International Journal of Computer Applications (0975 – 8887)
Volume 174 – No.6, September 2017

Effect of Explicit Constraint by Comparative Study of Techniques for Solving N-Queens Problem

Omkar Buchade
Department of Computer Engineering,
All India Shri Shivaji Memorial Society's Institute of Information Technology,
Pune, Maharashtra, India

Nilesh Mehta
Department of Computer Engineering,
All India Shri Shivaji Memorial Society's Institute of Information Technology,
Pune, Maharashtra, India

Vaibhav Suryawanshi
Department of Computer Engineering,
All India Shri Shivaji Memorial Society's Institute of Information Technology,
Pune, Maharashtra, India

ABSTRACT

N-Queen is a well-known problem which states that for a given N x N chessboard, place N queens in such a way that no two queens can attack each other. Thus, two Queens should not lie in the same row, column or diagonal to each other. There are various approaches to solve this problem like Brute Force, Backtracking, Branch and Bound, Ant Colony Optimization, Particle Swarm Optimization, Genetic Algorithm, Dynamic Programming Solution, etc. [1]. In this paper, a comparative study and analysis of computation time required to solve N-Queen problem by Brute Force Search and Backtracking approach is done. The corresponding graph of computational time required by aforementioned two algorithms is plotted to analyze their performance. Further, a constraint is added to N-Queen problem where the position of the first queen in the first row is kept fixed. Backtracking approach is applied to the problem after addition of the constraint and its results are compared with Backtracking algorithm without any explicitly defined constraint. The graphical analysis gives insight into their performance. Thus, this paper would also provide the impact of an explicit constraint on Backtracking algorithm.

General Terms

Brute Force Algorithm, Backtracking Algorithm.

Keywords

N-Queens, Brute Force Search, Backtracking, Constraint Satisfaction, Heuristic.

1. INTRODUCTION

N-Queens is a classical chess problem in which placement of queens should be such that no two queens can attack each other. This can be achieved by positioning the queens such that two queens do not share the same row, column and diagonal. The eight queen puzzle was published by Max Bezzel [2] in 1848 and the first solution was found out by Franz Nauck in 1850. Nauck further extended this puzzle of N queens to N x N squares. For N=8 i.e. 8 Queens problem, Brute Force approach gives 4,426,165,368 possible arrangements out of which only 92 are valid solutions. As the value of N increases, the total computation cost increases terribly. Thus, Brute Force approach is an expensive one. In 1972 Edsgar Dijkstra, using structured programming, published a highly detailed description of Depth First Backtracking algorithm [3]. Backtracking is an algorithm which finds the solutions to computational problems such as Constraint Satisfaction Problem by building the solution in an incremental approach wherein the partial solution candidate that does not lead to a solution are abandoned. This leads to reduced arrangements as compared to Brute Force approach which in turn reduces the computation cost.

In this paper, analysis of N-Queen problem using Brute Force approach and Backtracking is performed and further, the effect of addition of an explicit constraint on the performance of the algorithm is observed. The constraint enforced is that the location of the first queen is fixed. A simple algorithm for finding all possible solutions of N x N queen problem using recursive Backtracking by addition of one more constraint, that is, fixing the position of the first queen in the first row is implemented here.

2. TERMINOLOGY

For solving the N-Queen puzzle/problem according to Erbas et.al.[4] two queens located at (x_1, y_1) and (x_2, y_2) can be placed if and only if:

1. $x_1 \neq x_2$ (not on the same row)
2. $y_1 \neq y_2$ (not on the same column)
3. $x_1 + y_1 \neq x_2 + y_2$ (not on same diagonal)
4. $x_1 - y_1 \neq x_2 - y_2$ (not on same diagonal)

Diagrammatic representation of the above constraints is shown below in figure 1.

