# N-Queens Solving Algorithm by Sets and Backtracking

Serkan Güldal
Interdisciplinary Engineering Program
University of Alabama in Birmingham
Birmingham,USA
guldal@uab.edu

Veronica Baugh, Saleh Allehaibi
Department of Electric and Computer Engineering
University of Alabama in Birmingham
Birmingham,USA
vpowell@uab.edu, salehs@uab.edu

***Abstract*—The N-Queens problem has been studied for over a century. The N-Queens problem may be solved using a variety of methods including backtracking algorithms and mathematical equations such as magic squares. We propose a more efficient approach to the most used technique, backtracking, by removing the threatened cells in order to decrease the number of trial and error steps.**

***Keywords—N-Queens; set; backtracking; algorithm; Mathematica; efficiency; optimization***

## I. Introduction

The *N*-Queens problem is a generalization of the 8-Queens problem posed by a German chess player, Max Bezzel in 1848 [1, 2, 3]. The objective of the N-Queens problem is to arrange N queens so that no queen may threaten another queen. Thus each column, row, diagonal, and anti-diagonal must contain one and only one queen. The problem has been studied for over a century by many famous mathematicians such as Gauss [1, 2], Polya [4], and Lucas [5]. The N-Queens problem is a classic example used in computer science to demonstrate various algorithms such as backtracking, permutation generation, divide-and-conquer paradigm, and neural networks [6]. The N-Queens problem is classified as non-deterministic polynomial time hard (NP-hard) which is a class of problems that are "at least as hard as the hardest problems in NP" [7].

A backtracking algorithm is an algorithm for finding all (or some) solutions to some computational problem that builds a set of solutions and abandons any partial candidate solution as soon as the partial candidate solution is deemed invalid. Backtracking depends on a trial and error method, and therefore, the more errors that are encountered, the higher inefficiency of the algorithm. The set algorithm developed by Mr. Guldal reduces the runtime of the algorithm by reducing the number of trials [8]. The proposed set and backtracking hybrid algorithm improves efficiency by removing the threatened cells from the set before the algorithm attempts to generate a solution using them. The algorithm takes advantage of sets to describe the available positions of the board at each step in the process.

## II. Set and Backtracking Hybrid Algorithm, N-Queens Problem Solver

The backtracking algorithm places the first queen and marks the other cells in the attack path of the placed queen. In the second step, the algorithm checks the next row until an unthreatened cell is found. For small boards, this process is relatively efficient. However, for boards of size 12 and bigger, the number of trials increases dramatically (See TABLE 1). This problem is solved by deleting the threatened cells before placing the next queen.

TABLE 1 COMPARISON OF THE NUMBER OF TRIALS IN BACKTRACKING ALGORITHM AND ELIMINATION BY SETS FOR 5-QUEENS [8]

| Queen # | Backtracking Algorithm No. of trials | Elimination by Sets No. of trials |
|---|---|---|
| 1 | 1 | 1 |
| 2 | 3 | 1 |
| 3 | 5 | 1 |
| 4 | 2 | 1 |
| 5 | 4 | 1 |
| Total Trials | 15 | 5 |

In our hybrid algorithm, we prepare a matrix of sets. In these sets, we have three main parts: cell number where the queen can be placed, row number where the cell is located, and the threatened cells which are threatened by the queen placed in that particular cell as follows:

$$\{\{Cell\ Number, Row\ Number\}, \{Threatened\ Cells\}\}$$

Consequently, the algorithm prepares the matrix of sets for the cells that are threatened by these queens.

Initially, our algorithm places the queen in the first cell, saves the matrix for alternative solutions when needed, and removes the threatened cells. Such elimination reduces the numbers of trials and results in a more efficient algorithm. Then, the algorithm continues placing the queens in the first available cells after eliminating the threatened cells. Finally, the algorithm checks the result to determine if the proposed solution has the same length as the board size. In the case that the length of the solution is not equal to the board size, the backtracking algorithm is used to go one step backward and restore the eliminated cells. The backtracking algorithm may go farther steps backward if needed to look for a successful trial to find all potential solutions for the N-Queens problem. In any backtracking steps, the eliminated cells by the invalid queen placement are restored and the queen is placed in the next available cell as long as it is on the same row. Otherwise, the backtracking will go farther steps backward as necessary.

