Volume 6, Issue 4, April 2016 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Development of Algorithms for Ranking and Un-Ranking Function of N-Queens Problem

Nilesh Gupta*

Department of CSE, Kashi Institute of Technology, Varanasi, Dr. A.P.J Abdul Kalam Technical University, Lucknow, Utter Pradesh India

Abstract— The n-queens Problem, introduced in 1850 by Carl Gauss, is an extension of the original 8-queens problem. The problem requires us to find the placement of N queens on an NXN chessboard such that no queen is attacking the other. Only one queen must be positioned per row, per column and per diagonal. This problem belongs to the category of combinatorial problems, which require a lot of time and effort to be solved since there is no set formula for solving them. To find the most optimal solution, we need to consider every possible solution. There are methods such as heuristics to solve this problem. With increase in the size of the problem, the number of possibilities also increases exponentially. The n-queens problem can have many distinct solutions for each value of n. The n-queens problem is a popular classic puzzle where number of queens to be placed on n×n board such that no queen can attack any other queens. This paper mainly composed of some problems like (i) To evaluate all the solutions for given number of queens.(ii) Ranking Function: For given solution, to find rank or index of that solution in set of solution space (iii) Un-ranking Function: For given index find that solution in set of solutions.

Keywords— Algorithms for ranking function, un- ranking function, N-Queens Problems ,C implementation .

I. INTRODUCTION

1.1 What is N-Queens Problem?

The n-queens problem is a generalized form of 8-queens problem, proposed by the chess player Max Bezel. In 8- queen problem, 8 queens are required to be placed on an 8x8 chess board in such a way that no queen attacks any other queen. A queen can move in horizontal (in the same row), vertical (in the same column) and diagonal direction. Also an n-queens problem must follow the following rules:

1. There is at most one queen in each column.
2. There is at most one queen in each row.
3. There is at most one queen in each diagonal.

The 8-queens problem is computationally very expensive since the total numbers of possible arrangements of queen are $64! /(56! \times 8!) \sim 4.4 \times 10^9$ and the total number of possible solutions are 92. We can consider two solutions to be the same and can obtain one from the other by rotation or symmetry. So there exists only 12 different solutions and it becomes very hard to obtain a unique solution out of these. The n-queens problem follows the same rules as in 8-queens problem with n queens and an n×n chessboard. This problem falls in a special class of problems NP hard whose solution cannot be found out in polynomial time.

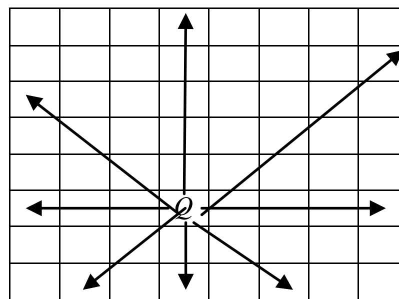


Fig.1

In the above figure queen is at position (6, 4) arrows show that no other queens can stay in the way of arrows.

1.2 N-Queens and CSP

We can represent the n-queens problems as a constraint satisfaction problem. A Constraint Satisfaction Problem consists of 3 components

- A set of variables.
- A set of values for each of the variables.
- A set of constraints between various collections of variables.

We must find a value for each of the variables that satisfy all of the constraints. We need to know where to place each of the n queens. So we could have n variables each of which has as a value $1 \ldots N^2$. The values represent where on the chessboard we will place the ith variable.

Q							
						Q	
				Q			
							Q
	Q						
			Q				
					Q		
		Q					

Fig.2

$Q_1 = 1$, $Q_2 = 15$, $Q_3 = 21$, $Q_4 = 32$, $Q_5 = 34$, $Q_6 = 44$, $Q_7 = 54$, $Q_8 = 59$

This representation has 64^8 =281,474,976,710,656 different possible assignments in the search space. In this case we know that we can never place two queens in the same column. So we can configure the problem as one where we assign one queen to each of the columns, and now we need to find out only which row each of these queens is to be placed in. So we can have n variables: $Q_1, Q_2 \ldots, Q_N$

The set of values for each of these variables will be $\{1,2,\ldots,N\}$

Q							
				Q			
							Q
					Q		
		Q					
						Q	
	Q						
			Q				

Fig.3

$Q_1 = 1$, $Q_2 = 5$, $Q_3 = 8$, $Q_4 = 6$, $Q_5 = 3$, $Q_6 = 7$, $Q_7 = 2$, $Q_8 = 4$.

