Peg solitaire Algorithm

Mahee Hossain 1080102

Introduction

Peg solitaire (pegsol) is a classic board game invented in Madagascar in 1687. The game involves a board with pegs that can move up, left, down and right, and the goal is to clear the board by jumping over the pegs. Solving pegsol is an NP-Complete solution, and it involves creating an AI which follows all possible paths with Depth First Search (DFS) to get the best solution.

The base code for pegsol was created by Maurits van der Schee, and was not altered, apart from some additional comments (for clarity) and refactoring (including creating useful constants). Furthermore, I added in a handful of extra functions in the base code to help with implementing the AI (including setting up a linked list which was edited from the linked list in Anh's solution for Assignment 1).

The AI appears to work fine, and all the structures involved in the implementation are freed. As an increased number of pegs means there more states possible, the cost of implementation increases exponentially with the pegs, and the fact that it requires a hash table, stack and linked list means that the solver requires a significant amount of memory to run. Therefore, not all the levels could be solved on JupyterLab, and they would require an increasing budget.

Testing

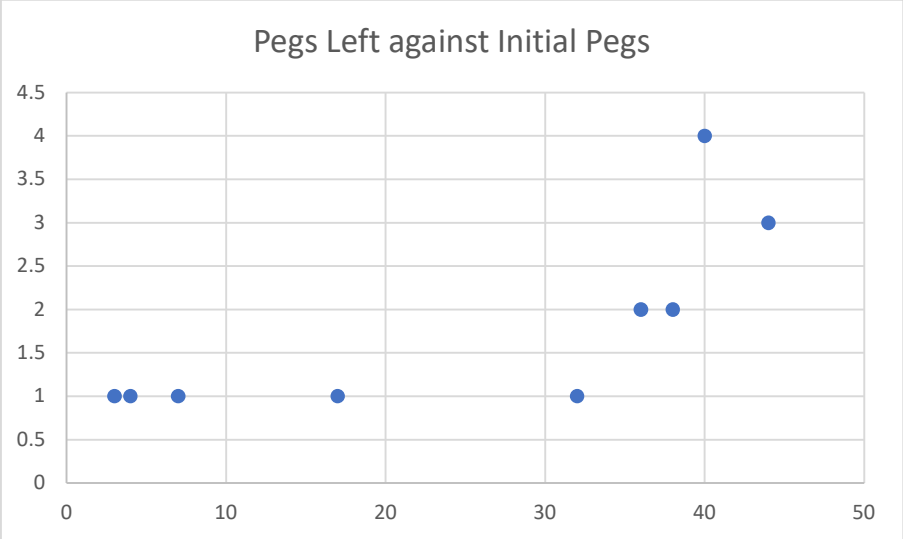
To test how the AI works, we gather statistics for the number of pegs left, amount of nodes generated, amount of nodes expanded (visited in the stack), how many nodes were expanded per second and the total execution time. We had to check these for each level with different budgets (10k, 100k, 1M and 1.5M), and we averaged the best of three takes.

The experimentation was done using the optimisation flag gcc -O3, but the case for Level 7 budget 10,000 had to be run using valgrind. That is as the program gives a "double free or corruption (fasttop)" error when running normally and dumps the core, but with valgrind included the program runs. This error seems to be the result of freeing a node that is already been freed, might be an issue with the implementation of the linked list. With valgrind, the program runs quite slower, through checking Level 6 budget 10,000 (arbitrarily chosen), valgrind appears to be roughly 15 times slower to run. Furthermore, Level 6 budget 1,500,000 would not run for my JupyterLab, without valgrind it had the fasttop error, while with valgrind the run failed (likely was too big).

Results

Level	Budget	Total Expanded Nodes	Total Generated Nodes	Total Solution Length	Total Number of Pegs Left	Total Expanded/Second	Total Time spent
0	10,000	2	2	2	1	54	0.037
	100,000	2	2	2	1	54	0.037
	1,000,000	2	2	2	1	51	0.039
	1,500,000	2	2	2	1	54	0.037
1	10,000	3	3	3	1	79	0.038
	100,000	3	3	3	1	77	0.039
	1,000,000	3	3	3	1	83	0.036
	1,500,000	3	3	3	1	82	0.037
2	10,000	7	8	6	1	185	0.038
	100,000	7	8	6	1	184	0.038
	1,000,000	7	8	6	1	183	0.038
	1,500,000	7	8	6	1	178	0.040
3	10,000	3,541	10,282	16	1	56,920	0.063
	100,000	3,541	10,282	16	1	59,061	0.060
	1,000,000	3,541	10,282	16	1	60,642	0.058
	1,500,000	3,541	10,282	16	1	59,238	0.060
4	10,000	1,065	2,418	31	1	24,440	0.044
	100,000	1,065	2,418	31	1	23,940	0.044
	1,000,000	1,065	2,418	31	1	23,170	0.046
	1,500,000	1,065	2,418	31	1	24,104	0.044
5	10,000	10,000	26,495	32	4	99,478	0.101
	100,000	100,000	359,818	33	3	205,832	0.486
	1,000,000	1,000,000	4,488,464	34	2	249,611	4.008
	1,500,000	1,090,275	4,898,609	35	1	250,645	4.356
6	10,000	10,000	29,368	39	5	97,438	0.103
	100,000	100,000	374,378	40	4	205,659	0.486
	1,000,000	1,000,000	4,481,233	41	3	256,040	3.906
	1,500,000	#VALUE!	#VALUE!	#VALUE!	#VALUE!	#VALUE!	#VALUE!
7	10,000	10,000	32,469	34	4	5,025	1.990
	100,000	100,000	386,440	36	2	183,748	0.547
	1,000,000	1,000,000	4,790,308	36	2	236,132	4.239
	1,500,000	1,500,000	7,173,504	36	2	238,880	6.287
8	10,000	10,000	27,562	34	6	97,222	0.103
	100,000	100,000	349,921	36	4	203,753	0.491
	1,000,000	1,000,000	4,073,028	36	4	261,710	3.826
	1,500,000	1,500,000	6,361,454	36	4	260,648	5.755

Table 1: Total data (averaged from three runs) for the experiment



Graph 1: Pegs left plotted as a function of initial pegs

As the data shows, the higher levels (6, 7 and 8) cannot be solved with the maximum budget of 1.5M. This is as this is an exponentially increasing NP Problem, thus the amount of budget required to solve these would be quite high. Expanded nodes was almost always quite lower than the amount of generated. It appears the maximum number of nodes expanded per second was around 260k, as it tapered off there for the higher budgets. There was some variance in time taken for the same run, thus the average of three was taken to account for outliers. If the P=NP problem is solved, then this experiment can be redone with a better algorithm to try and solve the harder problems in a reasonable amount of time and budget.