**978-1-5090-2246-5/16/$31.00 ©2016 IEEE**

**Example:** Consider the 4-Queens problem to demonstrate our algorithm. Every cell is indexed as shown in Fig. 1. Initially, the matrix of sets is prepared to start attacking cells. The cells that are threatened by the queen placed on the first cell at the first row are: 2, 3, 4, 5, 6, 9, 11, 13, and 16. Accordingly, the complete matrix will be similar to the matrix representation that is shown in Fig. 2.

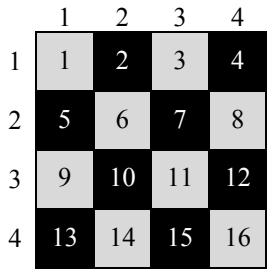


Fig. 1 4x4 Chessboard

$$\begin{pmatrix} \{1,\ 1\} & \{2,3,4,5,6,9,11,13,16\} \\ \{2,\ 1\} & \{1,3,4,5,6,7,10,12,14\} \\ \{3,\ 1\} & \{1,2,4,6,7,8,9,11\} \\ \{4,\ 1\} & \{1,2,3,7,8,10,12,13,16\} \\ \{5,\ 2\} & \{1,2,6,7,8,9,10,13,15\} \\ \{6,\ 2\} & \{1,2,3,5,7,8,9,10,11,14,16\} \\ \{7,\ 2\} & \{2,3,4,5,6,8,10,11,12,13,15\} \\ \{8,\ 2\} & \{3,4,5,6,7,11,12,14,16\} \\ \{9,\ 3\} & \{1,3,5,6,10,11,12,13,14\} \\ \{10,\ 3\} & \{2,4,5,6,7,9,11,12,13,14,15\} \\ \{11,\ 3\} & \{1,3,6,7,8,9,10,12,14,15,16\} \\ \{12,\ 3\} & \{2,4,7,8,9,10,11,15,16\} \\ \{13,\ 4\} & \{1,4,5,7,9,10,14,15,16\} \\ \{14,\ 4\} & \{2,6,8,9,10,11,13,15,16\} \\ \{15,\ 4\} & \{3,5,7,10,11,12,13,14,16\} \\ \{16,\ 4\} & \{1,4,6,8,11,12,13,14,15\} \end{pmatrix}$$

Fig. 2 Matrix of sets for 4-Queens problem

First, the algorithm places the queen in cell number 1 and saves the matrix before eliminating the threatened cells. The result is a reduced matrix which represents the board shown in Fig. 3 (See Fig. 4). Second, the algorithm places the queen in the next available cell which is cell number 7 and eliminates the threatened cells (See Fig. 5) after saving the matrix (See Fig. 6). Third, the only remaining cell to place the queen is cell number 14. Therefore, the algorithm places the queen on that cell (See Fig. 7), saves the matrix as shown in Fig. 8, and eliminates nothing since it is the last available cell. Then, the length of the solution is tested with the board size. Since they are not equal, the backtracking algorithm is used to go one step backward. Based on that, the latest matrix (shown in Fig. 6) is restored. Likewise, the backtracking algorithm is used to go one more step backward, since there is no available cell after cell number 14. Therefore, the matrix shown in Fig. 4 will be restored. Accordingly, the queen will be placed on cell number 8 instead of 7. Fourth, the threatened cells are eliminated as shown in Fig. 9 after saving that matrix as well (Fig. 10). The algorithm places the queen in cell number 10 and saves the matrix of sets before eliminating the threatened cells. Once again, the obtained length of the solution is not equal to the board size which is 4. Thus, the backtracking algorithm will be used one more time to go farther steps backward which will result in restoring the original matrix in Fig. 2. This time, the algorithm places the queen in cell number 2 and continues the same strategy until the solution with the same length as the board size is determined (See Fig. 11 and Fig. 12). Similarly, the hybrid algorithm backtracks until it finds all solutions of the 4-Queens problem depending on the set algorithm.

