Hope Fetrow  
CS4310 Design and Analysis of Algorithms   
Semester/Session: Summer I 2021

**Result Analysis**

The slowest and most impractical algorithm for solving the n-Queens problem would include checking every position on the chessboard times for queens. This would give n-Queens a worst-case time complexity of ).

Backtracking algorithm is used used to reduce upper bound on n-Queens problem. This is a recursive method which starts a queen at an edge and, places the next queen in a position that does not conflict with the first queen placed (i.e. first queen is not in same col or diagonal).

Then the next queen is placed in a safe position according to the first two queens placed, and so on. If it is found that n number of queens cannot be placed, it means there were not safe positions for all the queens, therefore it backtracks to a parent with a child that has not been visited and continues trying to find safe positions for all queens.

**[Analysis of N-Queens backtracking algorithm:](https://www.quora.com/What-is-worst-case-complexity-of-binary-search" \t "_top)**

[An upper bound on number of nodes checked in pruned state space tree before finding all solutions can be obtained by counting number of nodes in entire state space tree. The number of nodes at level 0 is 1, at level 1 there are nodes, at level 2 there are number of nodes, level three has , …, and nodes at level . The number of nodes in the entire state space tree is:](https://www.quora.com/What-is-worst-case-complexity-of-binary-search" \t "_top)

The purpose of backtracking is to avoid having to check many of these nodes by checking if a row is promising or not. The solve method itself depends on how many nodes it checked before all solutions are found. This has an upper bound of the number of promising nodes. The fact that no two queens can be placed in the same column is considered to obtain this value. For any value n, where n is the number of queens, the number of columns the queen can be placed in decreases by one each time a queen before it is placed.

For n=8, the first queen can be positioned in any of the eight columns. Once the first queen is positioned, the second can be positioned in any of the seven columns that the first queen is not in. Once the second is positioned, the third can be positioned in at most six columns, and so on. Therefore, there are at most

promising nodes

Arbitrarily the upper bound is

promising nodes

This does not consider the diagonal check in the function promising. Therefore, there could be far fewer promising nodes than this upper bound, meaning far less nodes checked to find all solutions.

**Analysis Summary:**

* The promise method takestime as it iterates through our array every time.
* For each invocation of the solve method, there is a loop which runs for time.
* In each iteration of this loop, there is promising method call which is and a recursive call with a smaller argument.

The promising method takes time as it iterates through an array of length n each time (one row).

The solve method itself depends on how many nodes it checks before all solutions are found. This value can vary but have an upper bound of the number of promising nodes, which is .

The algorithm was implemented to see the number of nodes checked vs. the n! value for different sizes n. The backtracking algorithm would only run for values of n < 18 before computation time was exceedingly long.

The figure below includes the values for brute force (), which is the upper bound on number of nodes in the pruned state space tree, or n to the power of n. The number of nodes checked if we count that no two queens can be in the same column is which is the upper bound on the number of promising nodes in the pruned state space tree.

Looking at the number of nodes checked from an algorithm as well as the upper bounds can give an idea of how much time is being saved. Given two instances with n values equal, a backtracking algorithm may check all nodes in state space tree for one instance and check very little for another instance. Because of this, an efficient time complexity will not be achieved. The bounding function are regarded as good if they substantially reduce the number of nodes that are generated.

This shows the how much time is saved backtracking since you can see on the graph which grows large after n=14.

Figure For values of n, the number of nodes checked to find all solutions is shown as well as the value for n^n, the upper bound for number of nodes in state space tree and n!, the upper bound on number of promising nodes.

The figure below contains only the n! upper bound and actual number of nodes checked backtracking. This shows the difference that checking the diagonal makes as well as the difference between the upper bound for nodes checked and the actual number checked while backtracking. It appears the backtracking algorithm does far better than the worst-case.

Figure For values of n, the number of nodes checked to find all solutions is shown as well as the value for n!, the upper bound on number of promising nodes.

