C+A5

arcex012, mayxx394

October 29, 2018

1 Experiments and Report

1. The metrics used in our experiment were the number of computation steps, the memory usage (max frontier count), and the time it took for each algorithm to run, timed with a python library called 'timeit'.

For graph search used in Towers of Hanoi, we implemented 3 different ways of maintaining duplicate states that shouldn't be revisited. The first is a simple visited list, the second is an advanced visited list that takes node states back out if a node with lower depth is passed into the valid children function, and the third is a parent trace, checking parent nodes' states. Each of these implementations can be seen in the code pushed to GitHub. For our experiments we decided to use the simple visited list for all our algorithms, as the other two approaches were more time consuming.

2. We chose to experiment with alpha, number of iterations, and starting value. Alpha was tested at 0.98 and 0.3 Number of iterations was tested at 500 and 2000 Starting value was tested at 600,000 and 10,000. End value was always 0.25

3. The data collected was submitted on GitHub as text files while the tables and graphs are within an excel spreadsheet also uploaded to GitHub.

4. Starting with Towers of Hanoi, the maximum problem size, n, that we could solve was:

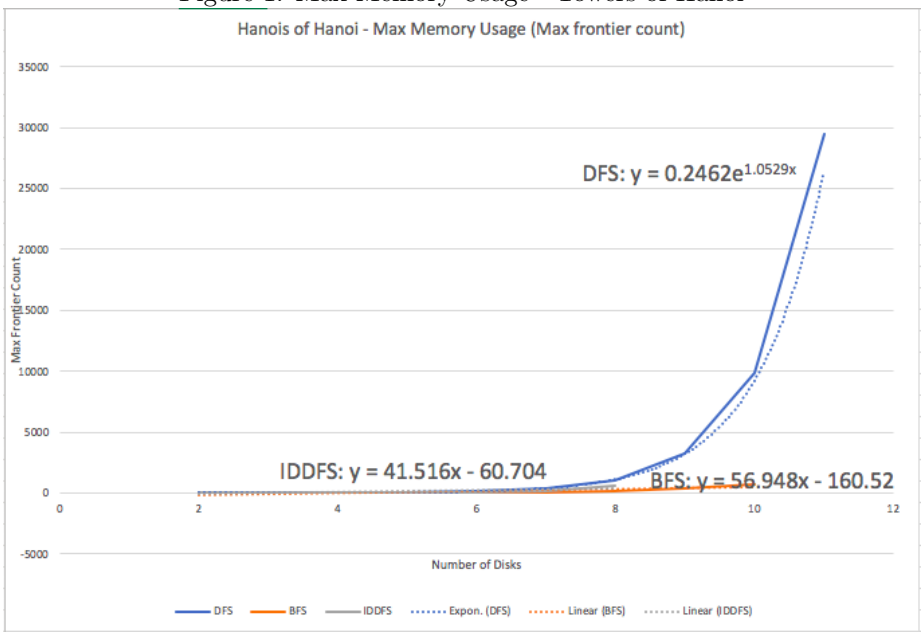
- Dfs Graph: n = 11. When there were 12 towers present when we hit the 1 hour time limit. So only data for 11 towers was collected.
- Bfs Graph: n = 10. When there were 11 towers present when we hit the 1 hour time limit. So only data for 10 towers was collected.
- IDDfs Graph: n = 8. When there were 9 towers present when we hit the 1 hour time limit. So only data for 8 towers was collected.

While for nQueens, the maximum problem size, n, that we could solve was:

- Bfs Tree: n = 11. When there were 12 queens present on the 12 x 12 board, when we hit the 1 hour time limit. So only data for 11 queens was collected.
- IDDfs Tree: n = 11. When there were 12 queens present on the 12 x 12 board, when we hit the 1 hour time limit. So only data for 11 queens was collected.
- Simulated Annealing: n = 15 By setting alpha to 0.98, number of iterations to 2,000 start to 600,000 and end to 0.25 we were able to solve the 15 x 15 board 5 times in a row.

