2020 Experiment (Assignment 3)

November 1

Algorithms and Data Structures COMP20003

Authored by: ***Fu Sheng (Jeremy) Huang 1046432***

Introduction

In this assignment, we will be expected to build a solver to play “Peg Solitaire”. When the AI solver is called, it should explore all possible paths (sequence of jump actions) following a “***Depth First Search (DFS)***” strategy, until consuming the budget or until a path solving the game is found. And the most important is the algorithm will ***return the best solution found***.

We are given open-source code and need to fill the missing parts which are the ***applyAction*** and ***find_solution***. I changed a ***stack_pop*** function to make me better understand the function when I implement pop action. And made a new linked list type to handle the memory leak problem.

We run the program by the following line:

```
./pegsol <level> AI <budget> play_solution
```

In my personal use, I only use play_solution once for checking my code. And out output file will contain number of ***expanded nodes number***, ***number of generated nodes***, ***pegs left in the solution***, ***nodes expanded per second***, and ***total search time(second).***

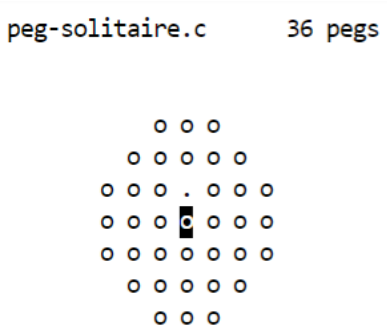

Figure 1

After solving the peg-solitaire, we should have only 1 peg at the end and should have the best result representing the best solution found based on our algorithm.

In the following report I will show the relationships between budget and the layout.

Result Overview

Table 1

0 (3 pegs)	Expanded Nodes	Generated Nodes	Solution Length	Number of Pegs Left	Expanded /second	Time (Seconds)
10K	2	2	2	1	15	0.130262
100K	2	2	2	1	15	0.126351
1M	2	2	2	1	15	0.128146
1.5M	2	2	2	1	15	0.127245

Table 2

1 (4 pegs)	Expanded Nodes	Generated Nodes	Solution Length	Number of Pegs Left	Expanded /second	Time (Seconds)
10K	3	3	3	1	23	0.136180
100K	3	3	3	1	23	0.128703
1M	3	3	3	1	23	0.127791
1.5M	3	3	3	1	22	0.129549

Table 3

2 (7 pegs)	Expanded Nodes	Generated Nodes	Solution Length	Number of Pegs Left	Expanded /second	Time (Seconds)
10K	7	8	6	1	54	0.129087
100K	7	8	6	1	54	0.132493
1M	7	8	6	1	54	0.127668
1.5M	7	8	6	1	54	0.128260

Table 4

3 (17 pegs)	Expanded Nodes	Generated Nodes	Solution Length	Number of Pegs Left	Expanded /second	Time (Seconds)
10K	3541	10282	16	1	25289	0.140019
100K	3541	10282	16	1	24555	0.144204
1M	3541	10282	16	1	24205	0.146288
1.5M	3541	10282	16	1	25057	0.142312

Table 5

4 (32 pegs)	Expanded Nodes	Generated Nodes	Solution Length	Number of Pegs Left	Expanded /second	Time (Seconds)
10K	1065	2418	31	1	8259	0.128936
100K	1065	2418	31	1	7997	0.133163
1M	1065	2418	31	1	8112	0.131283
1.5M	1065	2418	31	1	8219	0.129573

Table 6

5 (36 pegs)	Expanded Nodes	Generated Nodes	Solution Length	Number of Pegs Left	Expanded /second	Time (Seconds)
10K	10000	26495	32	4	60664	0.165207
100K	100000	359818	33	3	176912	0.577747
1M	1000000	4488464	34	2	196758	5.092381
1.5M	1090275	4898609	35	1	197003	5.534284

Table 7

6 (44 pegs)	Expanded Nodes	Generated Nodes	Solution Length	Number of Pegs Left	Expanded /second	Time (Seconds)
10K	10000	29368	39	5	60367	0.165651
100K	100000	374378	40	4	175824	0.568748
1M	1000000	4481233	41	3	204606	4.887437
1.5M	1500000	7020668	41	3	199263	7.527704

Table 8

7 (38 pegs)	Expanded Nodes	Generated Nodes	Solution Length	Number of Pegs Left	Expanded /second	Time (Seconds)
10K	10000	32469	34	4	60366	0.165655
100K	100000	386440	36	2	181156	0.552010
1M	1000000	4790308	36	2	192330	5.199385
1.5M	1500000	7173504	36	2	193996	7.732082

Table 9

8 (40 pegs)	Expanded Nodes	Generated Nodes	Solution Length	Number of Pegs Left	Expanded /second	Time (Seconds)
10K	10000	27562	34	6	59592	0.167806
100K	100000	349921	36	4	181084	0.552228
1M	1000000	4073028	36	4	208101	4.805350
1.5M	1500000	6361454	36	4	203001	7.389111

Last 4 Layouts Performance

We can know that the budget has positive relation to time in the last 4 layouts.

