Algorithms and Data Structures

AI Solver for Peg Solitaire
Graph Search

Eisaku Daniel Tanaka

Introduction

Graph data structures are useful when representing a complex relationship of a set of objects where some pairs are connected by links. The individual objects will be called “nodes”, and the links are “edges”. A graph can be implicitly specified without explicitly defining all edges and nodes using an adjacency list or matrix. This experiment aims to program an AI solver for Peg Solitaire through the construction of an implicit graph and depth first search.

Peg Solitaire is a board game where one player can move a peg jumping on top of another adjacent peg, only if there is a blank cell across the adjacent peg to land. By proceeding these “jumps”, the player would win if they can end with 1 peg left on the entire board. The AI will lose if there is more than 1 peg left or if the program runs out of “budgets” which is specified by the user of the program. The board has a “state”, and it is updated each time the player makes a move. The AI solver program was run on 9 different layouts (levels) with increasing complexity to solve the problem. The initial number of pegs increase in the order of 3 (layout 0), 4 (layout 1), 7 (layout 2), 17 (layout 3), 32 (layout 4), 36 (layout 5), 44 (layout 6), 38 (layout 7), 50 (layout 8). The left diagram of figure 1 is one of the initial layouts of the game (Fitzpatrick, 2020).

Implicit graphs and depth first search (DFS) can be used to create an AI solver for Peg Solitaire. Starting at the initial state of the board, any legal move is able to update the board’s state. By generating all possible actions from that state, it is possible to create a tree structure where each node contains a possible state. Each possible configuration (state) is a tuple made of: $m \times m$ grid board, position of the cursor, and whether the peg under the cursor has been selected. Each node of the implicit graph would contain a state and the edges are the possible jumps (left, right, up, down).

DFS is performed in order to find the best solution. Making use of a stack and a hash table can allow for searching as far as possible along each branch from the root node, which contains the initial state. The following image (figure 1) shows an initial board on the left and the four possible states that this node can expand to (Fitzpatrick, 2020). All of these will become a child of the initial board and will be piled on to a stack. This process is repeated by popping and visiting the top node on the stack, until a solution is found or the number of expanded nodes exceed the budget. Cycles and expanding already seen states can be avoided through the use of a hash table. The best solution can be found by comparing the best known solution and current solution.

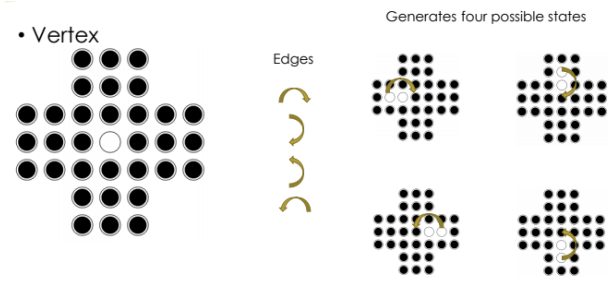

Figure 1

It is important to note that Peg Solitaire is a program that is known to be an NP-Complete problem. It is an NP problem because the program can choose a path to follow, and see if the path reaches a solution. Due to non-determinism, it can identify if the board has a solution with one peg in the final state in polynomial time. Since DFS is a “blind” search that attempts to search all of the nodes of the tree,

the time complexity is bounded by the size of the tree. Exploring the full tree would be in exponential time in this case. Therefore, by expressing the number of nodes in the tree using the depth of the tree and the branching factor of the state, which is 4 because there are only four possible moves (left, right, up, down), the big O can be estimated. The maximum depth of the tree would be n-1, so the big O can be estimated to be around $O(4^{n-1})$. The big O simply describes the behavior of algorithms as the number of data (n) scales to infinity. Therefore with this complexity, it can be estimated that the number of nodes would quickly skyrocket as the complexity of the board and the number of pegs increases.

Jupyter notebook testing was used to perform the AI solver on all of the 9 levels, each with the budget of 10k, 100k, 1M, and 1.5M. The program is able to generate the number of expanded nodes, number of generated nodes, length of steps to the best solution, number of pegs left, number of nodes expanded per second, and total execution time, in the form displayed in Figure 2. A node is "expanded" if it is popped from the stack and visited, and is "generated" if a new node is created as one possible state from the currently viewing state. All of the results for each layout and each budget was accumulated to a spreadsheet in order to answer the research question of this experiment: **To what extent do the initial number of pegs and budget affect the solution quality?** The solution quality here will refer to the final number of pegs that the program ends with. The result will be discussed in two parts. One is the initial number of pegs in relation to the resulting number of pegs. Secondly, the effects of the budget will be evaluated in relation to the resulting number of pegs.