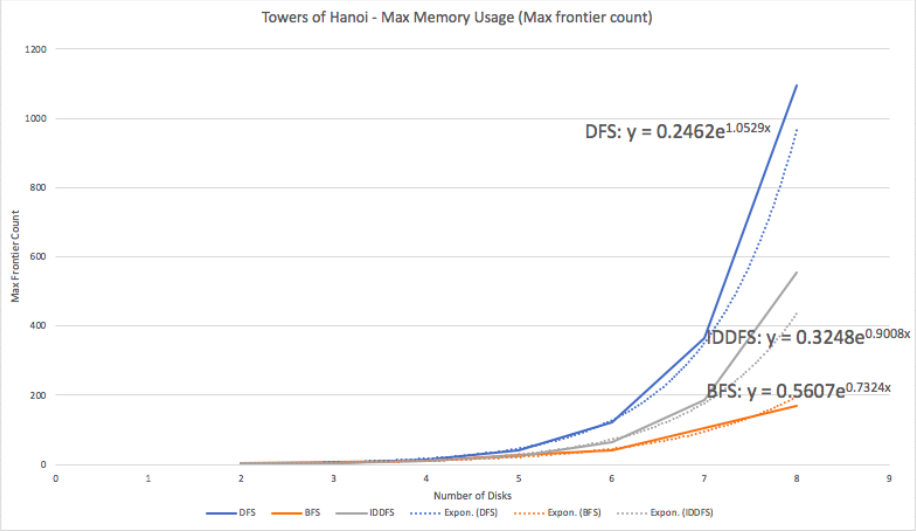
2 Graphs

Figure 1: Max Memory Usage - Towers of Hanoi



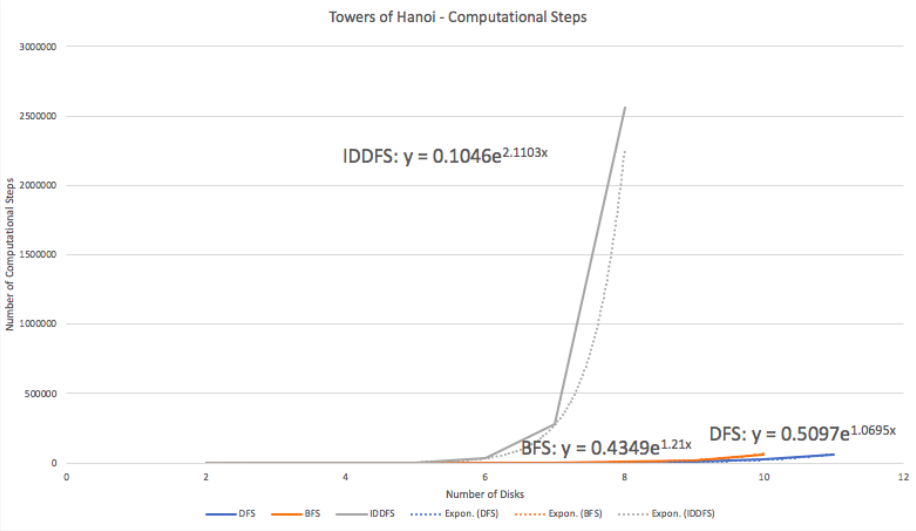
The above graph, Figure 1, shows that BFS has the lowest exponential factor which results in it using the least amount of memory as the number of discs increases.

Figure 2: Max Memory Usage - Towers of Hanoi



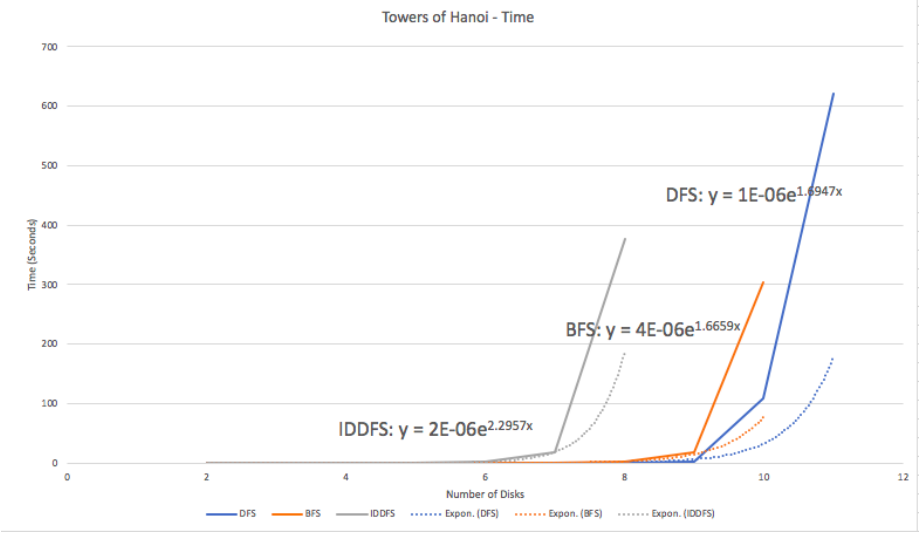
The above graph, Figure 2, is Figure 1 zoomed in. It shows that BFS has the lowest exponential factor which results in it using the least amount of memory as the number of discs increases.

