Experimentation of AI Solver Algorithm for Peg Solitaire Game

Algorithms and Data Structures 2020, Assignment 3

[redacted]

Full Name: Aoi Fujii

Introduction

In order to solve the peg solitaire game, an AI solver algorithm was created utilising DFS methods implemented in the assignment 3 specification. The algorithm searches for the next move from a current state of a m x m grid board. The action is executed in the following order: traverse $0 < x < m$, $0 < y < m$, and jump from {left, right, up, down}. The Graph $G = <V, E>$ is implicitly defined with V as states and E as the legal jump. The DFS algorithm search for a graph structure with $O(|V| + |E|)$ time complexity and $O(|V|)$ space complexity when the graph is traversed without repetition. Since DFS explores the nodes as far as possible before it reaches a baseline, the solution is produced with a relatively small budget. Since the AI solver will explore all paths until it finds a solution or it exceeds a given budget, this DFS strategy gives a path leading to a state with the least number of remaining pegs.

The AI solver algorithm was tested for 9 different game layouts with 4 different budgets (10K, 100K, 1M, 1.5M). The number of expanded nodes, generated nodes, solution length, number of pegs left, expanded nodes per seconds, and execution time were obtained from the standard output (Table 1). The generated data was used to compare the solution quality for different limits, different number of initial pegs, the time complexity of DFS with the size of a graph, and the time complexity for this AI solver algorithm with the initial number of pegs. Note that the graph size was used to analyse the time complexity in figure 5 as the actual execution time varies for each test. Here, linearity of the search time to the size of the graph is assumed.

Table 1. Statistics for AI solver for each layout with different budget.

Layout	0	0	0	0
Max budget	10K	100K	1M	1.5M
Expanded nodes	2	2	2	2
Generated nodes	2	2	2	2
Solution Length	2	2	2	2
Number of Pegs Left	1	1	1	1
Expanded/seconds	18	18	12	17
Time (seconds)	0.106577	0.10782	0.155262	0.113357

1	1	1	1	2	2	2	2
10K	100K	1M	1.5M	10K	100K	1M	1.5M
3	3	3	3	7	7	7	7
3	3	3	3	8	8	8	8
3	3	3	3	6	6	6	6
1	1	1	1	1	1	1	1
26	28	25	27	65	68	60	69
0.114853	0.104427	0.119424	0.108022	0.106099	0.102407	0.115236	0.101326

3	3	3	3	4	4	4	4
10K	100K	1M	1.5M	10K	100K	1M	1.5M
3541	3541	3541	3541	1065	1065	1065	1065
10282	10282	10282	10282	2418	2418	2418	2418
16	16	16	16	31	31	31	31
1	1	1	1	1	1	1	1
27453	28061	29155	27271	9349	9384	9117	9631
0.12898	0.126187	0.121453	0.129844	0.113907	0.113479	0.116813	0.110576

5	5	5	5	6	6	6	6
10K	100K	1M	1.5M	10K	100K	1M	1.5M
10000	100000	1000000	1090275	10000	100000	1000000	1500000
26495	359818	4488464	4898609	29368	374378	4481233	7020668
32	33	34	35	39	40	41	41
4	3	2	1	5	4	3	3
65168	102134	140017	154012	62316	155301	171626	133322
0.153449	0.979102	7.14195	7.079128	0.16047	0.64391	5.826623	11.250888

7	7	7	7	8	8	8	8
10K	100K	1M	1.5M	10K	100K	1M	1.5M
10000	100000	1000000	1500000	10000	100000	1000000	1500000
32469	386440	4790308	7173504	27562	349921	4073028	6361454
34	36	36	36	34	36	36	36
4	2	2	2	6	4	4	4
63504	150076	158431	164358	63845	163100	167905	172966
0.157469	0.666328	6.311887	9.126382	0.156628	0.61312	5.955746	8.672175

Methodology

1. Run the AI solver algorithm for each layout (0-8) with different budgets (10K, 100K, 1M, 1.5M).
2. Store the generated data for further analysis (Table 1).
   a. Plot the number of pegs left based on the number of initial pegs for each layout and different budget for comparison (Figure 1, 2).
   b. Calculate the solution quality of the algorithm for layout 5 - 8 with different budgets. Solution quality was calculated as follows: solution quality = (i + 1 – number of pegs left) / i , where i = the number of initial pegs on the board. Plot the solution quality based on different budget in the last 4 layouts (5-8) (Figure 3).
   c. Plot the execution time of this DFS algorithm as a function of the graph size (V+E) (Figure 4).
   d. Plot the graph size (V+E) based on the initial number of pegs on each layout 0 - 5 (Figure 5).

