## Introduction

In the given assignment, students were set the task to implement an AI solving algorithm to the game, Peg Solitaire. This was done through adding to the Peg Solitaire C implementation by Maurits van der Schee. The main goal of the assignment is to be able to develop a program capable of finding a solution with the least pegs to any given board, ideally a single remaining peg, through valid moves. Aside from this, another goal was to provide statistics from the solver for each board, namely: expanded nodes, generated nodes, solution length, number of remaining pegs, expanded nodes/seconds, and time to execute.

Peg Solitaire falls under a category of problems known as the NP-Complete Problems, to which so far, the best algorithms run in exponential time as its size increases.

The assignment tackles the use of Graphs, Graphing Algorithms, and Traversal over these Graphs. In particular, this assignment makes use of Depth First Search to find solutions.

## Method

1. Implement the following pseudocode into the Peg Solitaire solving program.

a.

**Algorithm 1** AI Peg Solitaire Algorithm

```
1:  procedure PegSolver(start, budget)
2:      n ← initNode(start)
3:      stackPush(n)
4:      remainingPegs ← numPegs(n)
5:      while stack ≠ empty do
6:          n ← stack.pop()
7:          exploredNodes ← exploredNodes + 1
8:          if numPegs(n) < remainingPegs then          ▷ Found a better solution
9:              saveSolution(n)
10:             remainingPegs ← numPegs(n)
11:         end if
12:         for each jump action a ∈ [0, m) × [0, m) × {Left, Right, Up, Down} do
13:             if a is a legal action for node n then
14:                 newNode ← applyAction(n, a)          ▷ Create Child node
15:                 generatedNodes ← generatedNodes + 1
16:                 if won(newNode) then                 ▷ Peg Solitaire Solved
17:                     saveSolution(newNode)
18:                     remainingPegs ← numPegs(newNode)
19:                     return
20:                 end if
21:                 if newNode.state.board seen for the first time then   ▷ Avoid duplicates
22:                     stackPush(newNode)               ▷ DFS strategy
23:                 end if
24:             end if
25:         end for
26:         if explored_nodes ≥ budget then
27:             return                                   ▷ Budget exhausted
28:         end if
29:     end while
30: end procedure
```

b. Adding to the algorithm, each new node pointer was stored into a linked list for memory-management purposes.

2. The following was entered into the terminal to obtain statistics: "./pegsol *board* AI *budget*", where *board* ∈ [0, 8], and *budget* ∈ [10000, 100000, 1000000, 1500000].
   a. It should be noted that during experimentation, the optimisation flag "gcc -O3" was used over "gcc -g", however the latter was used in the submission.
   b. Additionally, due to the memory-intensive nature of the algorithm, particularly for boards 6, 7, and 8, higher budgets were allocated of 3000000 and 5000000 in an attempt to obtain better insight and statistics for the boards.

## Results

*Table 1: Board 0: 3 pegs*

| Budget | Number of Pegs Left | Generated Nodes | Number of Pegs Left | Expanded Nodes/Second | Execution Time |
|---|---|---|---|---|---|
| 10,000 | 2 | 2 | 1 | 10 | 0.187500 |
| 100,000 | 2 | 2 | 1 | 6 | 0.312500 |
| 1,000,000 | 2 | 2 | 1 | 6 | 0.296875 |
| 1,500,000 | 2 | 2 | 1 | 6 | 0.328125 |

*Table 2: Board 1: 4 pegs*

| Budget | Expanded Nodes | Generated Nodes | Number of Pegs Left | Expanded Nodes/Second | Execution Time |
|---|---|---|---|---|---|
| 10,000 | 3 | 3 | 1 | 9 | 0.328125 |
| 100,000 | 3 | 3 | 1 | 8 | 0.343750 |
| 1,000,000 | 3 | 3 | 1 | 10 | 0.296875 |
| 1,500,000 | 3 | 3 | 1 | 9 | 0.328125 |

*Table 3: Board 2: 7 pegs*

| Budget | Expanded Nodes | Generated Nodes | Number of Pegs Left | Expanded Nodes/Second | Execution Time |
|---|---|---|---|---|---|
| 10,000 | 7 | 8 | 1 | 20 | 0.343750 |
| 100,000 | 7 | 8 | 1 | 21 | 0.328125 |
| 1,000,000 | 7 | 8 | 1 | 22 | 0.312500 |
| 1,500,000 | 7 | 8 | 1 | 21 | 0.328125 |

*Table 4: Board 3: 17 pegs*

| Budget | Expanded Nodes | Generated Nodes | Number of Pegs Left | Expanded Nodes/Second | Execution Time |
|---|---|---|---|---|---|
| 10,000 | 3541 | 10282 | 1 | 10301 | 0.343750 |

| | | | | | |
|---|---|---|---|---|---|
| 100,000 | 3541 | 10282 | 1 | 10301 | 0.343750 |
| 1,000,000 | 3541 | 10282 | 1 | 10301 | 0.343750 |
| 1,500,000 | 3541 | 10282 | 1 | 11927 | 0.296875 |