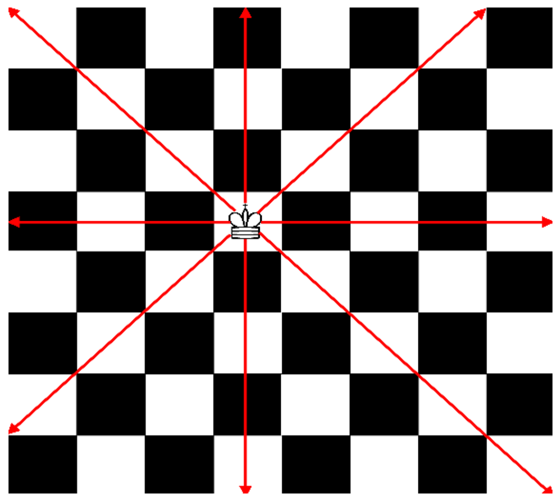

Figure 1: Queen Movement in 2-Dimensional board

The solution for N Queens exists only for $N \geq 4$ since for $N < 4$ the constraints above cannot be satisfied. The table 1 below shows the number of possible solutions for a different number of queens and the figure 2 shows a graphical representation of the same.

Table 1: Possible solutions corresponding to number of Queens

Number of Queens	Number of Solutions
3	0
4	2
5	10
6	4
7	40
8	92
9	352
10	724

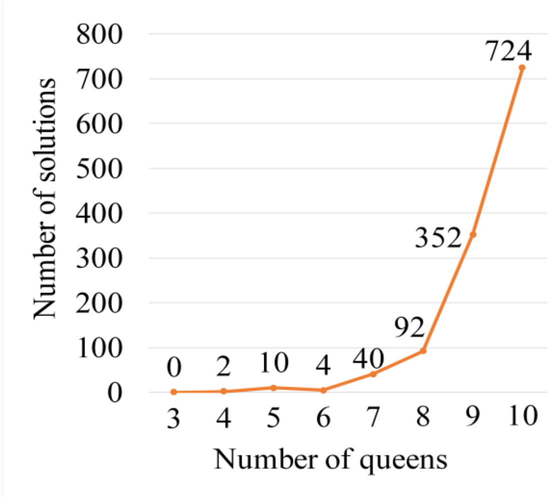

Figure 2: Graphical representation of N-Queens solutions

3. BRUTE FORCE SEARCH

Brute Force Search, also called Exhaustive Search is one of the most common problem-solving strategy, wherein all possible solutions for a given problem are checked whether it satisfies the condition required for the solution. This is one of the simplest form of problem-solving strategy where the solution is definitely generated if it exists, however, the cost required for finding the solution to a problem increases with the increase in the size of the problem. Thus, Brute Force Search is used whenever the simplicity of implementation is more important than the cost associated with the solution and thus should be always applied for a smaller instance of the problem. This technique can be improved when certain heuristic is applied which can reduce the size of the problem. For an 8-Queen problem using Brute Force Search, all possible arrangements of 8 queen pieces on a 64 square chess board are checked provided no two queens can threaten the position of each other. Brute Force Search checks for each of the 64 squares, each of the 8 queens can be placed in one of the 64 squares in 64^8 =281,474,976,710,656 ways. However, as all the queens have the same property, no two queens can be placed on the same square which reduces the possibility of choosing 8 squares from a set of 64 squares, the candidates of the solution being $64!/56!/8! = 4{,}426{,}165{,}368$ about $1/60{,}000$ of the previous estimate. Further, a constraint is applied such that two queens should not be in the same row or column or same diagonal for a solution to exist. This would reduce the candidate set for a solution [3][5].

Consider the case of solving the 4-Queen problem using Brute Force method. The possible states after applying heuristic (no two queens in the same row) are shown below in figure 3.