Data and comparisons to the theory

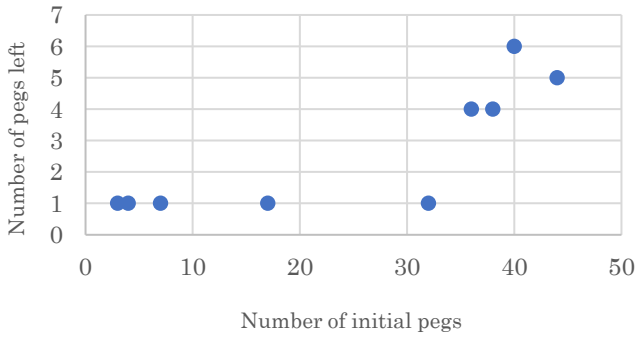
(1) Number of pegs left for different number of initial pegs

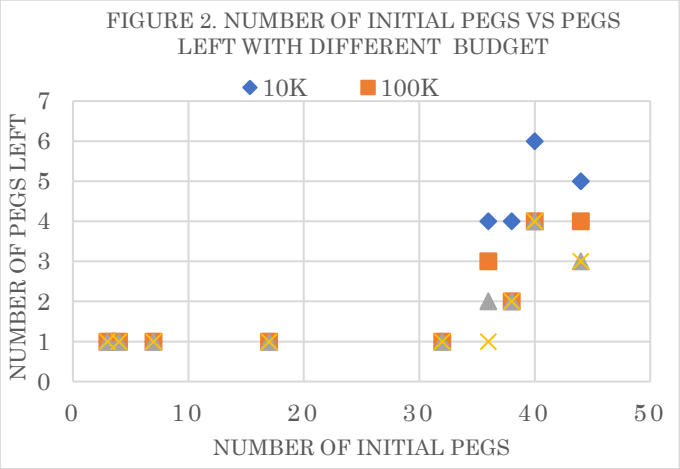
The number of pegs left for the 10K budget remained 1 until 32 initial pegs. After initial pegs reached 32, the number of pegs left increased as the initial pegs increased (Figure 1). Although the number of pegs left tends to be lower for a higher budget, the tendency that the remaining pegs remains 1 and starts to increase after a certain limit is the same for all 4 budgets (Figure 2).

This is expected as the limited number of nodes could be searched within a certain budget. After the algorithm runs out of budget to find a solution, the number of pegs left on the board will increase as the initial pegs increase.

However, the number of remaining pegs varies depending on the layout of the game board. Since an AI solver algorithm is searching a path in a certain order ($0 < x < m$, $0 < y < m$, {left, right, up, down}), some layout could be more suited to find a better solution with a smaller budget.

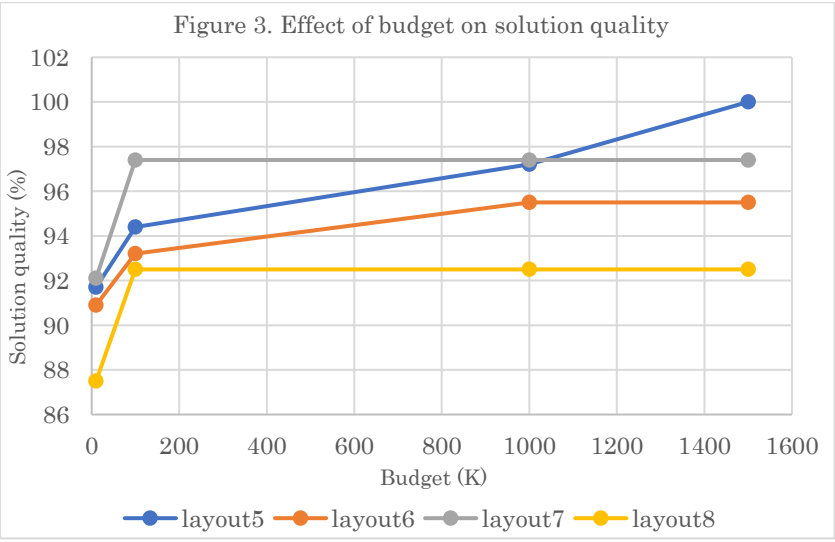
Figure 1. Number of initial pegs vs pegs left with 10K budget