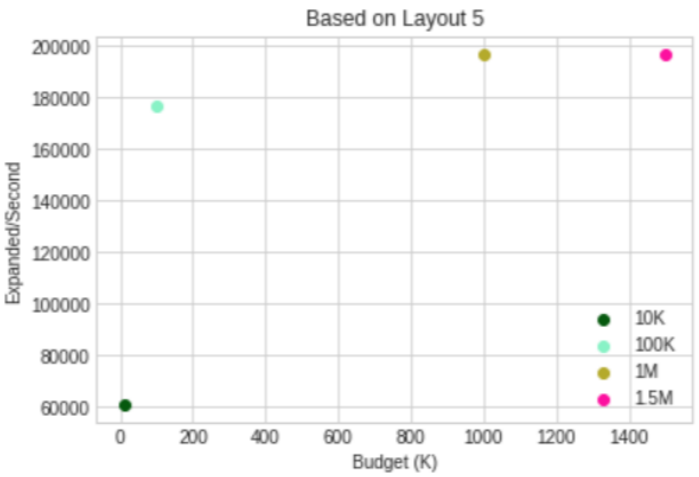

Figure 2

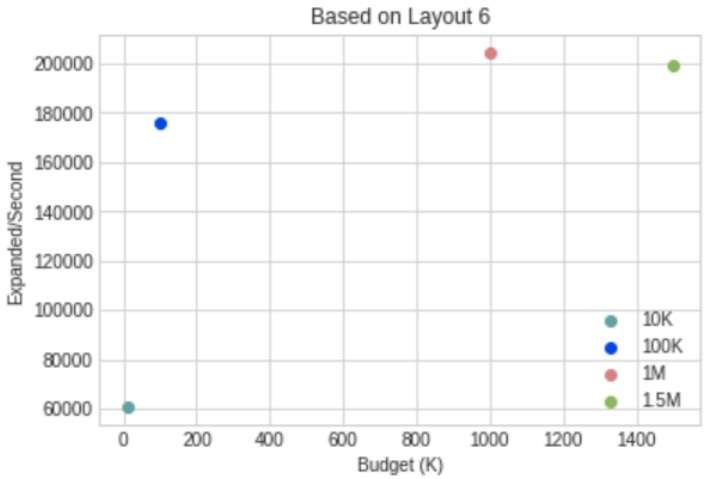

Figure 3

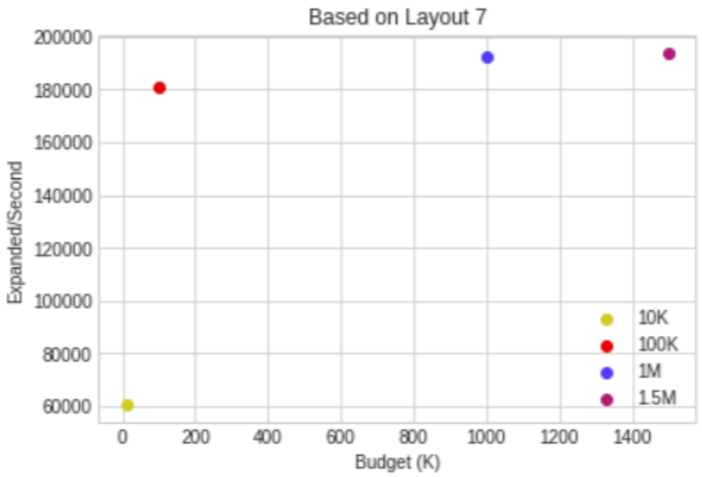

Figure 4

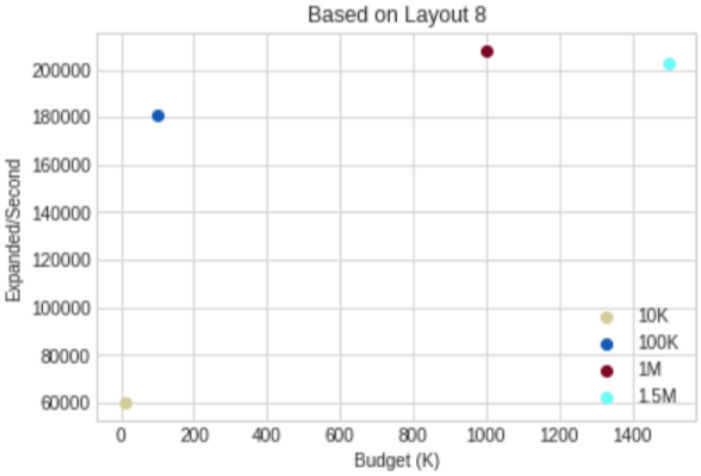

Figure 5

And the most obvious is their **distribution in the graphs are almost the same**, even layout 5 has 1 peg left when the budget is set to 1.5M, but it almost reaches its limit at the value.