This representation has 8^8 =16,777,216 different possible assignments in the search space. It is still too large to examine all of them, but a big improvement. The constraints are the key component in expressing a problem as a CSP.

Each constraint consists of

- A set of variables.
- A specification of the sets of assignments to those variables that satisfy the constraint

The idea is that we break the problem up into a set of distinct conditions each of which have to be satisfied for the problem to be solved.

In n-queens:

No queen can attack any other queen.

Given any two queens Q_i and Q_j they cannot attack each other.

Now we translate each of these individual conditions into a separate constraint.

Q_i cannot attack Q_j $(i \neq j)$

- Q_i is a queen to be placed in column i, Q_j is a queen to be placed in column j.
- The value of Q_i and Q_j are the rows the queens are to be placed in.

Queens can attack each other

- Vertically, if they are in the same column, this is impossible as Q_i and Q_j are placed in different columns.
- Horizontally, if they are in the same row, we need the constraint $Q_i \neq Q_j$.
- Along a diagonal ,they cannot be the same number of columns apart as they are rows apart, we need the constraint $|i-j| \neq |Q_i-Q_j|$

A solution to the n-queens problem will be any assignment of values to the variables $Q_1,\ldots,Q_n$ that satisfies all of the constraints

© 2016, IJARCSSE All Rights Reserved

1.3 A Brief History

In 1848 chess player Max Bezel proposed the 8-queens problem and since then many mathematicians including Gauss, have worked on this problem and its generalized form i.e. n-queens problem. The first solution to the 8-queens problem was found out by Franz Nauck in 1850. Nauck later extended this problem to n-queens (placing n queens on an n×n chessboard such that no queen attacks any other queen using the standard moves). In 1874 S. Gunther proposed a method of finding solutions by using determinants and J.W.L. Glaisher refined this approach. Alternatively, search-based algorithms have been developed. For example, a backtracking search, an algorithm generates all possible solution of a given n×n board (Bitner and Reingold, 1975 and Purdom and Brown, 1983).

1.4 Number of Solutions for Different Number of Queens

It is a big challenge to determine the number of solutions when the size of the board increases. This is especially true for the "normal" n-queens problem

Table.1

Number of queens	*Total number of solutions*
1	*1*
2	*0*
3	*0*
4	*2*
5	*10*
6	*4*
7	*40*
8	*92*
9	*352*
10	*724*
11	*2680*
12	*14200*
13	*73712*
14	*365596*
15	*2279184*
16	*14772512*
17	*95815104*
18	*666090624*
19	*4968057848*
20	*39029188884*
21	*314666222712*
22	*7016442691008*
23	*24233937684440*
24	*227514171973736*
25	*2207893435808350*
26	*22317699616364000*

II. VARIOUS METHODS OF SOLVING N-QUEENS PROBLEM

2.1 A Novel Method for Solving N-Queens Problem, [6]

The n-queens problem can have many distinct solutions for each value of n. we present a pattern that is observed while finding the unique solutions for every value of n. Using this pattern, we can leverage the results of an $n \times n$ board to find a unique solution for an (n+1) × (n+1) board in polynomial time.

2.1.1: The Pattern

2.1.1.1: Solution for 4×4

The n-queens problem have a solution for all N>3. Thus, the first solution is obtained when N=4, i.e., for a 4×4 board. There are two possible solutions for a 4-queens problem, as shown in figure 4 and figure 5. We can see that one is a reflection of the other. Thus, we can build our pattern from figure 4 and extend it to figure 5 later.

First solution of 4 queens problem

		Q	
Q			
			Q
	Q		

Fig.4

© 2016, IJARCSSE All Rights Reserved

2.1.1.2: Solution for 5✕5

In the 4✕4 solution board, add one row at the top and one column to the right to convert the board into a 5✕5 one. The 5th queen must now be placed at the intersection of the newly added row and column, i.e. at position (1,5). This is one of the unique solutions for 5✕5 board.