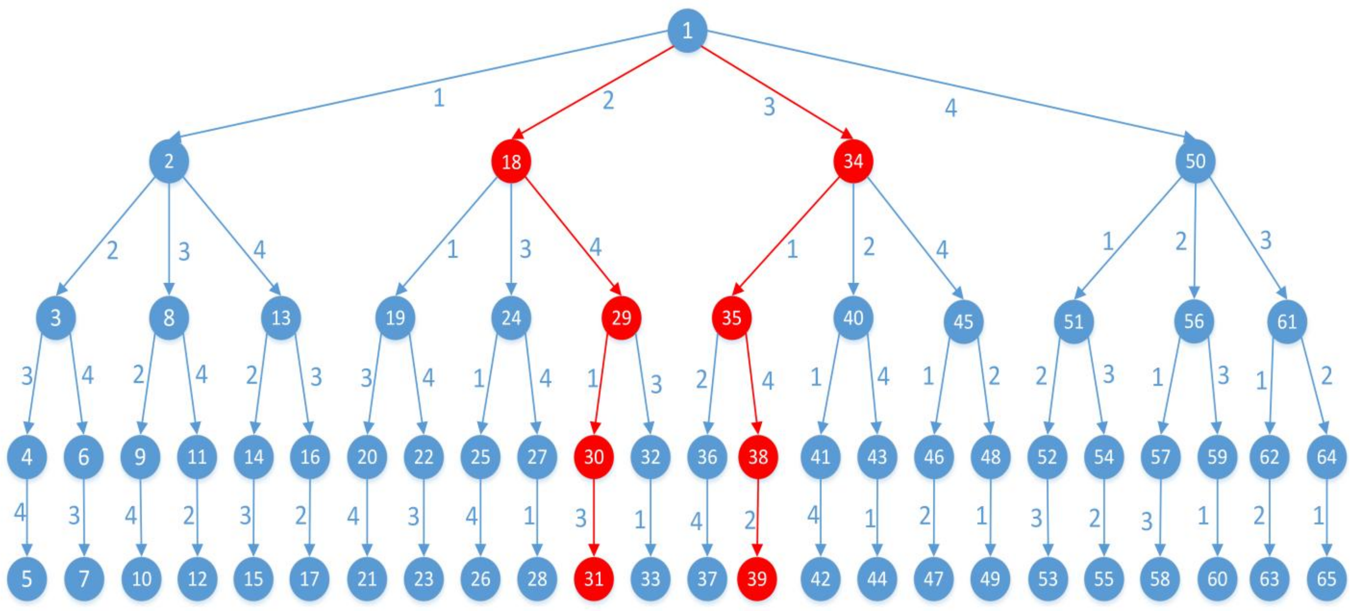

Figure 3: State Space Tree for 4 Queen problem using Brute Force approach

All the possible states are shown in the above state space tree with the solution states and paths marked in red. The total number of possible states for four queen problem after applying a heuristic (one row will have only one queen) are 65.

4. BACKTRACKING

Backtracking is an algorithm, generally recursive for finding solutions to computational problems which are constraint oriented i.e. Constraint Satisfaction Problem. It builds the solution to a problem in an incremental approach. The partial candidates are built to the complete solution until a proper solution is found. However, partial candidates are ignored as soon as it is found that partial candidate cannot lead to a solution. This reduces the number of feasible options by overlooking partial solution candidates that do not lead to a solution thus giving a smaller number of possible states reducing the time required for searching and computation. In Backtracking, one of the possible moves is selected consecutively until it leads to a solution. However, if the current move does not lead to a solution, Backtracking is performed to find some other move which may lead to a solution. This process continues till a solution is found. However, even after backtrack, if it doesn't generate a solution, it is claimed that for a given problem, solution doesn't exist [6]. Consider the case of solving the 4-Queen problem again but now with Backtracking approach. The possible states are shown below in the state space tree, with the solution states and path marked in red. It can be clearly seen from the state space tree figure 4 that the number of states required by Backtracking is 33 which is far less compared to Brute Force.