The figure below shows the actual run time for the backtracking algorithm to find all solutions for sizes n=1 to n=17. Just as the number of nodes checked, the actual time is noticeably better than the upper bound since it checks diagonals in the promising method.

Figure Time for the backtracking algorithm to solve n. Values for n!, the upper bound, are also plotted.

**Analysis using the Monte Carlo method:**

The Monte Carlo technique can be used to show effectiveness of this backtracking algorithm used. This method generates a path from root to leaf and then estimate the number of nodes that would be generated in the tree for that path. The Monte Carlo method will use the same promise method the backtracking algorithm uses so that the estimate is more accurate. It then estimates the number of nodes in this pruned state space tree produced by the algorithm.

The upper bound for number of nodes the estimate algorithm gives is O(n!) since that is the bound for number of possible promising nodes.

Shown below, the average over 20 tests was taken for estimate value given from Monte Carlo algorithm and the upper bound for the entire state space tree, which is and the upper bound for promising nodes which is . The max it was able to give estimate values for was n = 20 since the *long* data type could not store any larger values, so it started showing negative values.

Figure For values of n, the estimated number of nodes that would be checked to find all solutions is shown as well as the value for n^n, the upper bound for number of nodes in state space tree and n!, the upper bound on number of promising nodes.

The figure below contains only the n! upper bound and estimated number that would be checked backtracking. The estimates are given forare shown below as well as the value for . Just as the actual backtracking algorithm, the number of nodes estimated is far under the upper bound n!. This also is due to the diagonal check in the promising method.

Figure For values of n, the estimated number of nodes that would be checked to find all solutions is shown as well as the value for n!, the upper bound on number of promising nodes.

Comparing the estimated number of nodes that would be checked and the actual number of nodes checked from backtracking algorithm can show if the estimate algorithm is somewhat accurate by looking at how similar the number of nodes are for different sizes n. The more similar they are, the more accurate the estimate at estimating the number of computations or the time it would take to find all solutions for n-Queens problem.

In the figure below (Figure 5), the input values for n=1 to n=15 are shown. The output values given from estimate method are shown in orange. This is the estimated number of nodes that the backtracking algorithm is going to check before finding all solutions. The blue data points represent the actual number of nodes that were checked. The values from the estimate method appear to be larger than the actual. This means the estimate algorithm was somewhat efficient at predicting the number nodes since the trend is similar until n=15. This means it is somewhat effective in determining how effective or ineffective this backtracking algorithm is.

Figure number of nodes checked by backtracking and the estimated number of nodes that would be checked backtracking

**Output:**

Below output is the estimated number of nodes that would be checked from Monte Carlo algorithm, the actual number of nodes checked during backtracking, and the number of promising found. The number of promising found is the number of nodes in the solution tree.

For n=1 and n=2:

Text

Description automatically generated

For n=3 and n=4:

Text

Description automatically generated

For n = 5 and n=6:

Text

Description automatically generated

For n = 7:

A picture containing electronics, keyboard

Description automatically generatedText

Description automatically generated

For n = 8:

A screenshot of a computer

Description automatically generated with low confidenceA picture containing keyboard, electronics

Description automatically generatedGraphical user interface, text

Description automatically generated

For n=9 and n=10 (I stopped printing all solutions):

A screenshot of a computer

Description automatically generated with medium confidence

For n=11 and 12:

Text

Description automatically generated

For n=13:

Text

Description automatically generated

Below is the output from running the Monte Carlo Estimate algorithm 20 times and gathering the average for sizes n=1 to n=20. It prints the estimate each iteration and then the average of 20 tests below. After 20 the sizes were too large for the long variable type in Java. Text

Description automatically generatedText

Description automatically generatedText

Description automatically generatedText

Description automatically generatedText

Description automatically generatedText

Description automatically generatedText

Description automatically generatedText

Description automatically generatedText

Description automatically generatedText

Description automatically generated

Text

Description automatically generated

Below output is the solutions for different sizes n:

Random Output:

A screen shot of a computer

Description automatically generated with low confidence