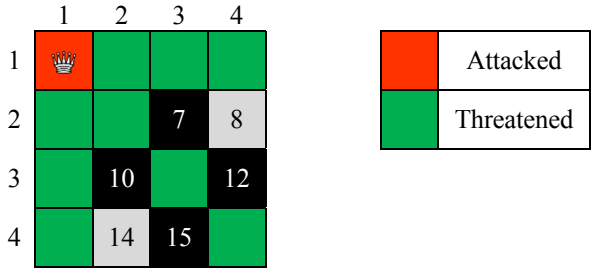


Fig. 3 Updated 4x4 Chessboard for 4-Queens solution after 1st step

$$\begin{pmatrix} \{1,\ 1\} & \{2,3,4,5,6,9,11,13,16\} \\ \{7,\ 2\} & \{2,3,4,5,6,8,10,11,12,13,15\} \\ \{8,\ 2\} & \{3,4,5,6,7,11,12,14,16\} \\ \{10,\ 3\} & \{2,4,5,6,7,9,11,12,13,14,15\} \\ \{12,\ 3\} & \{2,4,7,8,9,10,11,15,16\} \\ \{14,\ 4\} & \{2,6,8,9,10,11,13,15,16\} \\ \{15,\ 4\} & \{3,5,7,10,11,12,13,14,16\} \end{pmatrix}$$

Fig. 4 Updated matrix of sets after 1st step

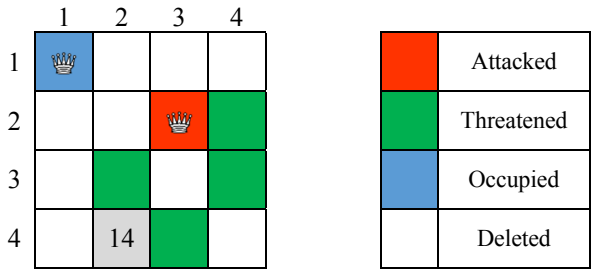


Fig. 5 Updated 4x4 Chessboard for 4-Queens solution after 2nd step

$$\begin{pmatrix} \{1,\ 1\} & \{2,3,4,5,6,9,11,13,16\} \\ \{7,\ 2\} & \{2,3,4,5,6,8,10,11,12,13,15\} \\ \{14,\ 4\} & \{2,6,8,9,10,11,13,15,16\} \end{pmatrix}$$

Fig. 6 Updated matrix of sets after 2nd step

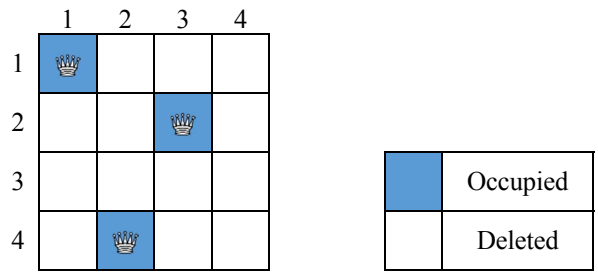


Fig. 7 Updated 4x4 Chessboard for 4-Queens solution after 3rd step

$$\begin{pmatrix} \{1,\ 1\} & \{2,3,4,5,6,9,11,13,16\} \\ \{7,\ 2\} & \{2,3,4,5,6,8,10,11,12,13,15\} \\ \{14,\ 4\} & \{2,6,8,9,10,11,13,15,16\} \end{pmatrix}$$

Fig. 8 Updated matrix of sets after 3rd step

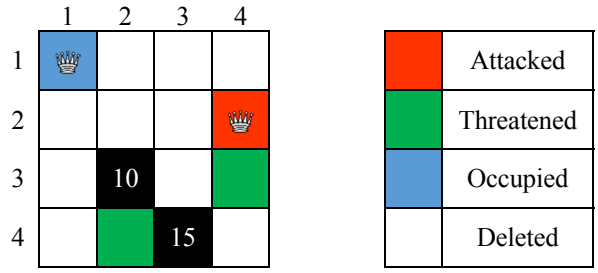


Fig. 9 Updated 4x4 Chessboard for 4-Queens solution after 4th step

$$\begin{pmatrix} \{1,\ 1\} & \{2,3,4,5,6,9,11,13,16\} \\ \{8,\ 2\} & \{3,4,5,6,7,11,12,14,16\} \\ \{10,\ 3\} & \{2,4,5,6,7,9,11,12,13,14,15\} \\ \{15,\ 4\} & \{3,5,7,10,11,12,13,14,16\} \end{pmatrix}$$