*Table 5: Board 4: 32 pegs*

| Budget | Expanded Nodes | Generated Nodes | Number of Pegs Left | Expanded Nodes/Second | Execution Time |
|---|---|---|---|---|---|
| 10,000 | 1065 | 2418 | 1 | 2840 | 0.375000 |
| 100,000 | 1065 | 2418 | 1 | 3245 | 0.328125 |
| 1,000,000 | 1065 | 2418 | 1 | 3098 | 0.343750 |
| 1,500,000 | 1065 | 2418 | 1 | 3245 | 0.328125 |

*Table 6: Board 5: 36 pegs*

| Budget | Expanded Nodes | Generated Nodes | Number of Pegs Left | Expanded Nodes/Second | Execution Time |
|---|---|---|---|---|---|
| 10,000 | 10000 | 26495 | 4 | 58181 | 0.171875 |
| 100,000 | 100000 | 359818 | 3 | 246153 | 0.406250 |
| 1,000,000 | 1000000 | 4488464 | 2 | 333333 | 3.000000 |
| 1,500,000 | 1090275 | 4898609 | 1 | 329139 | 3.312500 |
| 3,000,000 | 1090275 | 4898609 | 1 | 324546 | 3.359375 |
| 5,000,000 | 1090275 | 4898609 | 1 | 314313 | 3.468750 |

*Table 7: Board 6: 44 pegs*

| Budget | Expanded Nodes | Generated Nodes | Number of Pegs Left | Expanded Nodes/Second | Execution Time |
|---|---|---|---|---|---|
| 10,000 | 10000 | 29368 | 5 | 25600 | 0.390625 |
| 100,000 | 100000 | 374378 | 4 | 142222 | 0.703125 |
| 1,000,000 | 1000000 | 4481233 | 3 | 164524 | 6.078125 |
| 1,500,000 | 1500000 | 7020668 | 3 | 160535 | 9.343750 |
| 3,000,000 | 3000000 | 14729710 | 3 | 153110 | 19.593750 |
| 5,000,000 | 5000000 | 24838155 | 3 | 73360 | 68.156250 |

*Table 8: Board 7: 38 pegs*

| Budget | Expanded Nodes | Generated Nodes | Number of Pegs Left | Expanded Nodes/Second | Execution Time |
|---|---|---|---|---|---|
| 10,000 | 10000 | 32469 | 4 | 25600 | 0.390625 |
| 100,000 | 100000 | 386440 | 2 | 123076 | 0.812500 |
| 1,000,000 | 1000000 | 4790308 | 2 | 158024 | 6.328125 |
| 1,500,000 | 1500000 | 7173504 | 2 | 158154 | 9.484375 |
| 3,000,000 | 3000000 | 15143755 | 2 | 97067 | 30.906250 |
| 5,000,000 | 5000000 | 26115880 | 2 | 86932 | 57.515625 |

*Table 9: Board 8: 40 pegs*

| Budget | Expanded Nodes | Generated Nodes | Number of Pegs Left | Expanded Nodes/Second | Execution Time |
|---|---|---|---|---|---|
| 10,000 | 10000 | 27562 | 6 | 27826 | 0.359375 |
| 100,000 | 100000 | 349921 | 4 | 123076 | 0.812500 |
| 1,000,000 | 1000000 | 4073028 | 4 | 177285 | 5.640625 |
| 1,500,000 | 1500000 | 6361454 | 4 | 172043 | 8.718750 |
| 3,000,000 | 3000000 | 13603617 | 4 | 295384 | 10.156250 |
| 5,000,000 | 5000000 | 23335070 | 4 | 153036 | 32.671875 |