Figure 3: Number of Steps Used - Towers of Hanoi



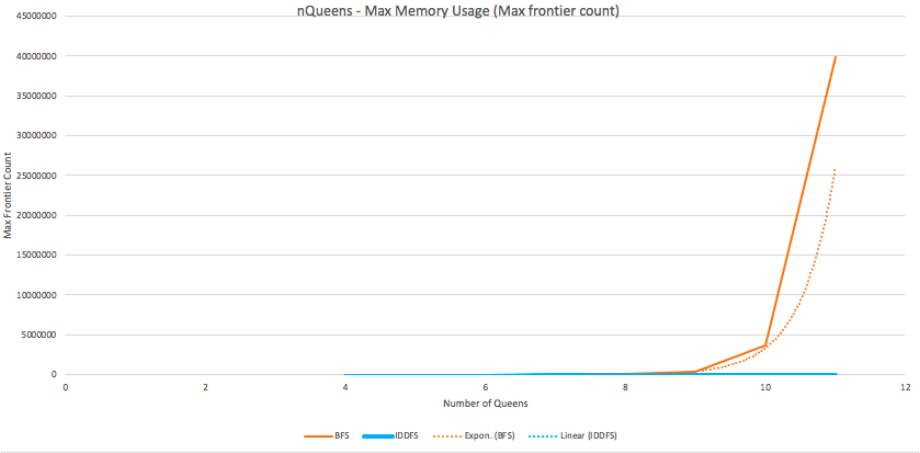
In Figure 3, it is evident that IDDFS is the least efficient due to the huge spike in steps used, while BFS is the most efficient due to it having the smallest y value as x grows larger.

Figure 4: Time Taken - Towers of Hanoi



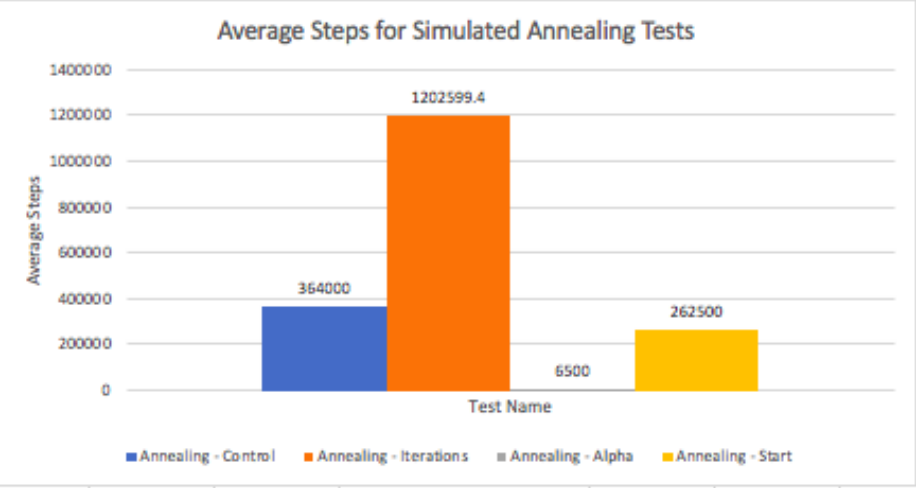
As seen in Figure 4, DFS performs faster than both the other algorithms, solving higher discs problems in less time.

Figure 5: Max Memory Usage - NQueens



In the above Figure 5, Bfs uses much more memory than IDDFS.