Fig. 10 Updated matrix of sets after 4th step

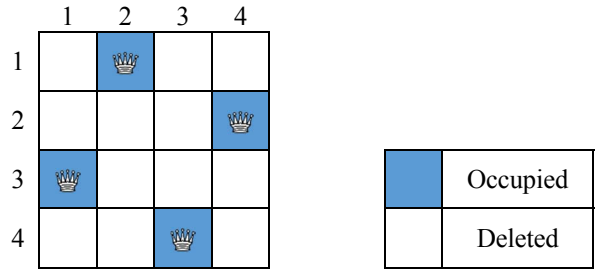


Fig. 11 4-Queens solution on 4x4 Chessboard

$$\begin{pmatrix} \{2,\ 1\} & \{1,3,4,5,6,7,10,12,14\} \\ \{8,\ 2\} & \{3,4,5,6,7,11,12,14,16\} \\ \{9,\ 3\} & \{1,3,5,6,10,11,12,13,14\} \\ \{15,\ 4\} & \{3,5,7,10,11,12,13,14,16\} \end{pmatrix}$$

Fig. 12 4-Queens solution

## III. Results and Discussion

As a proof of concept, we implemented two algorithms using Mathematica to solve the well-known N-Queens problem. The first one is using the backtracking algorithm, while the other one is the proposed hybrid model. The goal of this case study is to measure the efficiency of our algorithm as compared to the classic backtracking algorithm. Additionally, we recorded the runtimes of both of these algorithms on several computers to see the variation of performance based on the computers' specifications as well as operating systems.

The proposed algorithm is more efficient and solves the N-Queens problem in less time than the backtracking algorithm. The gap in efficiency between the two algorithms is dramatically increased after the 12-Queen problem in favor of the proposed algorithm. With increased board size, the proposed algorithm performs with increasing improvement over the traditional backtracking algorithm (See Fig. 13). The runtimes of both the backtracking and the hybrid algorithms may be represented by an exponential trend-line as shown in Fig. 13.

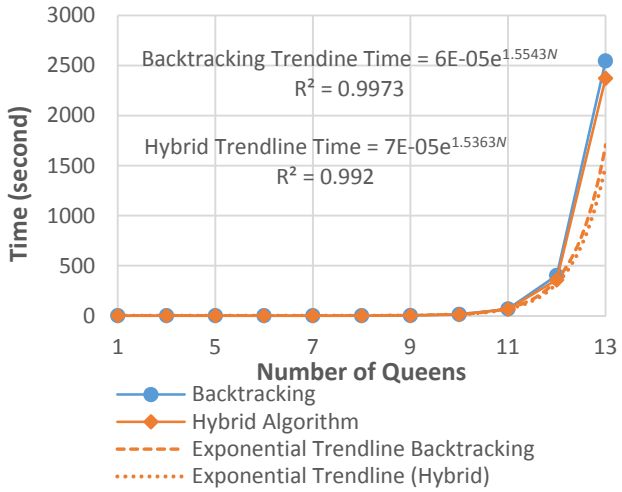


Fig. 13 Comparison of Backtracking and Hybrid Algorithms

Next, we compare the variation of the algorithm performance based on various computer specifications and operating systems. We executed the algorithms on three different computers with the features listed in TABLE 2. As expected, there is an inverse correlation between computers' specifications and the runtimes, as the computer specifications increased the runtimes decreased as shown in TABLE 3 and Fig. 14.