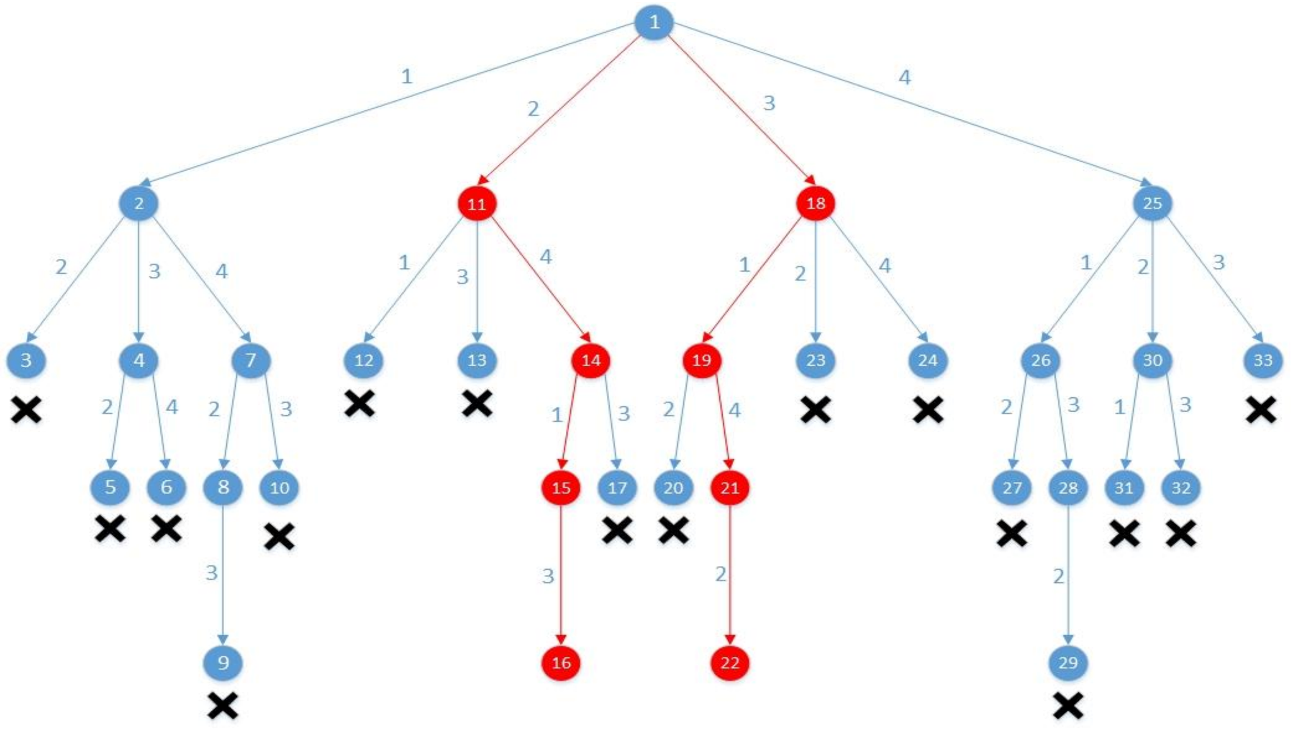

Figure 4: State Space Tree for 4 Queen problem using Backtracking approach

5. BACKTRACKING WITH ADDED CONSTRAINT

In order to improve performance over Brute Force Search approach, Backtracking was introduced which eliminated partial candidate solutions which could not lead to a complete solution. A problem may come with added constraints in accordance to which the problem needs to be solved. In the proposed work, the position of the first queen is kept fixed in the first row and its effect on computation time is found out. The aim is to find out to what extent does an introduction of a constraint affects the algorithm. Greater the constraints, smaller is the possible solution set for the problem which will eventually lead to smaller computational time.

6. IMPLEMENTATION

In this work, following algorithms are used for the purpose of implementation.

6.1 Algorithm for Brute Force

Input: No of Queens i.e. N

Output: The placement of Queens such that no two queens attack each other.

Step 1: Place queens on all possible positions.

Step 2: Check each time if the queens cannot capture each other. If not, then it has found a solution.

Step 3: Repeat step 2 until all possible solutions are found.

Note: Because of the vast amount of possible positions ($N^N$for a table of size N while each row has 1 queen), this algorithm is not practical even for small table sizes (like $N = 12$).

6.2 Algorithm for Backtracking

Input: No of Queens i.e. N

Output: The placement of Queens such that no two queens attack each other.

Step 1: Start from first row, first column

Step 2: If all queens are placed

return true

Step 3: Try all rows in the current column. And do the following for every tried row.

a) If queen is safe at position [row, column], mark this position as a part of solution and recursively check if queen at this position lead to a solution

b) Return True if the queen placed in [row, column] leads to a solution
c) Unmark this [row, column] (Backtrack) if placing Queen does not lead to a solution then go to step (a) to try other rows.

Step 4: Upon checking of all rows, if solution is not generated, return false and prompt Backtracking.