Figure 6: Average Steps for Simulated Annealing - NQueens 20 Control = alpha:0.98 start:600,000 iterations:500



As depicted in Figure 6, annealing was the quickest with our low alpha value, and the slowest with our higher iterations value.

Figure 7: Average Value for Simulated Annealing - NQueens 20 Control = alpha:0.98 start:600,000 iterations:500

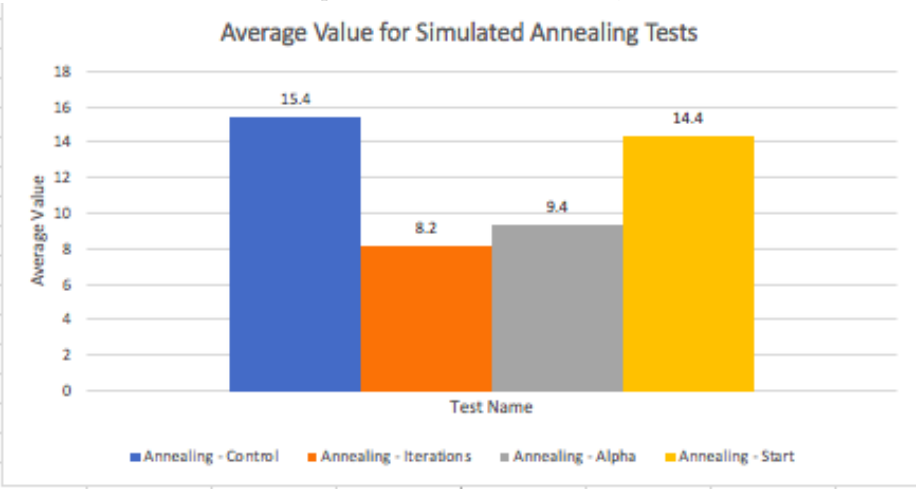


Figure 7 shows that changing our iterations to 2,000 gave us the closest average state to the solution.

Figure 8: Highest Solvable Problem - NQueens 20
Control = alpha:0.98 start:600,000 iterations:500

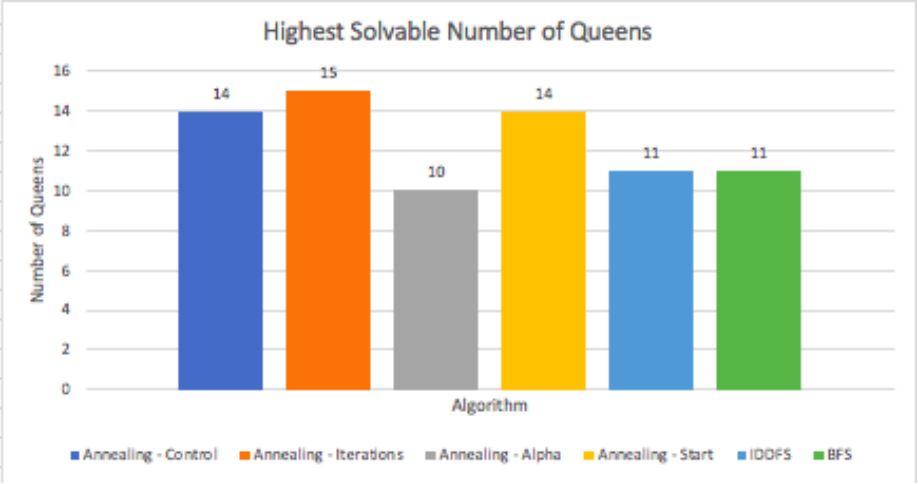


Figure 8 depicts that simulated annealing with a high iteration value let us solve larger NQueens problems than IDDFS and BFS.