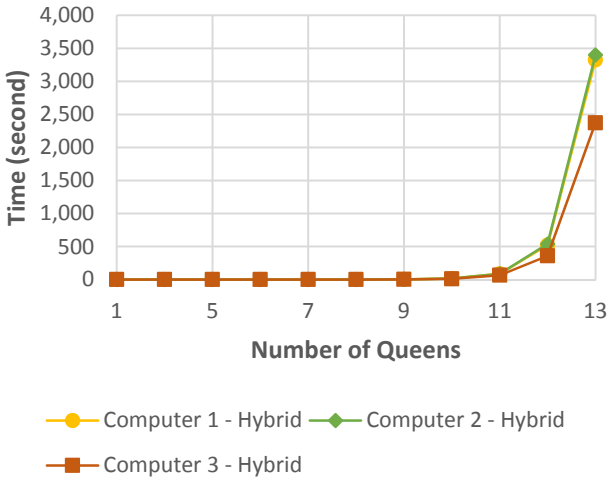


Fig. 14 Comparison of Performance in Executing Hybrid Algorithm on Various Computers

In TABLE 3, we have calculated the optimization percentage based on the accumulative difference between the Backtracking and Hybrid algorithms runtimes divided by the accumulative average of the runtimes for the two algorithms, represented as a percentage by using below equation:

$$Optimization = \frac{Backtracking\ time - Hybrid\ time}{\left(\frac{Backtracking\ time + Hybrid\ time}{2}\right)} \times 100\%$$

The optimization percentage of hybrid algorithm varies from computer to another which could be justified by the operating system and hardware configuration. This is demonstrated in Fig. 15, Fig. 16, and Fig. 17 to show performance variation of executing backtracking and hybrid algorithm in all three computers.

TABLE 2 COMPUTERS' FEATURES

| Features | Computer 1 | Computer 2 | Computer 3 |
|---|---|---|---|
| CPU | Intel(R) Xeon(R) CPU E5-2609 v2 @ 2.50 GHz | Intel(R) Core(TM) i7-2630QM CPU @ 2.00 GHz | Intel Core i5 2.6 GHz |
| RAM | 32 GB | 6 GB | 16 GB 1600 MHz DDR3 |
| Operating System | Windows 7 Enterprise | Windows 10 Pro | OS X Yosemite Version 10.10.5 |

In Fig. 14, we compared the effects of different hardware and operating systems. We expected computer 1 should be the fastest among the three computers. However, MacBook, which is computer 3, achieves the shortest time to finish the calculation. One of the reasons is that Mathematica has better compatibility with Mac OS. Another reason might be that the MacBook CPU has higher frequency.

TABLE 3 COMPARISON OF ALGORITHMS PERFORMANCE ON VARIOUS COMPUTERS

| N-Queens | Runtimes (Seconds) | | | | | |
|---|---|---|---|---|---|---|
| | Computer 1 | | Computer 2 | | Computer 3 | |
| | Backtracking | Hybrid | Backtracking | Hybrid | Backtracking | Hybrid |
| 1 | 0 | 0 | 0 | 0 | 0.000289 | 0.000178 |
| 4 | 0 | 0 | 0 | 0 | 0.002533 | 0.003528 |
| 5 | 0 | 0 | 0.015625 | 0.015625 | 0.007399 | 0.01256 |
| 6 | 0.0312 | 0.0312 | 0.03125 | 0.046875 | 0.025145 | 0.02692 |
| 7 | 0.171601 | 0.156001 | 0.140625 | 0.15625 | 0.12502 | 0.11591 |
| 8 | 0.748805 | 0.686404 | 0.71875 | 0.75 | 0.58242 | 0.466075 |
| 9 | 3.572423 | 3.18242 | 3.4375 | 3.26563 | 2.798172 | 2.333989 |
| 10 | 17.815314 | 16.114903 | 17.4531 | 16.2188 | 14.06615 | 11.722245 |
| 11 | 96.861021 | 87.703762 | 95.7969 | 87.5938 | 67.478512 | 65.569083 |
| 12 | 558.077977 | 522.60335 | 568.266 | 539.234 | 401.994278 | 359.384065 |
| 13 | 3599.707475 | 3324.006908 | 3796.11 | 3402.36 | 2545.083225 | 2372.736646 |
| Total | 4276.985816 | 3954.484948 | 4481.96975 | 4049.64098 | 3032.163143 | 2812.371199 |
| Optimization | 7.84% | | 10.13% | | 7.52% | |