6.3 Algorithm for Backtracking with added constraint

Input: No of Queens i.e. N

Output: The placement of Queens such that no two queens attack each other.

Step 1: Place the first queen on the desired location in the first row.

Step 2: Start from second row, first column

Step 3: If all queens are placed
return true

Step 4: Try all rows in the current column. And do the following for every tried row.
a) If queen is safe at position [row, column], mark this position as a part of solution and recursively check if queen at this position lead to a solution
b) Return True if the queen placed in [row, column] leads to a solution
c) Unmark this [row, column] (Backtrack) if placing Queen does not lead to a solution then go to step (a) to try other rows.

Step 5: Upon checking of all rows, if solution is not generated, return false and prompt Backtracking.

For Experimentation, the queen is placed at index=1 where index ranges from 0 to N-1 where N is the number of Queens on board.

7. RESULT

For experimentation, number of Queens are taken as input for implementation. The system specifications used for experimentation are given below in Table 2.

Table 2. Specifications of the System Used

Hardware Used	256 GB Hard Disk and 4 GB RAM
Processor Type	Intel Core i3-4130 CPU
CPU Speed	3.40 GHz
Number of Cores	4
Operating System	Fedora 20, 64-bit
Execution Platform	Terminal

The comparison of time taken by Brute Force, Backtracking and different suggested algorithms for a number of Queens is tabulated below in table 3 and table 4 and the corresponding graphical representation is given in figure 5 and figure 6 respectively.

Table 3. Comparison between Number of Queens and Execution time in milliseconds between Brute Force(BF) and Backtracking(BT)

Number of Queens	Distinct Solutions	BF time	BT time
4	2	49	15
5	10	51	20
6	4	48	20
7	40	84	26
8	92	140	36
9	352	372	102
10	724	534	449
11	2680	2119	2018
12	14200	20899	12389
13	73712	275000	76000

Table 4. Comparison of distinct solutions and Execution time in millisecond between Backtracking (BT) without adding explicit constraint vs Backtracking by adding explicit constraint (BT with Cons)

No. of Queens	Distinct Solution	BT time	Distinct Solutions for BT with Cons	BT with cons time
4	2	15	1	12
5	10	20	2	14
6	4	20	1	11
7	40	26	7	19
8	92	36	8	36
9	352	102	30	38
10	724	449	48	69
11	2680	2018	219	219
12	14200	12389	806	983
13	73712	76000	3799	5188