3 Tables

Figure 9: NQueens Data for BFS and IDDFS

Nqueens								
BFS								
Number of Queens	4	5	6	7	8	9	10	11
Max frontier count	24	120	720	5040	40320	362880	3628800	39916800
Computational steps	28	97	704	3807	31804	268336	2665488	29133633
Time(Seconds)	0.0004702	0.0016952	0.0106199	0.0483638	0.4539334	4.5566584	48.520302	558.77387
IDDFS								
Number of Queens	4	5	6	7	8	9	10	11
Max frontier count	7	11	16	22	29	37	46	56
Computational steps	42	139	1042	5363	45206	376089	3730159	40710859
Time(Seconds)	0.0008463	0.0023877	0.0148146	0.0693671	0.6293518	5.7291227	61.585471	727.30018

Figure 9 shows our IDDFS Tree and BFS Tree data for NQueens. Computational steps and time go hand in hand, but both are displayed for reference. IDDFS has a much lower memory usage (max frontier count) than BFS, but is slightly slower.

Figure 10: Towers of Hanoi Data

Data for Towers of Hanoi										
DFS										
Number of Disks	2	3	4	5	6	7	8	9	10	11
Max frontier count	3	5	15	41	123	365	1095	3281	9843	29525
Computational steps	6	9	42	81	366	729	3282	6561	29526	59049
Time (Seconds)	0.00019127	0.00026126	0.00119542	0.00410079	0.02033662	0.04810806	0.60537429	2.54150117	108.161611	620.664237
BFS										
Number of Disks	2	3	4	5	6	7	8	9	10	
Max frontier count	2	6	10	26	42	106	170	426	682	
Computational steps	4	18	60	206	644	2038	6220	19086	57684	
Time(Seconds)	0.00037487	0.00098135	0.0033014	0.00777787	0.03517545	0.2790886	1.72380704	17.8409508	304.546665	
IDDFS										
Number of Disks	2	3	4	5	6	7	8			
Max frontier count	3	4	10	23	66	186	554			
Computational steps	9	50	499	3191	32172	278599	2557254			
Time(Seconds)	0.00036645	0.001926	0.01520737	0.09088914	1.24680334	18.5267811	376.880623			

Figure 10 shows the data for all our graph search algorithms. You can see that DFS runs the fastest, but also uses the most memory. BFS finds an optimal solution, but takes a significant while longer to run than DFS. IDDFS also finds the optimal solution, but runs slightly slower than BFS due to our visited list implementation.

Figure 11: Data From Simulated Annealing Testing for problem size 20

For simulated annealing, the control's values are: alpha = 0.98, start = 600,000 , end = 0.25,						iterations = 500
Simulated Annealing						
Control	1	2	3	4	5	Average
Value	12	16	17	16	16	15.4
Steps	364000	364000	364000	364000	364000	364000
Time (Seconds)	18.239922	18.377146	18.353412	18.288589	18.292816	18.310377
Iterations -> 2,000	1	2	3	4	5	Average
Value	12	0	10	7	12	8.2
Steps	1456000	188997	1456000	1456000	1456000	1202599.4
Time (Seconds)	72.868204	9.7798112	72.89423	73.430147	72.960566	60.386592
Alpha -> 0.30	1	2	3	4	5	Average
Value	9	13	9	10	6	9.4
Steps	6500	6500	6500	6500	6500	6500
Time (Seconds)	0.3313537	0.326943	0.3259364	0.3258942	0.3254283	0.3271111
Start -> 10,000	1	2	3	4	5	Average
Value	17	11	11	12	21	14.4
Steps	262500	262500	262500	262500	262500	262500
Time (Seconds)	13.164903	13.141871	13.150054	13.26	13.147836	13.172933
Highest Solve	Highest Solvable Size		Steps	Time		
Control	14		80004	2.3049034		
Iterations	15		431947	13.77123		
Alpha	10		1515	0.0279004		
Start	14		167703	4.7940433		

In the above Figure 11, three different changes are compared to the initial control for simulated annealing, each ran on NQueens with size 20. Each has a different proficiency in value number of queens attacking each other(value), the number of steps taken, and the time it took to complete. Changing iterations to 2,000 gave us the lowest Value, with zero being the optimal solution. Changing alpha drastically reduced run time, and changing start reduced run time a little bit while giving us similar results to our control.

Figure 12: Comparison Results for Simulated Annealing

Highest Solve	Highest Solvable Size	Steps	Time
Control	14	80004	2.3049034
Iterations	15	431947	13.77123
Alpha	10	1515	0.0279004
Start	14	167703	4.7940433

Figure 12 shows the highest size problem we can reliably solve 5 times in a row with each of our algorithms, and the time it takes to do so.