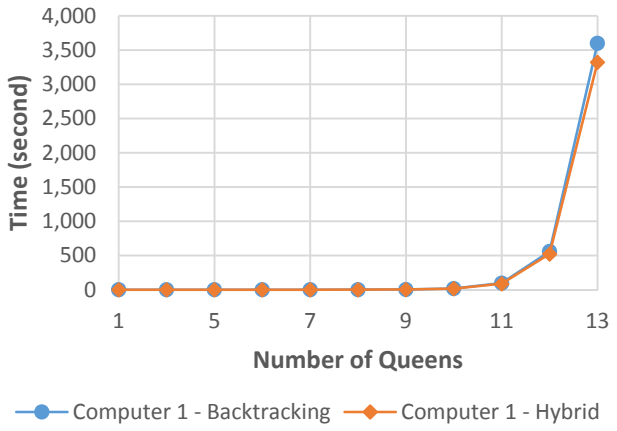


Fig. 15 Comparison between Backtracking and Hybrid in Computer 1

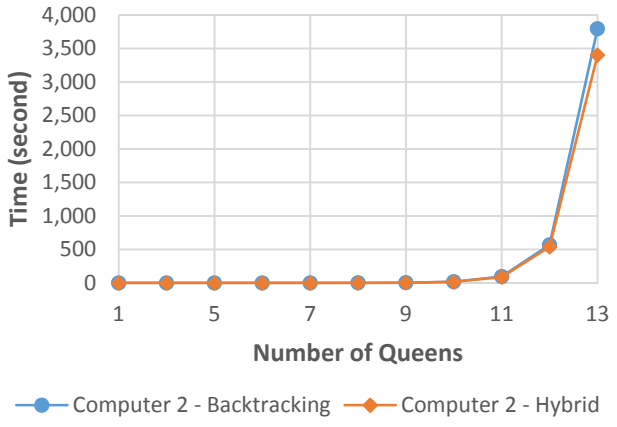


Fig. 16 Comparison between Backtracking and Hybrid in Computer 2

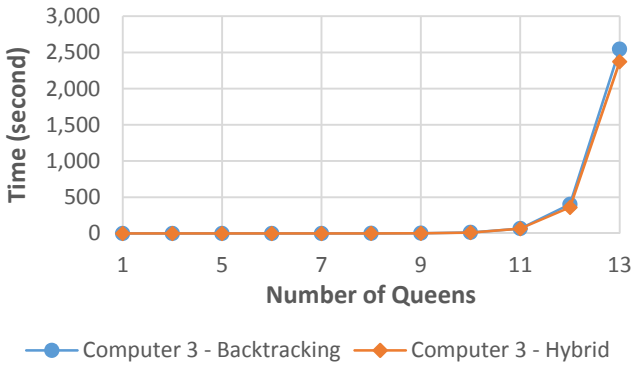


Fig. 17 Comparison between Backtracking and Hybrid in Computer 3

In addition to hardware properties, there are independent sets that could be programmed as small modules in the hybrid algorithm to improve code readability and debugging.

Since the hybrid algorithm is based on the sets, mathematical tools applied to the sets could improve the performance.

## IV. Conclusion

In this paper, we demonstrated that the efficiency of the traditional backtracking algorithm may be improved by the use of a hybrid approach taking advantage of sets to reduce the number of trial and error attempts.

Future work will include a complexity analysis to compare the proposed hybrid algorithm with a variety of widely used N-Queens algorithms.

## V. Acknowledgment

We are grateful to Dr. Tanik who introduced the N-Queens problem to us and encouraged us to study the N-Queens problem in detail.

## VI. References

[1] C. F. Gauss and H. C. Schumacher, *Briefwechsel zwischen,* pp. 105-122, 1865.

[2] J. Gingsburg, "Gauss's arithmetrization of the problem of n queens," *Scripta Math. 5,* pp. 63-66, 1939.

[3] S. Gunther, "Zur mathematisches theorie des Schachbretts," *Archiv der Mathematik und Physik,* vol. 56, pp. 281-292, 1874.

[4] G. Polya, "Uber die 'doppelt-periodischen' losungen des n-damen-problems," *Mathematische Unterhaltungen und Spiele,* pp. 364-374, 1918.

[5] E. Lucas, Recreations mathematiques, 1891.

[6] C. Erbas, S. Sarkeshik and M. M. Tanik, "Different Perspectives of the N-Queens Problem," *ACM,* pp. 99-108, 1992.