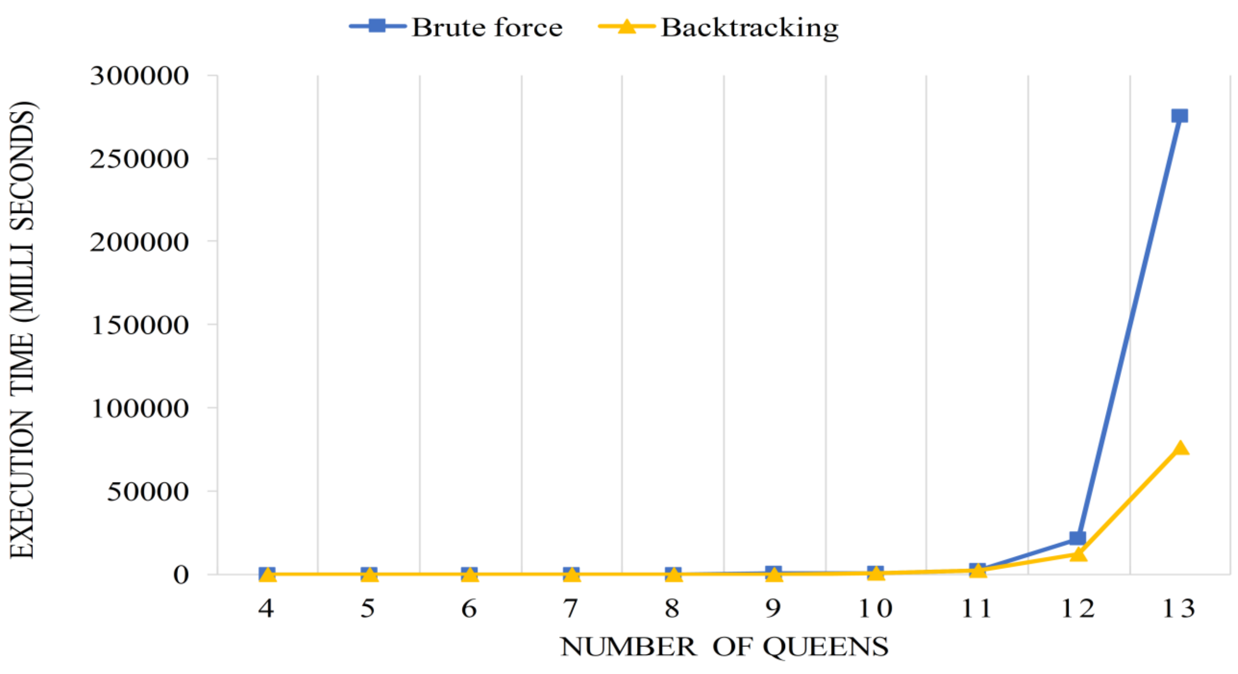

Figure 5: Graphical comparison of execution time between Brute Force approach and Backtracking approach

The graph shows the execution time of Brute Force Search and Backtracking approach for solving the N-Queens problem for different number of Queens. It can be inferred from the graph that, for smaller number of Queens the time taken by Brute Force and Backtracking is nearly same. But as the number of solutions increased, the time taken increased exponentially. This is because Brute Force tries all possible combinations available with minimal heuristics while Backtracking approach prunes and Backtracks as soon as it identifies that it cannot lead to a valid solution, thereby reducing computation time. In Brute Force Search approach Queens are placed at all possible position and only afterward it is checked for the non-attacking position. Due to this the number of solutions generated is very large as compared to Backtracking approach. Thus for smaller number of Queens, there is barely any difference between Brute Force Search and Backtracking. However, as the number of Queens increased, Backtracking clearly comes out as a better approach as Brute Force will require larger computation time and resources.