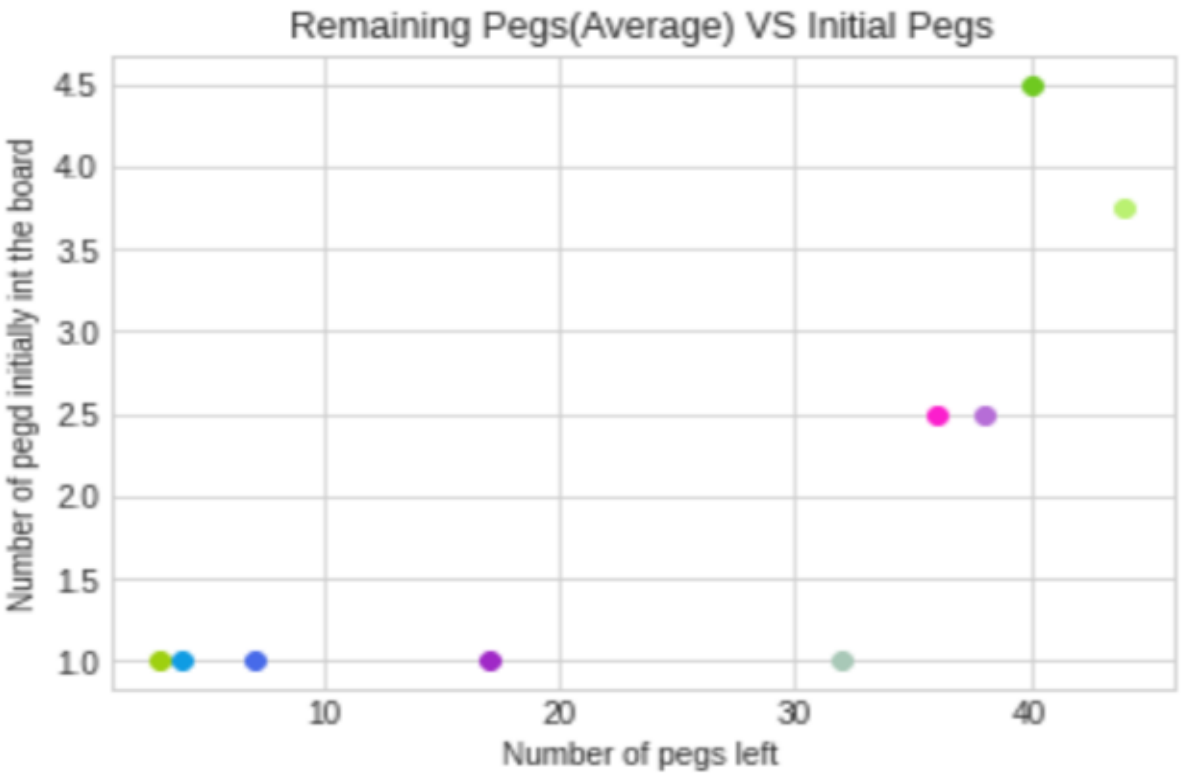

Figure 6

But when we reference to figure 6, which shows our algorithm has the limitation with limit budget set to the implementation. Having said that, this program still performs really great under about 32 pegs initially in the board.

Combine the results from above, we can see our algorithm probably faced the barrier that 36 pegs for the starting execution where we even give it 1.5M budget, it still cannot perform as perfect as the pegs numbers under 32. And its expanded nodes have showed it already got the best performance under these conditions.

Besides the performance in last 4 layouts, we can see our algorithm is really stable when the initial pegs numbers are below 32.

Looking into our Algorithm, it competed in $O(n^3)$ time complexity because we have to loop the board and the actions. Then I use the linked list to have better performance in freeing the unneeded expanded nodes. Then we can see why the number of pegs initially in board has limitation.

Conclusion

Overall, we have pretty good algorithm to be the AI solver in our game. But with high time complexity, it will affect our performance. Also, the budget has played an important role. The larger budget and the number of pegs it has, the harder chance we can find the problem and the more time we need to spend. Vice versa. But with limited budget, we probably cannot find the solution because the AI may not be able to try as many times as it should have.

Improvement

Accordingly, every algorithm can never be proved to be of any given complexity via empirical analysis and instead must be analysed theoretically. Hence, our algorithm that completed in $O(n^3)$ complexity, in most cases like in real world data solving or other cutting-edge online game, it may have even worse performance under the execution algorithm.

We are using the Depth-First-Search concept. In my point of view, I may want to try other advanced algorithm such as A-Star Algorithm or other graph-theory based algorithm in our solver or to minimize any possible unnecessary expanded nodes or generated nodes. Then keep my linked list to work for the memory management, it probably be a better version of the solver.