Lab Report A2
CS4300
Derek Heldt-Werle 1,3,5
Matthew Lemon 2,4,6
9/8/16

1. Introduction

Author: Derek Heldt-Werle

This lab will study the statistics and complexity of using variations of A∗ search on the Wumpus World problem. The measure of complexity is the total number of nodes generated during a search; this means that when a node is expanded, all its children are added to the tree (unless their state has already been created). For A∗ search we use the heuristic of Manhattan distance between the current state and the goal state. We will then compare two ways to insert nodes into the priority queue for the frontier: (1) insert before greater than or equal cost states, (2) insert after less than or equal cost states. We will run 2000 trials using random boards with Wumpus and 20% probability of a pit in each (non-start) cell. We will then test the hypothesis that option 1 A∗ search (given above) is 10% better than option 2 at the 95 % confidence level.

Through testing, we intend to answer a few questions:

Does insertion into the priority queue lead to a 10% performance gain when we insert before greater than or equal cost states versus insertion after less than or equal cost states at the 95% confidence level?

What is the mean number of search tree nodes produced by A* options 1 and 2 on these problems?

What is the variance in the number of search tree nodes produced by A* options 1 and 2 on these problems?

2. Method

Author: Matthew Lemon

We ran an A* search algorithm on 2000 randomly generated Wumpus World Boards.

Our cost analysis for the A* search was g + h, defined below
- g - level of state in the tree (how far from the root the node is)
- h - Manhattan distance heuristic from current node to goal node

When running the algorithm we first generate the root node (1,1,0) and then add it to the priority queue. Then, while the priority queue is not empty, we loop doing these steps:

1. Remove a node from the priority queue and mark it as visited
2. Add the node to the nodes out parameter
3. Check if our current state is on BOTH the WUMPUS and GOLD in which case there is no solution and we return.
4. Check if we arrived at our goal at which point we build the solution states by following the path of parents back up the tree to the root node and then return.
5. Get the child states of the current node and add them to the priority queue
    a. They are added in order of actions being FORWARD, then RIGHT, then LEFT
    b. Before we add the children we first check if they have already been visited, or if they are a PIT, or if they are a WUMPUS in which case we do not add the children.

3. Verification of Program

Author: Derek Heldt-Werle

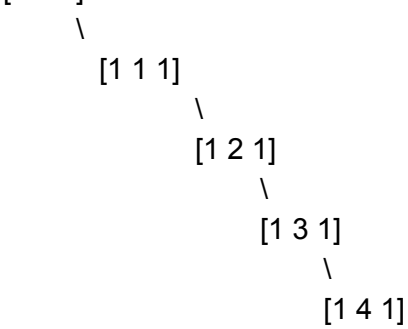
(a)

2	0	0	0
0	0	3	0
0	0	0	0
0	0	0	0

Option 1:

Solution: 1,1,0 -> 2,1,0 -> 3,1,0 -> 4,1,0

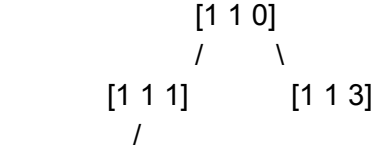
Tree:

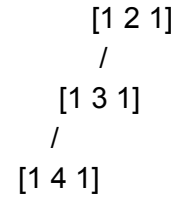


Option 2:

Solution: 1,1,0 -> 2,1,0 -> 3,1,0 ->4,1,0

Tree

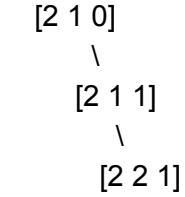




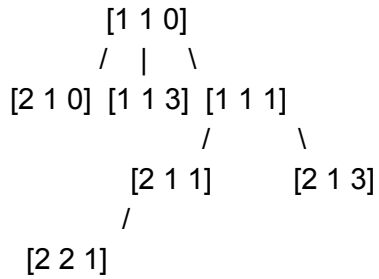
Matlab produced the same output.