				Q
		Q		
Q				
			Q	
	Q			

Fig.5

2.1.1.3: Solution for 6✕6

To the solution of the 5✕5 board, add one new row after (6/2 = 3) rows and a new column after the rightmost column. The new board is now 6✕6. Again, the 6th queen will be placed at the intersection of the newly added row and column, i.e. at position (4,6). This is a unique solution for the 6✕6 board.

				Q	
		Q			
Q					
					Q
			Q		
	Q				

Fig.6

2.1.1.4: Solution for 7✕7

In the 6✕6 solution board, add a new row at the top and a new column after the rightmost column to convert the board into a 7✕7 one. The 7th queen must now be placed at the intersection of the newly added row and column, i.e. at position (1,7). This is a unique solution for the 7✕7 board.

						Q
				Q		
		Q				
Q						
					Q	
			Q			
	Q					

Fig.7

2.2 Minimal Conflict Algorithms

The role of minimal conflict algorithm is in improving genetic algorithm. Each state of search-space of the problem can be a candidate for solution. To remember, each cell of decision variable's array corresponds to a column of chessboard. This algorithm moves along candidate's array and by reaching to each column which its queen is in conflict with the other queens, tries to place it in a better row. If there is more than one location with least conflicts (has more than one choice) one of them is selected randomly. Eventually the result of this operation led to reducing conflicts on entire chessboard. In below figure, an example of how algorithm works is illustrated. In this figure, the algorithm is looking at the third column of chessboard. In part 'a', the queen belongs to the seventh row is dealing with three conflicts. Then the algorithm assesses the situation of other cells in the third column, in order to find a better place for the queen. Finally it concludes that fifth row with just one conflict is a better place for the queen. This change has been applied in part 'b'.

Part a:

8	3	4	2	Q	1	1	2	2
7	1	4	Q	1	2	2	2	1
6	Q	3	3	4	2	3	1	2
5	3	2	1	4	Q	1	3	1
4	2	1	3	1	3	Q	1	2
3	1	Q	2	2	2	3	4	2

© 2016, IJARCSSE All Rights Reserved

2	**2**	**2**	**2**	**2**	**2**	**1**	**Q**	**4**
1	**2**	**1**	**2**	**2**	**1**	**3**	**1**	**Q**

Fig.11, [1]

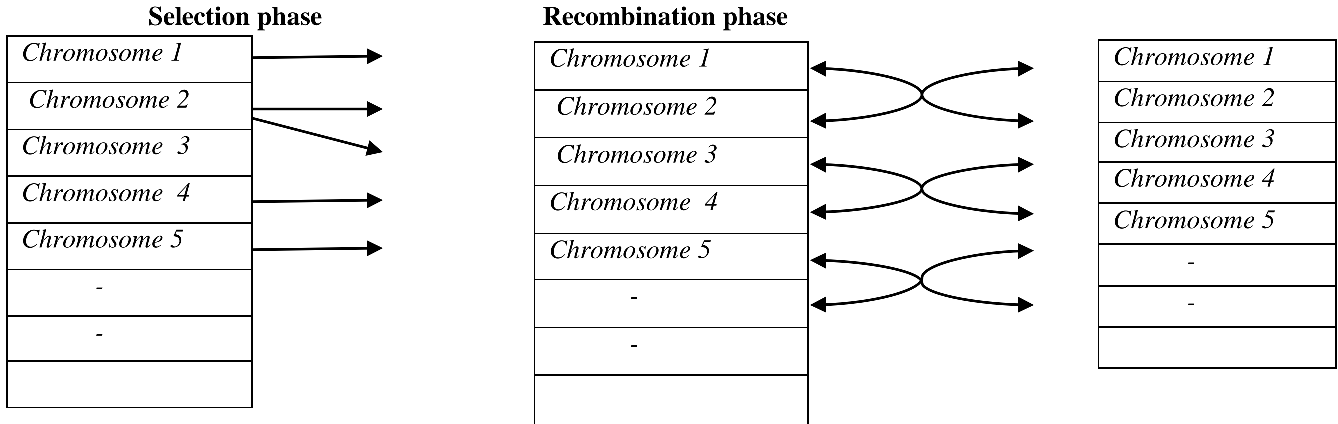
Part b:

8	**3**	**3**	**2**	**Q**	**1**	**2**	**2**	**2**
7	**1**	**3**	**3**	**0**	**2**	**1**	**1**	**0**
6	**Q**	**3**	**3**	**4**	**2**	**3**	**1**	**2**
5	**3**	**3**	**Q**	**5**	**Q**	**2**	**4**	**2**
4	**2**	**2**	**3**	**2**	**3**	**Q**	**1**	**2**
3	**2**	**Q**	**2**	**2**	**3**	**3**	**3**	**1**
2	**2**	**2**	**2**	**2**	**2**	**2**	**Q**	**3**
1	**2**	**1**	**2**	**2**	**1**	**3**	**2**	**Q**

Fig.12, [1]

Each permutation of possible values of the decision variable can be a candidate to problem solution. These candidates are also called chromosomes. A collection of candidates are called population. Genetic algorithm is consisted of several operators. Applying these operators cause population modification and during these modifications new generations are created.

Below figure shows that in this implementation of genetic algorithm, each iteration is divided into selection phase and recombination phase.



(a) shows current population, (b) shows intermediate population (c) shows next population.

Fig.13,[1]

Fitness function is measure of distance or proximity of a candidate for problem's solution which is shown by f(x). At the beginning of selection phase, current-population is initialized

by initial population, at the first iteration. In other iterations it is initialized by next-population belongs to previous iteration. Then, fitness function assesses candidates and each candidate will be labelled by fitness-value which shows how candidate is close to the answer. If the answer is found then the algorithm stops but if the answer is not found, intermediate-population would be composed based on current population, during selection operation. Among the algorithms that are used for selection operation. Here we used roulette wheel technique for selection operation. The probability of being selected through roulette wheel is calculated from the following formula [9]

$$P_i = \frac{f_i}{\sum_{i=1}^{k} f_i}$$

f_i is the fitness function and k is the size of population. Before continuing, intermediate population must be shuffled sufficiently.

During recombination phase, next-population is created by applying crossover and mutation on candidates from intermediate-population. Crossover operator chooses a pair of candidates. Then it recombines them with the probability P_c to form two new candidates. Crossover operator has various types like 1-point crossover, 2-point crossover.

2.2.2: 1-Point Crossover

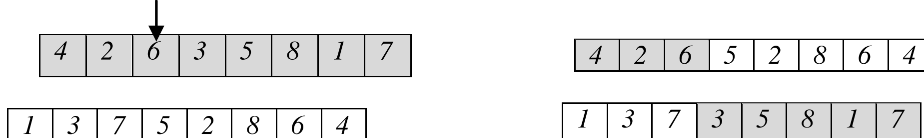


Fig.14,[1]

© 2016, IJARCSSE All Rights Reserved

After crossover operation has been carried out, we can apply mutation operation. Each value from candidates in the population can be changed under the probability Pm.

2.2.3: Mutation

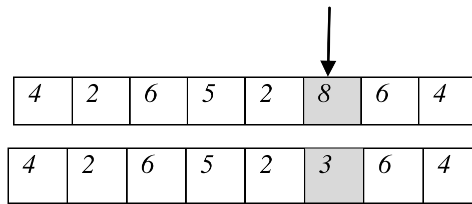


Fig.15,[1]

2.3 Solution of N-Queens Problem Using Ant Colony Optimization

2.3.1: Background of ACO

The ant colony optimization (ACO) is inspired by intelligent behaviours of ants. Ant behaviour depicts the behaviour of an agent in the system. The first ant algorithm was developed by Dorigo. Ants in nature modify their environment by constantly depositing a chemical substance called pheromone. The pheromone is used as an indirect communication among ants and guides them to find shortest path from their nest to a food place. If there are multiple paths to a food place, ants choose one with high concentration of pheromone. In Figure 16, we can logically find that how a shortest path will be concentrated with the high pheromone values as most of the ants on this path will come back quickly

2.3.2: Simple ACO Algorithm [2]

Let G = (V, E) be a connected graph where |V | shows total number of nodes and |E | shows total number of edges in graph. The simple ant colony optimization can be used to find the shortest path between a given source node V_s and a given destination node V_d in the graph 'G'. The path length is either given by the number of nodes on the path or summation of cost values on edges constituting the path. Each edge e(i, j) ϵ E of the graph connecting the nodes V_i and V_j has a variable (artificial pheromone), which is modified by the ants when they visit the nodes.