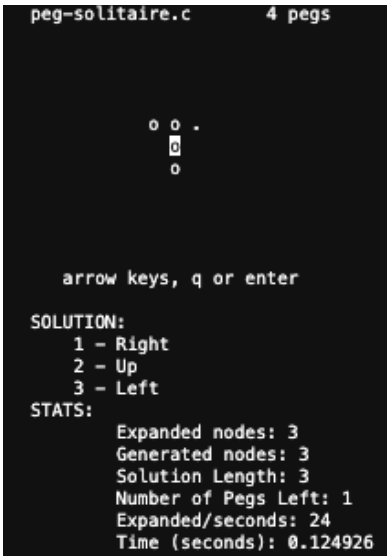

Figure 2

Hypothesis

The initial number of pegs is able to affect the solution quality, simply because with a greater amount of pegs left on the board from the start, the more moves are required to win. Because Peg Solitaire is only solved in exponential time for an AI, it can be expected that a greater number of initial pegs can make the number of expanding nodes rapidly increase, and therefore, will not allow the program to win within the budget. Therefore it can be hypothesized that with a higher number of pegs, the solution quality will worsen, unless the budget can expand enough nodes to find the solution.

It may be wise as well to observe whether the initial number of pegs or the layout level has a bigger impact on the solution quality. This is because although the layouts increase in complexity, level 6 has the highest number of initial pegs. One hypothesis that can be made is that while level 6 does have a higher initial number of pegs than level 7 and 8, it does not differ by much. Therefore, it can be hypothesized that the complexity that comes from the levels would dominate over the initial number of pegs.

The budget can influence the solution quality because the number of nodes the program can expand is bounded by the budget and therefore the resulting number of pegs left is dependent on it as well. Once the number of expanded nodes exceed the budget, it will terminate itself and the AI will execute the best known saved solution. Therefore, the solution quality is influenced by the budget, just like distance a car can travel is affected by how far away a wall is put in front of it. Hence, it can be hypothesized that a low budget may be able to get the program to win with one peg left at a low level layout, but higher level layouts may not be able to finish within the budget with the lower side of the budget. Therefore, the number of pegs left can be expected to increase as the layout level increases, and the effect can be expected to be greater with lower budgets.

Method

1. Go to JupyterLab and open visualize ass3_testing script.
2. Enter all possible levels “0” to “8”.
3. Run the test script with a 10,000 budget and record results on a spreadsheet.
4. Run the test script with a 100,000 budget and record results on a spreadsheet.
5. Run the test script with a 1,000,000 budget and record results on a spreadsheet.
6. Run the test script with a 1,500,000 budget and record results on a spreadsheet.
7. Use specific columns to make various observations.

Results and Discussion

This section will go through the results relevant to answering the research question. Figure 3 shows a raw data of all of the computed results. A discussion on the effects of the initial number of pegs and budget on the resulting number of pegs left will follow.:

Layouts	*Budget*	*Initial Number of Pegs*	*Expanded Nodes*	*Generated Nodes*	*Solution Length*	*Number of Pegs Left*	*Expanded/Second*	*Total Execution Time (Seconds)*
0	10000	3	2	2	2	1	12	0.162527
1	10000	4	3	3	3	1	22	0.131219
2	10000	7	7	8	6	1	55	0.12566
3	10000	17	3541	10282	16	1	25679	0.137893
4	10000	32	1065	2418	31	1	8484	0.125528
5	10000	36	10000	26495	32	4	62250	0.160642
6	10000	44	10000	29368	39	5	61884	0.161592
7	10000	38	10000	32469	34	4	50715	0.197178
8	10000	40	10000	27562	34	6	45775	0.218458
0	100000	3	2	2	2	1	15	0.127402
1	100000	4	3	3	3	1	23	0.128032
2	100000	7	7	8	6	1	56	0.124816
3	100000	17	3541	10282	16	1	25637	0.138116
4	100000	32	1065	2418	31	1	8299	0.128324
5	100000	36	100000	359818	33	3	186139	0.537232
6	100000	44	100000	374378	40	4	184966	0.540637
7	100000	38	100000	386440	36	2	174897	0.571762
8	100000	40	100000	349921	36	4	183730	0.544276
0	1000000	3	2	2	2	1	15	0.125956
1	1000000	4	3	3	3	1	23	0.124803
2	1000000	7	7	8	6	1	56	0.12447
3	1000000	17	3541	10282	16	1	24993	0.141675
4	1000000	32	1065	2418	31	1	7805	0.136437
5	1000000	36	1000000	4488464	34	2	201892	4.953129
6	1000000	44	1000000	4481233	41	3	209294	4.777953
7	1000000	38	1000000	4790308	36	2	200852	4.978772
8	1000000	40	1000000	4073028	36	4	215498	4.640404
0	1500000	3	2	2	2	1	15	0.131969
1	1500000	4	3	3	3	1	24	0.124926
2	1500000	7	7	8	6	1	56	0.123413
3	1500000	17	3541	10282	16	1	25840	0.137034
4	1500000	32	1065	2418	31	1	8257	0.128967
5	1500000	36	1090275	4898609	35	1	204585	5.329202
6	1500000	44	1500000	7020668	41	3	205216	7.309338
7	1500000	38	1500000	7173504	36	2	200037	7.49859
8	1500000	40	1500000	6361454	36	4	2111794	7.082321

Figure 3

Initial Number of Pegs vs. Number of Pegs Left

By accumulating the number of pegs left for each budget as a function of the number of pegs initially on the board the following table can be created (Figure 4):

Initial Number of Pegs	*Layouts*	*Budget = 10k*	*Budget = 100k*	*Budget = 1M*	*Budget = 1.5M*
3	0	1	1	1	1
4	1	1	1	1	1
7	2	1	1	1	1
17	3	1	1	1	1
32	4	1	1	1	1
36	5	4	3	2	1
44	6	5	4	3	3
38	7	4	2	2	2
40	8	6	4	4	4

Figure 4

This can then be plotted as a scatter plot as in Figure 5 to view the trend of the solution quality as the layout level increases and the initial number of pegs changes.