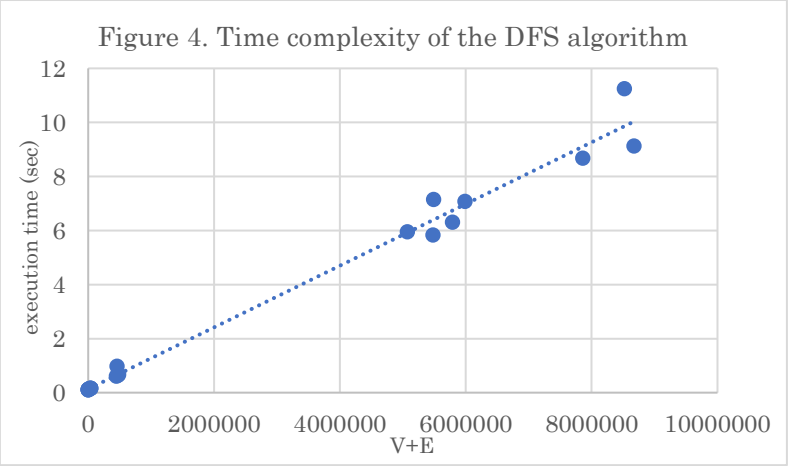
(2) Solution quality for different limits

The solution quality becomes closer to 100% as the budget increases for layout 5 since it requires a smaller budget to find a solution. However, the increase of solution quality is not proportional to the budget increase. Especially for the layout 7 and 8, the solution qualities remained the same which are 97.4% and 92.5% respectively after 100K budget (Figure 3). This is expected as the layouts require a much higher budget to get a better solution. Generally, a better solution is obtained with a higher budget as it allows more nodes to be explored. However, since the DFS algorithm searches for the other path when it fails to find a solution (reached the baseline), exploring a higher number of nodes does not directly increase the solution quality.



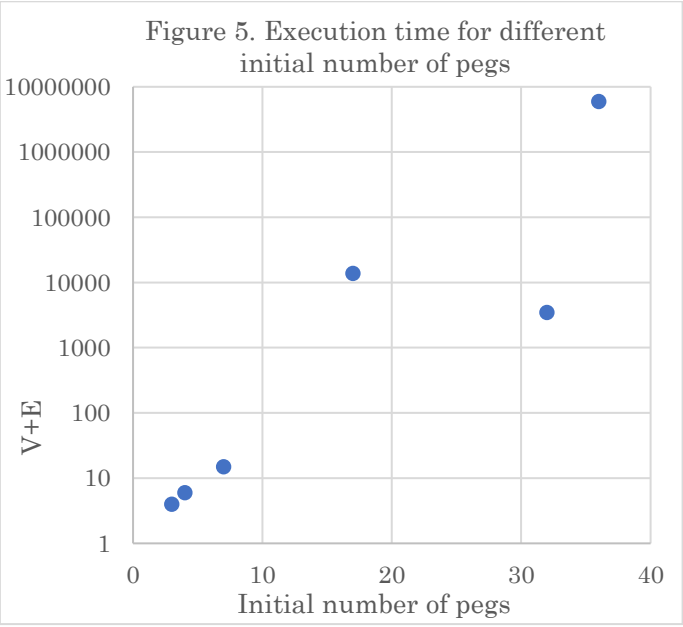
(3) Time complexity of DFS strategy

The execution time of this algorithm increased proportionally to the graph size (V+E). This is expected since the DFS algorithm has a time complexity of O(|V|+|E|) in theory which is linear in the size of the graph. More data points could be obtained to observe this trend more clearly.



(4) Time complexity of the AI solver algorithm for different number of initial pegs

The graph size (V+E) increases exponentially as the number of initial pegs increases (notice the logarithmic scale) (Figure 5). This is because the AI solver problem is an NP-Complete problem which runs in exponential time based on the size of the problem. Hence, the time complexity O(|V|+|E|) to find a solution develops exponentially as the number of initial pegs increases. However, some points may vary which could be caused by the suitability of some layouts to the AI solver algorithm which searches in a certain order (0 < x < m, 0 < y < m, {left, right, up, down}).



Conclusion

It was found that the number of pegs left remains 1 and starts to increase after the algorithm runs out of budget to find a solution. The number of remaining pegs varies depending on the layout of the game board due to its suitability to the specific searching order of the AI solver algorithm. The increase of solution quality was not proportional to the increase in the number of explored nodes, but the solution tends to be better as more nodes are explored. This better solution obtained by searching more nodes compensates with time for searching. The observed execution time of this DFS algorithm followed the theoretical complexity of $O(|V|+|E|)$. The theoretical complexity of NP-Complete problem was also matched with the observed evidence as expected. The search for a solution runs in exponential time as a function of the initial number of pegs.

Recommendations

The experiment could be further conducted with a larger number of initial pegs on the computer with higher RAM. For future studies, it might be interesting to consider the search algorithm with different order (not $0 < x < m$, $0 < y < m$, {left, right, up, down}) or different search methods such as BFS to compare its solution quality and time complexity. It might be also interesting to find how the difficulty level of the game changes between some layouts with the same number of initial nodes. However, theoretical analyses should be put on focus as empirical analysis will not prove the time complexity of searches.

References

Assignment 3 Specification (2020, October). Retrieved from https://canvas.lms.unimelb.edu.au/courses/87167/files/5464648/download?wrap=1