Zeus Polanco Salgado
CS 203 - Assignment 1: Technical Report
May 2021

Technical Report

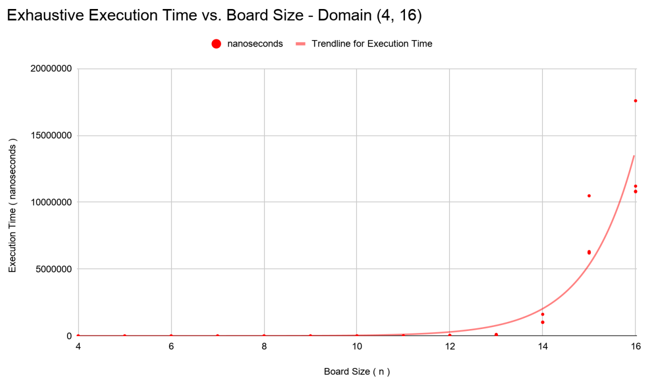
Summary

The purpose of this experiment is to perform an empirical analysis on two algorithms that solve the same problem and compare their results. The two approaches taken to solve the NQueens problem are Exhaustive Search and Iterative Repair. An arrangement of queens is considered a solution when there are no attacks between queens. The algorithms are implemented in Java and executed on a 1.7 GHz Quad-Core Intel Core i7 processor. The data collected from the experiment allows us to observe how the input size (the dimension of the board) affects the execution time (nano seconds) for each algorithm.

Results

Exhaustive Search

The exhaustive/brute force implementation generates permutations using Johnson-Trotter algorithm until a correct solution is found. Theoretically, Johnson-Trotter and the solution check performs the same amount of basic operations for each input size; therefore, there would be no need to calculate best case versus worst case scenarios.

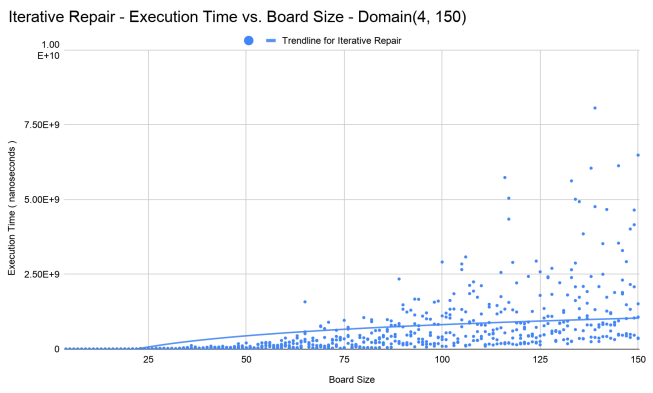


The data shown in Figure 1 shows 5 Exhaustive Search tests for input size of range 4-16. Data shows that the algorithm does not obey linear time complexity. The trendline in the scatter graph is generated using exponential regression and fits appropriately to the data points. The visualization suggests that the algorithm may follow an exponential time complexity but such claims can not be certain without performing a theoretical analysis of this algorithm.

Iterative Repair

Sosic, R., & Gu, J. (1990) A Polynomial Time Algorithm for The N-Queens Problem

The iterative repair approach is an implementation of Sosic, R. & Gu, J 's proposed algorithm. The algorithm generates a pseudorandom permutation and performs a series of swaps until a solution is found; if no solution is found then a new pseudorandom permutation is generated and the process is repeated. The *randomness* of the algorithm adds another factor to the number of operations required to solve the problem



which is the initial setup of the queens before performing any swaps. This means that it is possible for the algorithm to generate a correct solution on the first permutation and not have to perform any swaps.

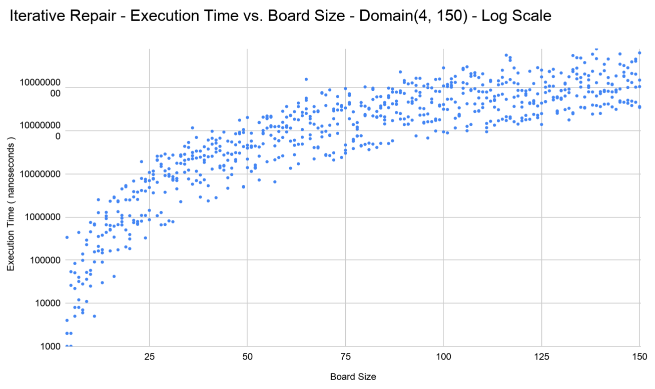
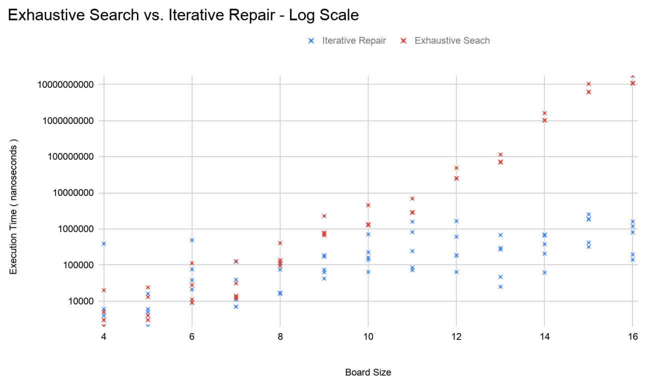


Figure 2 & 3 shows 5 Iterative Repair tests for input size range 4-150. Figure 2 allows us to observe that the algorithm's lower asymptotic efficiency class may not match the upper asymptotic efficiency class. The trendline shown in the graph is generated using logarithmic regression; this trendline could be a potential representation of the average time efficiency. Figure 3 was included because it can allow us to further understand the algorithms' behavior. The data points were graphed on a logarithmic scale which show a much more consistent trend from Figure 2. It shows that the Execution Time approaches an asymptote as the input size increases as well as a possible range of execution time for each input size.

Comparison

Figure 4 shows both algorithms' test results side by side (input range 4-16). Data shows that for the first couple of test cases, until approximately sizes 8-10, the performance of the algorithms are almost the same. After sizes 8-10, Exhaustive Search execution time shows to increase dramatically compared to Iterative Repair.



Conclusion

By observing the test results from both algorithms, we can conclude that Iterative Repair out performs Exhaustive Search as the input size increases. For extremely small cases, the comparison is almost negligible, it might even be possible for Exhaustive Search to perform a faster execution time than Iterative Search for small input sizes. Iterative Search might also provide a much larger range and less predictable performance compared to Exhaustive Search. Regardless, Iterative Repair guarantees quicker performance on average for larger input sizes.