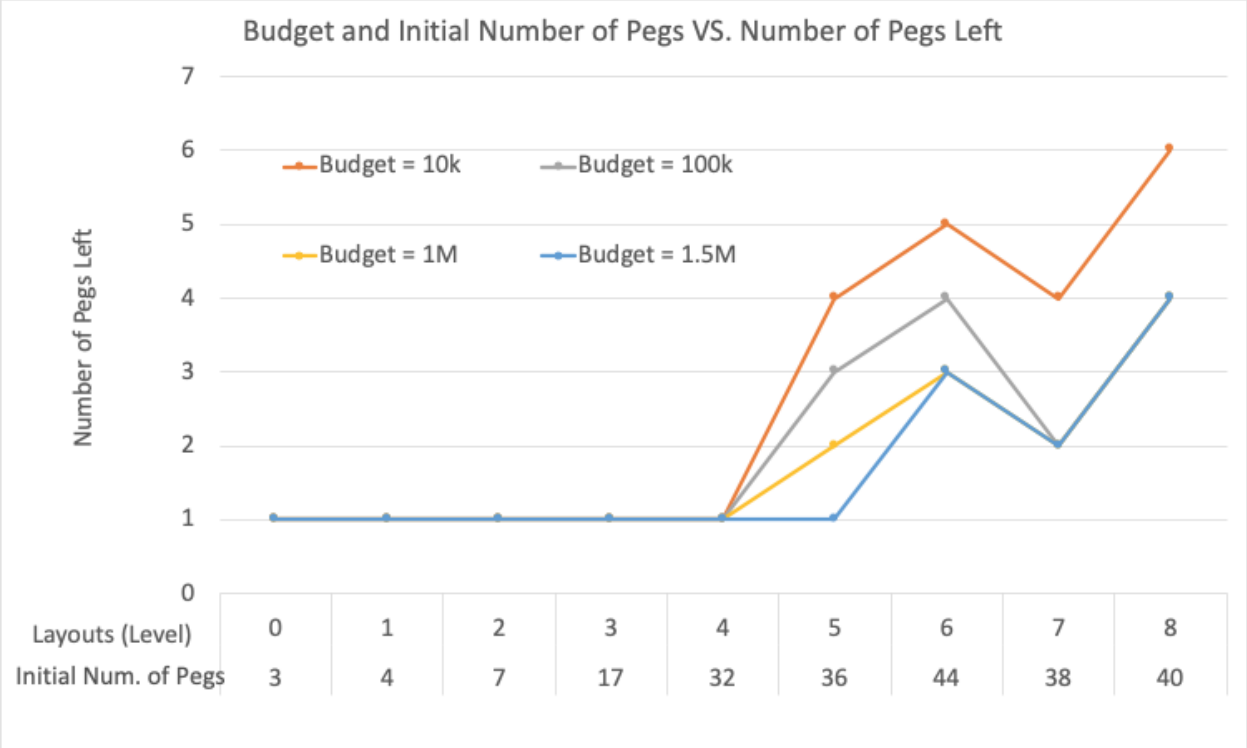

Figure 5

The x-axis represents the layouts and the initial number of pegs. The y-axis represents the number of pegs left when the program finishes running. Each line represents a different budget and the amount is specified in the figure. From a broad perspective, although all of the results finish with one peg left until

layout number 3, the number of pegs left at the end of the program increases from layout level 4. This shows that the computational complexity grows from level 4. Such observation can be attested from viewing figure 3 as well, because in the columns for expanded nodes and generated nodes suddenly increase from layout 4.

The higher level layouts have a generally higher number of initial pegs. And it seems that from level 5, some of the budgets are unable to reach the winning state within the budget. The effects of budgets will be discussed in detail in the next section for "budget vs. the last four layouts", but the general trend of figure 5 shows that the solution quality is worsened as the initial number of pegs increase, and the effect is greater when the budget is lower.

The hypothesis explained that the number of pegs left would increase as the layout level increases. However, this graph shows that it is not definitely so. Though level 7 level is thought to be harder to solve than level 6, the graph shows a greater amount of pegs left for level 6 than level 7. On the other hand however, level 8 with a lower number of pegs than level 6 has an even higher number of pegs left.

Perhaps the effects of pegs left for levels and initial pegs is evident when the average amount of pegs left across budgets are compared for each layout (Figure 6). Considering from layout level 0, although there is a general trend that the solution quality is worse as the levels increase, it can be seen that the initial number of pegs can significantly impact the solution quality because level 6 has an average value much higher than level 5 and level 7. However, the complexity that comes from the levels can have a greater impact if the difference in initial pegs is small, because level 8 has the worst result with 40 pegs to begin with.

Layouts	**Average amount of Pegs Left Across Budget**
5 (36 initial pegs)	2.5
6 (44 initial pegs)	3.75
7 (38 initial pegs)	2.5
8 (40 initial pegs)	4.5

Figure 6

Budget vs. Number of Pegs Left (The Last Four Layouts)

Figure 5 shows that the program is able to win the game within the budget for the first five layouts, but this deviates from the layouts beyond. And it seems like the solution for the number of pegs left differs for each budget at most of the layout levels. Therefore, it is worthy to focus on the last four layouts and how the budget has affected its solution quality. First, the effect on the final number of pegs will be evaluated in accordance to Figure 6, which uses the same table as Figure 4 but focuses on the last four layouts.