*Figure 1: Effect of Budget on Solution Quality*

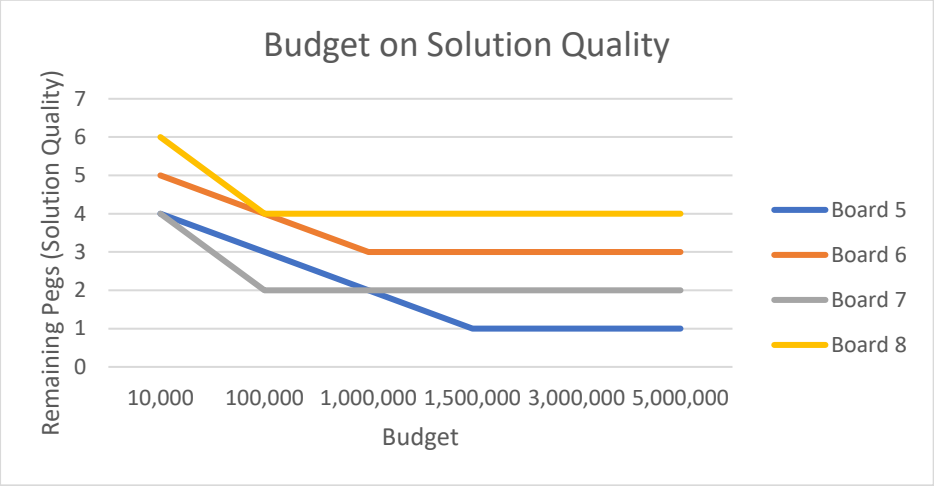


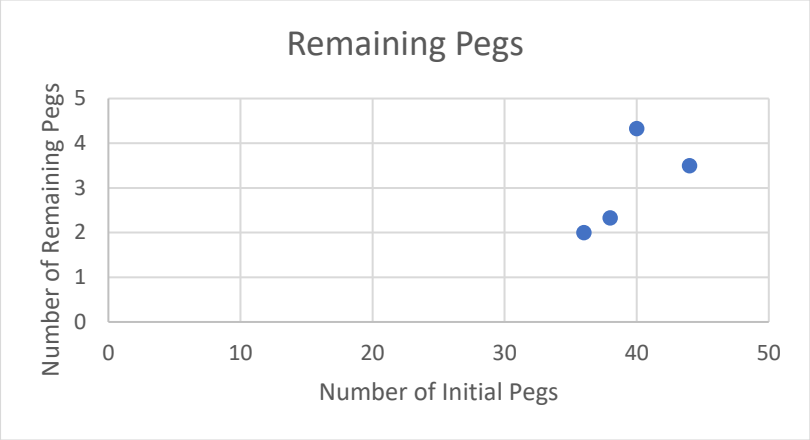

*Table 10: Remaining Pegs as Budget increases*

| | Remaining Pegs | | | |
|---|---|---|---|---|
| Budget | Board 5 | Board 6 | Board 7 | Board 8 |
| 10,000 | 4 | 5 | 4 | 6 |
| 100,000 | 3 | 4 | 2 | 4 |
| 1,000,000 | 2 | 3 | 2 | 4 |
| 1,500,000 | 1 | 3 | 2 | 4 |
| 3,000,000 | 1 | 3 | 2 | 4 |
| 5,000,000 | 1 | 3 | 2 | 4 |
| Average Pegs | 2 | 3.5 | 2.333333 | 4.333333 |

*Table 11: Remaining Pegs as Initial Pegs increase*

| Initial Pegs | Remaining Pegs |
|---|---|
| 36 | 2 |
| 44 | 3.5 |
| 38 | 2.333333 |
| 40 | 4.333333 |

*Figure 2: Remaining Pegs as a Function of Initial Peg Count*




## Discussion

As can be observed from the tables, boards zero through four require a relatively low budget to function completely. Onward however, from board five through 8, require a significant amount more budget to decrease the number of pegs. Therefore, further analysis on these tables will be centred on tables six to nine.

It can be seen through *Figure 1* that as budget increases, the number of remaining pegs decrease. Assuming perfect implementation of the AI Peg Solver algorithm, the remaining pegs should eventually theoretically decrease to 1. However, due to the exponential nature of NP-Complete Problems, obtaining more complete information would require an exponential amount more budget. In this regard, so far at least, the results are consistent with current algorithms for this classification of problems.

This can be related to *Figure 2*, where it can be seen that as initial peg count increases, the amount of remaining pegs increases as well. Theoretically, assuming a sufficient amount of memory, this should be 1. However, given a limited budget/memory, as previously stated, finding solutions with less pegs requires an exponential amount more memory.

## Recommendations

Data gathering was not done all at once. Given that the student's device was a laptop that adapts performance to battery levels, the execution time could vary massively. Going

forward, perhaps data gathering should be done with less expounding variables (constantly being plugged in, less applications open, etc.).

As can be seen from the results, the program is heavily memory-based, and is therefore limited by the amount of memory of the device. Going forward, ideally, a device with more memory could be used.

While the overall algorithm works perfectly fine to achieve a result, the student's implementation could be improved in the sense that memory management (frees) could be done on the fly, rather than in the end. While the time needed to execute the algorithm would perhaps be the same, the amount of memory available at any given moment would be more. This would allow the code to run better. Similarly, the overall base code could be optimised.

In regard to the experiment itself, it could be argued that the current meaning of a quality solution is inadequate. As of this paper, the highest quality solution is the first solution with the least number of pegs. This is however, without regard to future possibilities. One unfinished solution could have two pegs remaining but with no way of connecting them, whereas another could have three pegs with a winning path given sufficient memory. Moving forward, perhaps finding a way to assess how good a certain board state is, would be beneficial to the analysis of the board.