[7] Z. Wang, D. Huang, J. Tan, T. Liu, K. Zhao and L. Li, "A parallel algorithm for solving the n-queens problem based on inspired computational model," *BioSystems,* pp. 22-29, 2013.

[8] S. Güldal, *N-Queens Solution Algorithm by Using Sets,* Gatlinburg, TN: Poster presented at ACM Mid-Southeast 2015 Conference, 2015.

## APPENDIX

## BACKTRACKING ALGORITHM

```
Timing[n=8;
  board=Table[0,{i,n},{j,n}];
  solutions={};
  tt=0;
  tt1={};
  For[i=1,i<=n,i++,
   For[j=1,j<=n,j++,
    If[board[[i,j]]==0,

      board[[i,j]]=Q;
      (*Row*)Table[If[k!=j,board[[i,k]]=board[[i,k]]+1],{k,1,n}];
      (*Column*)Table[If[k!=i ,board[[k,j]]=board[[k,j]]+1],{k,1,n}];
      (*lDiagonalLower=====================*)
      k=i+1;
      l=j+1;
      While[k!=n+1&&l!=n+1,board[[k,l]]=board[[k,l]]+1;k++;l++];
      (*rDiagonalLower=====================*)
      k=i+1;
      l=j-1;
      While[k<=  n&&l>= 1,board[[k,l]]=board[[k,l]]+1;k++;l--];
      (*====================================*)
      ];
    ];

   If[i==n,
    solution={};

Table[If[MemberQ[board[[row]],Q],posQ=Position[board[[row]],Q][[1]][[1]];AppendTo[solution,posQ];];,{row,n}]
;
    If[Length@solution==n,AppendTo[solutions,solution]];
    ];


   (*This part stops calculation after queen moved from last column to first column in the first row*)
   If[tt==0,If[board[[1,n]]==Q,tt++]];
   If[tt==1&&board[[1,1]]==Q,Break[]];


   (*Backtracking Loop=====================*)
   If[i==n,
    test=True;
    While[test,
     If[MemberQ[board[[i]],Q],
      pos=Position[board[[i]],Q][[1]][[1]];
      board[[i,pos]]=0;

      j=pos;
      (*Row*)Table[If[k!=j,board[[i,k]]=board[[i,k]]-1],{k,1,n}];
      (*Column*)Table[If[k!=i ,board[[k,j]]=board[[k,j]]-1],{k,1,n}];
      (*lDiagonalLower=====================*)
      k=i+1;
      l=j+1;
      While[k!=n+1&&l!=n+1,board[[k,l]]=board[[k,l]]-1;k++;l++];
      (*rDiagonalLower=====================*)
      k=i+1;
      l=j-1;
      While[k<=  n&&l>= 1,board[[k,l]]=board[[k,l]]-1;k++;l--];
      (*====================================*)

      Table[ If[board[[i,iter]]==0,
         board[[i,iter]]=Q;

         j=iter;
         (*Row*)Table[If[k!=j,board[[i,k]]=board[[i,k]]+1],{k,1,n}];
         (*Column*)Table[If[k!=i ,board[[k,j]]=board[[k,j]]+1],{k,1,n}];
         (*lDiagonalLower=====================*)
         k=i+1;
         l=j+1;
         While[k!=n+1&&l!=n+1,board[[k,l]]=board[[k,l]]+1;k++;l++];
         (*rDiagonalLower=====================*)
         k=i+1;
         l=j-1;
```

```
          While[k<=  n&&l>= 1,board[[k,l]]=board[[k,l]]+1;k++;l--];
          (*====================================*)


          test=False;

          ];,{iter,pos+1,n}];
        i--; If[i==0,Break[]];
        ,i--;If[i==0,Break[]];
        ];If[i==0,Break[]];

       ];
      ];
    ];][[1]]
```