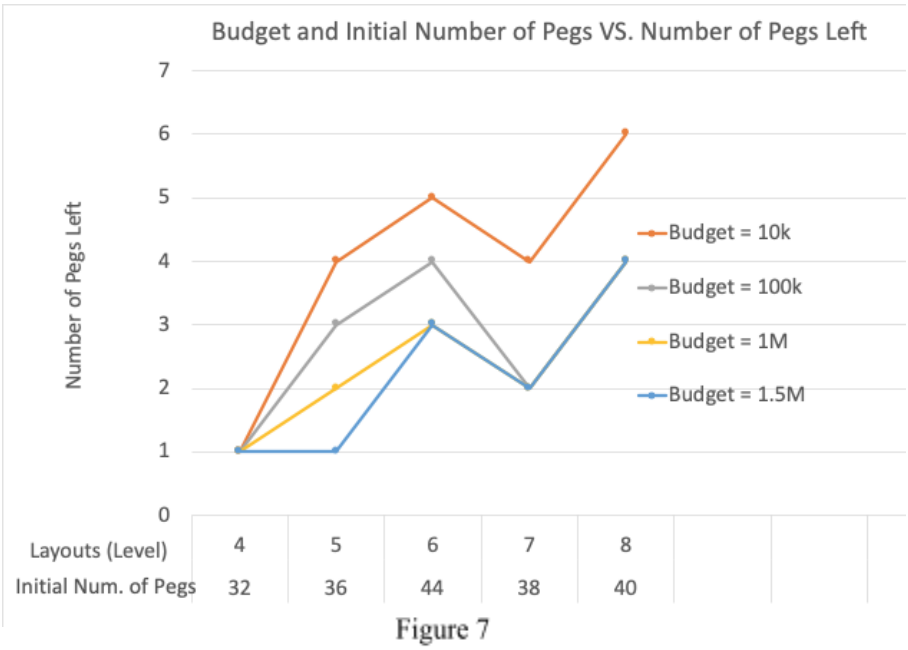

Figure 7

Level 4 was intentionally left in the figure to show that all of the budgets ended with one peg left, and won the game at that level. However, the budget starts to increase afterwards. It is important to note that the higher the budget is when running the program, the lines are lower. This means that with a lower budget the number of pegs left at the end of the program are generally higher than the number of pegs left when starting with a high budget. This can be explained due to the fact that the program's solution is bounded by the budget it has, as explained in the hypothesis. Therefore, the program will require a greater budget to expand on more nodes if the problem is more complex or if the number of initial pegs are high. This can explain why the worst solution is at the top right where the program ran with a budget of 10,000 for layout 8 ended with 6 pegs left. Therefore, although the initial number of pegs and layout level can heavily impact the solution quality, this result shows that the solution is subject to the budget given by the user, with a greater budget allowing for more expanded nodes.

As an addendum to support the theory that the greater budget will allow for a greater number of expanded nodes, the expanded number of nodes can be plotted as a function of the initial number of pegs (Figure 8).