0	0	0	0
0	0	3	0
1	2	0	0
0	0	0	0

Option 1:
Solution: 1,1,0 -> 2,1,0 -> 2,1,1 -> 2,2,1
Tree:



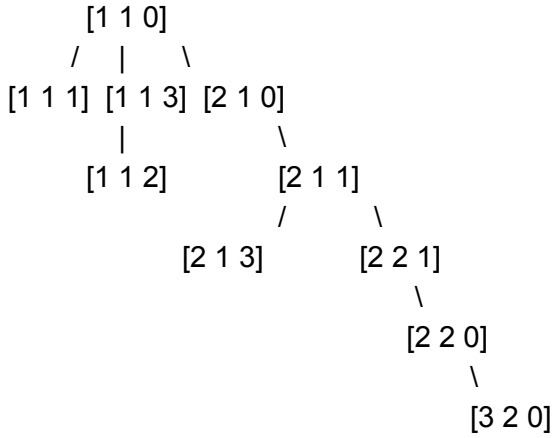
Option 2:
Nodes 1,1,0 -> 2,1,0 -> 2,1,1 -> 2,2,1
Tree:



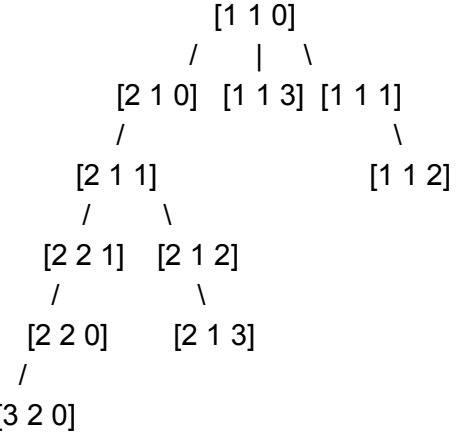
Matlab produced the same output.

0	0	0	0
1	1	0	0
1	0	2	0
0	0	3	0

Option 1:
Solution: 1,1,0 -> 2,1,0 -> 2,1,1 -> 2,2,1 -> 2,2,0 -> 3,2,0
Tree



:
Option 2:
Solution: 1,1,0 -> 2,1,0 -> 2,1,1 -> 2,2,1 -> 2,2,0 -> 3,2,0



(b)
For both option 1 and 2 the min tree size is 1 node. That one node in the search tree is simply the root, in the situation in which the gold is located in 1,1.

For both option 1 and 2 the max tree size is 60 nodes given no pits and a wumpus. In this situation every single node is explored before landing on the gold. This is obtained by having 16 nodes and 4 different directions, thus giving us a total of 60.

In our data the actual minima was indeed 1, but the maxima was 52. With the chance of the gold being in the first position being .625 per run, it makes sense the minima was encountered. The maxima also makes sense with there being three pits on average and a wumpus in 1 there is only 52 actions to make (13 * 4).

4. Data and Analysis

Author Matthew Lemon

Nodes in search tree
Table data format = Insertion Type 1 : Insertion Type 2

5 : 6	6 : 12	7 : 18	8 : 24
4 : 5	5 : 10	6 : 15	7 : 20
3 : 4	4 : 8	5 : 12	6 : 16
1 : 1	2 : 2	3 : 3	4 : 4

Figure 1: Number of Nodes Generated When Goal is at [x;y]
Wumpus World A*, 2000 trials:

Insertion Type 1

Mean	9.609
Variance	77.6519
Confidence Interval	9.2849 - 9.9331

Figure 2: mean, variance, and confidence interval of insertion option 1

Insertion Type 2

Mean	12.808
Variance	98.9786
Confidence Interval	12.4421 - 13.1739

Figure 3: mean, variance, and confidence interval of insertion option 2

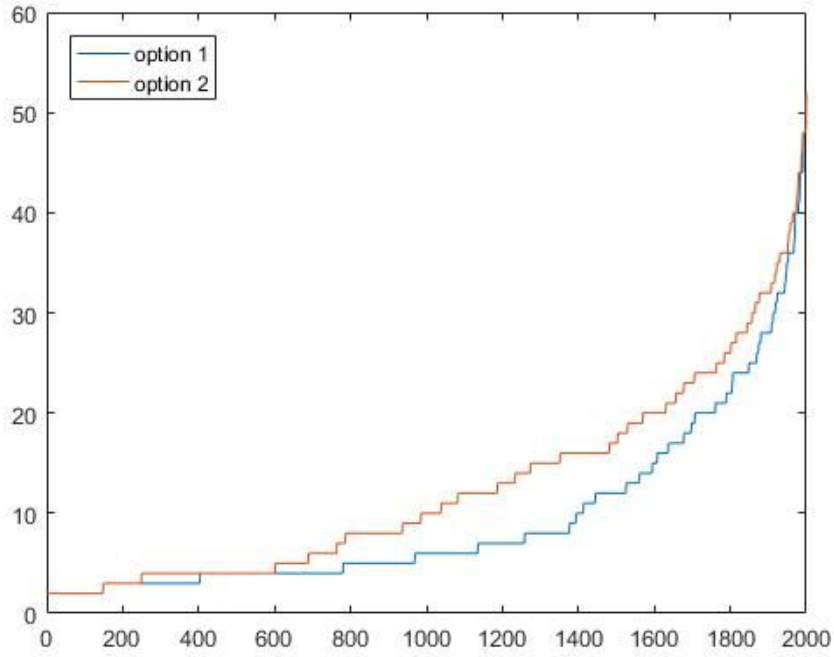


Figure 4: option 1 and option 2 plot (x = number of nodes in tree, y = trial number)

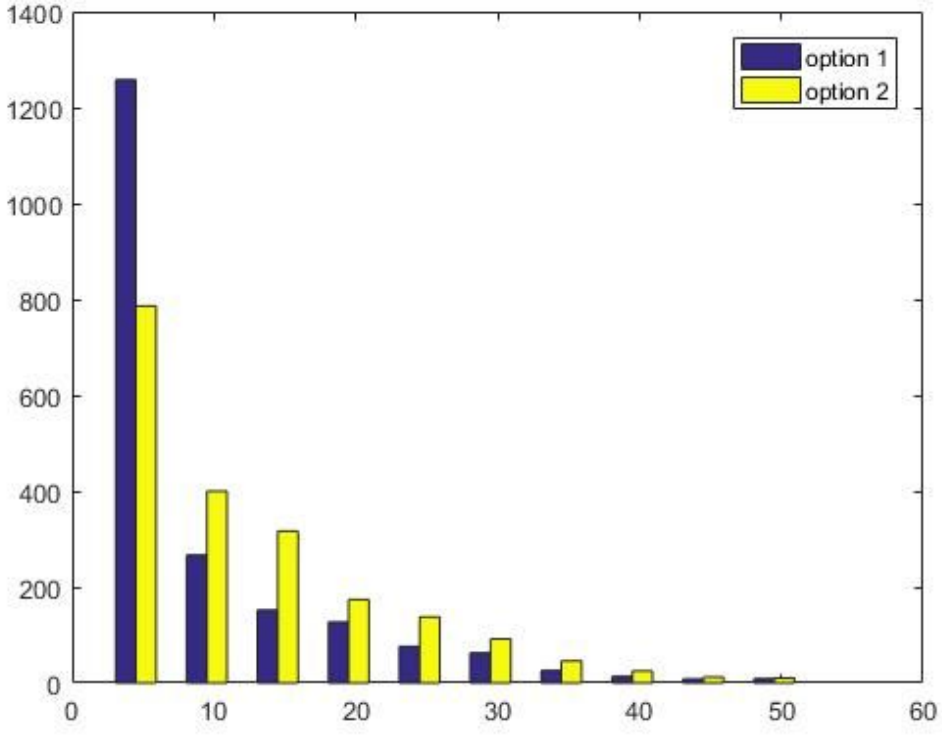


Figure 5: option 1 and 2 histogram (y = number of trials, x = number of nodes in tree)

5. Interpretation

Author: Derek Heldt-Werle

To answer the questions we addressed in the introduction we can begin by looking at figure 1 and figure 2. These figures describe the mean, variance, as well as the 95% confidence interval for the A* algorithm. From examining these we can see that for option 1 the mean number of nodes generated was 9.61, a variance of 77.65, and a confidence interval of 9.28 - 9.93. For option 2 we have a mean of 12.8, a variance of 98.97 with a confidence interval of 12.4421 - 13.173. Based on these we can then conclude that our hypothesis was incorrect, as the second option was worse than the 10% performance hit we expected it to take (~26%).

6. Critique

Author Matthew Lemon

During this lab we gained a lot more experience working specifically with matlab and learning how to work with its data structures and functions. We had created our own class to represent Nodes and our tree structure which although was more work, helped us gain a better understanding of Matlab. We also created our own Priority Queue implementation and found ways of creating arrays of objects.Our most important discovery was the Matlab debugger, which was an invaluable tool that we will continue to use heavily in the future.

7. Log

Author Matthew Lemon & Derek Heldt-Werle

Coding Portion (Worked together): 10 hours

Report (Derek): 2 hours

Report (Matt): 2 hours