## BACKTRACKING AND SET: HYBRID ALGORITHM

```
Timing[n=13;
  board=Table[j+n i,{i,0,n-1},{j,n}] ;

  rows={};
  Table[i=Position[board,q][[1,1]];
   row={q};
   Table[If[board[[i,c]]!=q,AppendTo[row,board[[i,c]]]];,{c,n}];
   AppendTo[rows,row];
   ,{q,n^2}];

  columns={};
  Table[j=Position[board,q][[1,2]];
   column={q};
   Table[If[board[[r,j]]!=q,AppendTo[column,board[[r,j]]]];,{r,n}];
   AppendTo[columns,column];
   ,{q,n^2}];

  lDiagonals={};
  Table[i=Position[board,q][[1,1]];
   j=Position[board,q][[1,2]];
   lDiagonal={q};
   Table[If[i+li<= n && j+li<=n,AppendTo[lDiagonal,board[[i+li,j+li]]]];,{li,n}];
   Table[If[i-li>= 1 && j-li>= 1,AppendTo[lDiagonal,board[[i-li,j-li]]]];,{li,q}];
   AppendTo[lDiagonals,lDiagonal];
   ,{q,n^2}];

  rDiagonals={};
  Table[i=Position[board,q][[1,1]];
   j=Position[board,q][[1,2]];
   rDiagonal={q};
   Table[If[i+li<=n && j-li>= 1,AppendTo[rDiagonal,board[[i+li,j-li]]]];,{li,n}];
   Table[If[i-li>= 1 && j+li<=n,AppendTo[rDiagonal,board[[i-li,j+li]]]];,{li,n}];
   AppendTo[rDiagonals,rDiagonal];
   ,{q, n^2}];

  attackable={};
  Table[AppendTo[attackable,Union[rows[[i]],columns[[i]],rDiagonals[[i]],lDiagonals[[i]]]],{i,n^2}];

  attack=Table[{{i, Ceiling[i/n]},attackable[[i]]},{i,n^2}];
  Table[attack[[i,2]]=Delete[attack[[i,2]],Position[attack[[i,2]],i]],{i,n^2}] ;

  (*====================================*)
  sol=attack;
  solutions={}; (*List of solutions*)
  states=Table[{Null},{iter,n}]; (*To save the matrix before delete anything*)
  i=0;

  While[n+1!=sol[[All,1]][[1,1]]&& Length[sol]>=i, i++; (*Place the queen*)
   If[i>Length[sol],i=i-1];

   If[Dimensions[states[[i]]][[1]]==1,states[[i]]=sol,
    If[states[[i]][[All,1]][[1;;i-1,1]]===sol[[All,1]][[1;;i-1,1]],

      qq=sol[[All,1]][[i,1]];
      posqq=Position[states[[i]][[All,1]][[All,1]],qq][[1]][[1]];

      If[Length[states[[i]][[All,1]]]>posqq,
       If[states[[i]][[All,1]][[posqq,2]]==states[[i]][[All,1]][[posqq+1,2]],
         ,states[[i]]=sol;];
       ,states[[i]]=sol;];
```

```
      ,states[[i]]=sol;];
    ];

   threatenedCells=sol[[i,2]];
   availableCells=sol[[All,1]][[All,1]];
   survivors=Complement[availableCells,threatenedCells];

survivorsPos=Table[Position[sol[[All,1]][[All,1]],survivors[[iter]]][[1]],{iter,Length@survivors}][[All,1]];
   sol=sol[[survivorsPos]];

   If[Length[availableCells]==n && i==n,AppendTo[solutions,availableCells]];

   If[i>=  Length[sol],
    iter=True;While[iter, If[i==0,Break[]];

     bq=sol[[All,1]][[i,1]]; (*Backtracked Queen*)
     If[Length[states[[i]]]>i,sol=states[[i]];
      bqPos=Position[sol[[All,1]][[All,1]],bq][[1]][[1]];

      If[sol[[bqPos,1]][[2]]==sol[[bqPos+1,1]][[2]],
       ibackup=i;i=bqPos+1;

       threatenedCells=sol[[i,2]];
       Do[ (*Delete cells under attack*)
        availableCells=sol[[All,1]][[All,1]];
        Do[
         If[k==j,sol=Delete[sol,Position[sol[[All,1]][[All,1]],k][[1]][[1]]]]
         ,{k,availableCells}];
        ,{j,threatenedCells}];

       i=ibackup;iter=False;

       ,i=i-1];
      ,i=i-1];
     ];
    ];If[i==0,Break[]];
   ];
  ][[1]]
```