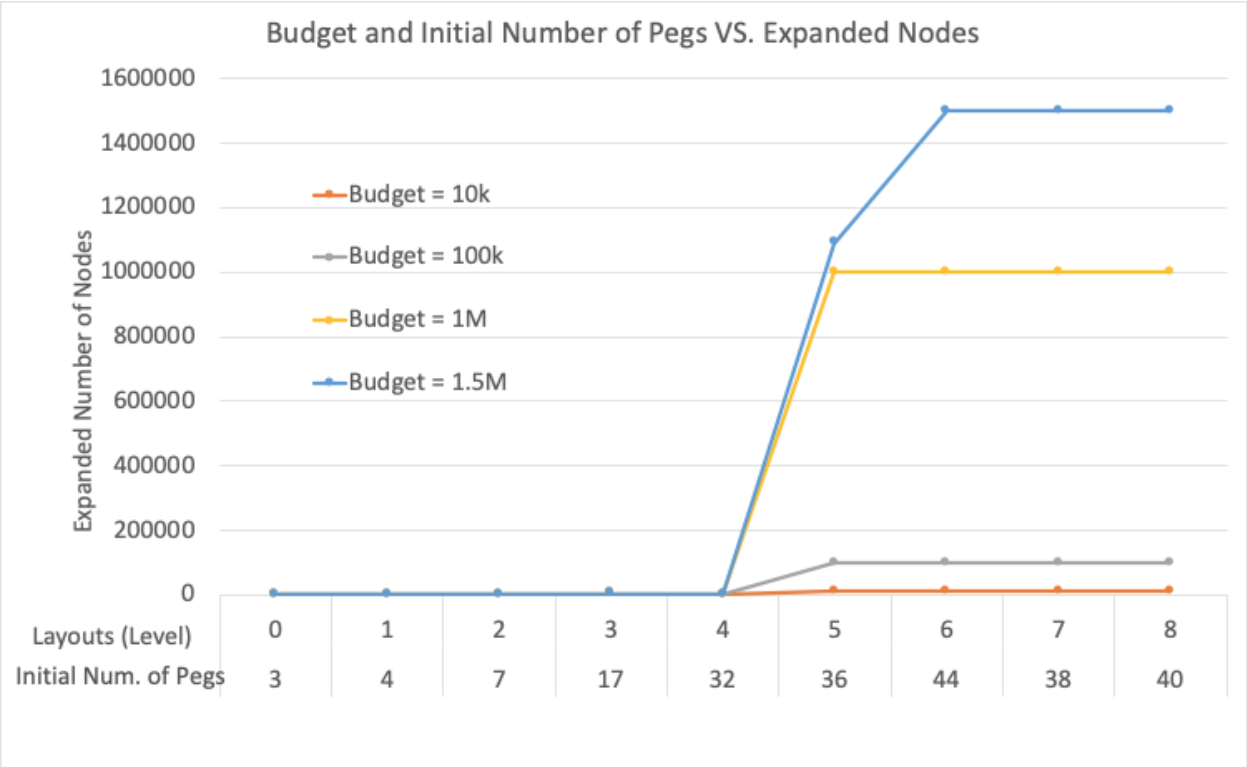

Figure 8

From the above figure, it is evident that the higher budget expands more nodes, and due to the exponential time complexity of the NP-Completeness of Peg Solitaire all of the budgets seem to literally be capped from a certain level, because the amount of expanded nodes skyrocket from level 5. All of the budgets seem to increase in number of expanded nodes from level 5, the level that only a budget of 1.5 million was able to solve (Figure 7). With Figure 8, it can be seen that level 5 was completed at around 1.1 million expanded nodes. The flat line from level 5 for all budgets except for the 1.5 million budget run shows that their budgets couldn't withstand the amount of expanded nodes. The 1.5 million budget run however seems to hit its limit at level 6.

Conclusion

In light of the detailed experimentation and analysis above, the research question of: **To what extent do the initial number of pegs and budget affect the solution quality?** has been understood and answered. It was found firstly that the initial number of pegs can significantly impact the solution quality especially for a lower budget, because the greater the initial number of pegs are, a greater budget is required. Another observation made was that the initial number of pegs do not dominate the solution quality, but does have a significant impact on it. The hypothesis that the solution quality will worsen as the initial number of pegs increase has been attested. However, it was unexpected that the effects of the initial number of pegs is so big that the average number of pegs left at the solution for level 6 was greater by more than one whole peg than level 7, which was deemed to be harder than level 6 (Figure 6).

As the number of pegs increased, the solution required a greater budget. The results showed that although the initial number of pegs and the difficulty of the layout can affect the solution quality, all was subject to the set budget. The greater the budget, the program was able to explore more nodes and the solution quality was better. The lesser the budget and the higher the level, the solution quality worsened. This supported the hypothesis.

Figure 8 was crucial in showing that the greater the budget is, the program is able to expand more nodes and potentially reach a solution. Level 5 was such an example. All budgets were unable to reach the solution with their budgets, except for the 1.5M budget run. As an additional experiment the program was run with a 5M budget, but all of the last three levels were unable to be solved. This not only supports the exponential time complexity of the “NP-Complete”ness of creating an AI for Peg Solitaire, but also shows just how important an efficient data structure, algorithm, and optimization is required to run programs in the real world. Such importance is not easily evident because all algorithms and computation are hidden from the outside interface, but is heavily important for those who write the programs. The fact that the complexity was exponential worsened the effects of a higher number of pegs because more states can be generated, and therefore there are many more nodes to expand to be able to reach a solution.

One limitation from the AI program was that although the program written for this experiment did not use any data structures than the provided hash table and stack, a greater optimization could have been done. This is where the program would be able to identify a symmetry on the board. If there is a symmetry on the board it is likely that the program can produce the same state from multiple different moves. Therefore, identifying this and not considering the moves that turn out to be duplicates when the board is rotated, could have cut the program’s complexity significantly because many of the extra nodes can be skipped over.

Another limitation in the discussion is that the difference in complexity between each level has not been recognized. It was discussed that the initial number of pegs can significantly impact the number of pegs left at the end, but it is difficult to discern the effects that come from the initial number of pegs and the difficulty of the board.

Resources

Fitzpatrick, G 2020, Workshop 10: Graph Search Algorithms, lecture slides, COMP20003, University of Melbourne.