Iterative Deepening

Consider a search space that consists of a tree with branching factor $b > 1$, so each node has exactly $b$ unique children. The root node is at depth 0. The first node checked on level $d > 0$ is a solution to the search. How many nodes *total* will an iterative deepening search check before it halts?

## 3.1 Answer

For the first iteration, we must check the root node. This action requires 1 check. The next iteration, we must check the root node and it's children. This action requires $1 + b$ checks. The next iteration, we must check: the root (1), its children ($b$), and all of its children's children ($b^2$). The total number of checks for this iteration is therefore $1 + b + b^2$. This pattern will continue until the iteration reaches the depth $d > 0$ and checks $1 + b + b^2 + b^3 + \cdots + b^{d-2} + b^{d-1} + 1$ nodes. Note that the 1 that terminates the sequence is because the first node checked on that iteration (the iteration at level $d$) is the solution to the search. So, the *total* number of nodes checked by the iterative deepening search is the sum of checks performed across all iterations. A full mathematical interpretation of this result is given in Equation 1.

$$\begin{aligned}\text{Total checks} = 1 + (1 + b) + \left(1 + b + b^2\right) + \left(1 + b + b^2 + b^3\right) + \left(1 + b + b^2 + b^3 + b^4\right) + \cdots \\ \cdots + \left(1 + b + b^2 + b^3 + b^4 + \cdots + b^{d-2} + b^{d-1} + 1\right) \quad (1)\end{aligned}$$