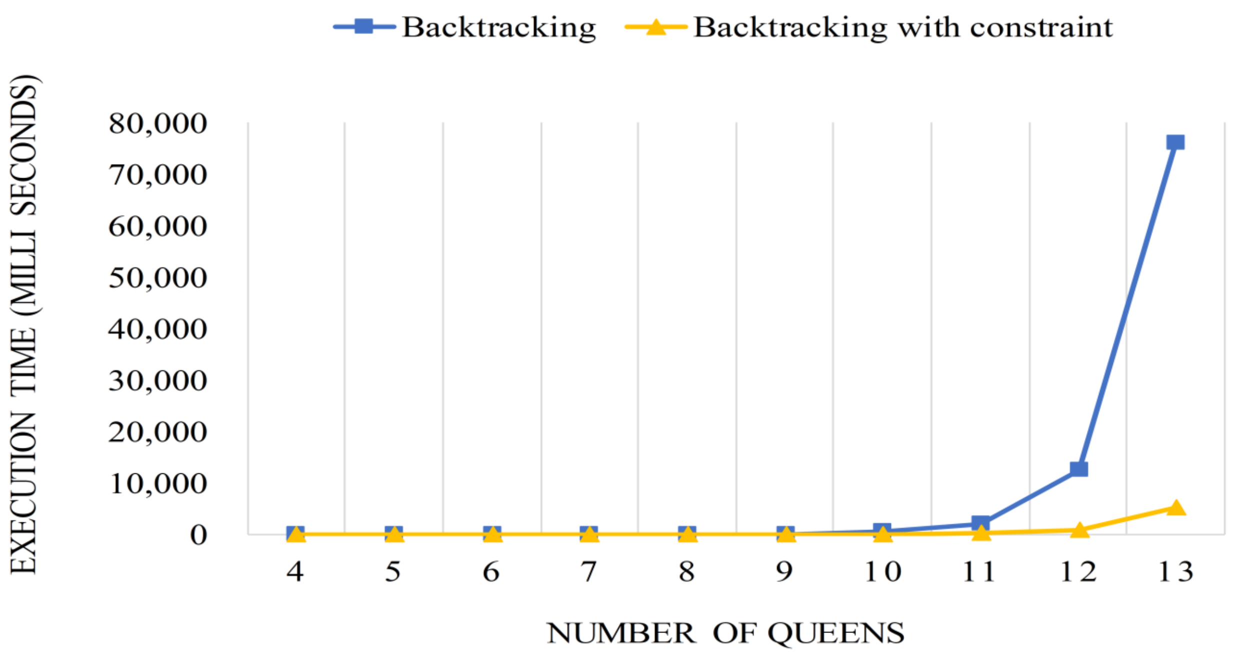

Figure 6: Graphical comparison of execution time between Backtracking without constraint and Backtracking with constraint

From the graph, it is evident that computation of N Queen problem with a constraint varies largely when compared to normal Backtracking approach. Due to constraint, the number of solutions of a problem reduces. This will require smaller amount of computation and generate faster results. Thus, N Queen problem with a constraint achieves way faster result as compared to Backtracking without a constraint.

8. CONCLUSION

This paper does a comparative study of Brute Force approach and Backtracking approach for finding solutions to a Constraint Satisfaction Problem. Here N Queen problem is considered for their comparative analysis. It is clear from the study that Brute Force Search approach generates all possible ways to place N Queens on the chess board and then tests to make sure no two queens attack each other. On the other hand, Backtracking approach identifies partial candidate solution and removes them if they don’t look promising. Thus, Backtracking approach requires less computational time as compared to Brute Force Search. The paper further analyses the impact of the presence of additional constraints in a problem. From the results obtained, it shows that a problem with additional constraint requires significantly less computation time as compared to normal Backtracking. The number of solutions generated in Backtracking with a constraint is also less. Thus, when Brute Force and Backtracking approach is employed, possible solutions for a problem are same however the computational time is faster in Backtracking. On the other hand, Backtracking with constraint has lesser possible solutions and hence computes significantly faster amongst others.

9. FUTURE WORK

This paper analyses the Brute Force Search, Backtracking without any constraint and Backtracking approach with added constraints up to number of queens being 13. This work can be extended for higher number of queens and a mathematical relation between computation time taken and the number of solutions generated can be found out. For 8 queens, 92 solutions are generated out of which only 12 are unique, as many solutions are just rotations of others. In order to avoid this repetition of result, a technique called Symmetry Breaking can be used which in turn will reduce the amount of computation needed. [7] The analysis can also be further extended to 3 and more dimensional board. [8] Implementation of Permutation Matrix which is regarded as a set of N points lying on a square of N x N chessboard such that each row or column contains only one point. Permutation Matrix can hence be used for adding heuristics to Brute Force Search and thus improving the algorithm for larger number of Queens. [9]

10. REFERENCES

[1] S. Pothumani, “Solving N Queen Problem Using Various Algorithms – A Survey”, International Journal of Advanced Research in Computer Science and Software Engineering, Volume 3, Issue 2, February 2013

[2] F.W.M. Bezzel. Proposal of eight queens problem. Berliner Schachzeitung, 1848.

[3] Eight queens puzzle [Online] https://en.wikipedia.org/wiki/Eight_queens_puzzle

[4] C.Erbas and M.M. Tanik S. Sarkeshik. “Different perspectives of the n-queens problem”, Proceedings of the 1992 ACM Annual Conference on Communications, ACM Press, page 99108, 1992

[5] Brute-force search [Online] http://en.wikipedia.org/wiki/Brute-force_search.

[6] Backtracking [Online] http://en.wikipedia.org/wiki/ Backtracking.

[7] Toby Walsh, “Symmetry Breaking”, National ICT Australia and School of CSE, University of New South Wales, Sydney, Australia, tw@cse.unsw.edu.au

[8] Soham Mukherjee, Santanu Datta, Pramit Brata Chanda and Pratik Pathak, “Comparative Study of Different Algorithms to Solve N Queens Problem”, International Journal in Foundations of Computer Science & Technology (IJFCST), Vol.5, No.2, March 2015

[9] Jordan Bell, Brett Stevens, “A survey of known results and research areas for n-queens”, Elsevier, Volume 309, Issue 1, 6 January 2009.