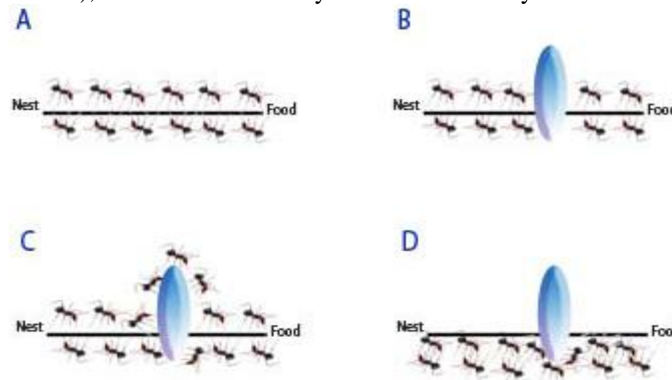


Fig .16,[2].

Ants finding shortest path

i. No obstacle.
ii. Obstacle Placed.
iii. Most ants on shortest path and a few on longest.
iv. Ants found shortest path.

From a node, when an ant decides which node to move next, it uses 2 parameters to calculate the probability of moving to a particular node, first distance to that node and second is amount of pheromone on the connecting edge. Let $d_{i,j}$ be the distance between the nodes i and j, the probability p_{ij} [2] that the ant chooses j as the next city after it has arrived at city i where j is in the set 'S' of cities that have not been visited is

$$p_{i,j} = \frac{[\tau_{i,j}]^{\alpha}[\eta_{i,j}]^{\beta}}{\sum_{k\in S}[\tau_{i,k}]^{\alpha}[\eta_{i,k}]^{\beta}} \quad . \ . \ . \ . \ . \ . \ . \ . \ . \ . \ . \ . \ (1)$$

Where $\tau_{i,j}$ is the pheromone value on edge (i, j), $\eta_{i,j}$ is a heuristic value calculated as 1/di,j . The parameters α and β are influencing factors of pheromone value and heuristic value respectively. Some best ants or all ants modify the pheromone values on the edges added to their tour. One possible modification may be done as [2]

$$\tau_{i,j} = \tau_{i,j} + \frac{Q}{L}$$

Where Q is some constant and L is the length of the tour, small the value of L high the pheromone value added to the previous pheromone value on edge.

With time, concentration of pheromone decreases due to diffusion affects, a natural phenomenon known as evaporation. This also ensures that old pheromone should not have a too strong influence on the future. This can be done as [2]

© 2016, IJARCSSE All Rights Reserved

$$\tau_{I,J} = \tau_{I,J} \cdot \rho$$

Where ρ will be between 0 and 1

2.3.3: Solution/ Search Space

In order to solve the n-queen problem, basic ACO is modified and some constraints are added. Moreover, the equations described above are also changed a little bit. The cores of this
solution include "formation of search space" & "calculation of heuristic value". In figure 17 formation of search space is described which makes the search efficient and fast, the vertices in search space is organized as a grid of n^2 rows and n columns. Every vertex in a column is connected to all the vertices in the next column through directed edges except vertices in n^{th} column. The vertices label shows the chess cell position e.g. label 1 is used as a mapping to cell [0, 0], label 2 is used as a mapping to cell [0, 1] and n^2 is used as a mapping to cell [n-1, n-1]. For 8-queen problem or 8×8 chess board positions, n will be 8, above grid will have 648 vertices, 64×64×7 edges and n^2 i.e. 64 will map to cell [7,7] at chess board. In order to simplify the things, we will consider n equal to 8 i.e. we will now talk about 8-queen problem, as from the search space above it is evident that solution to 8-queen problem is easily extendible to n-queen problem.

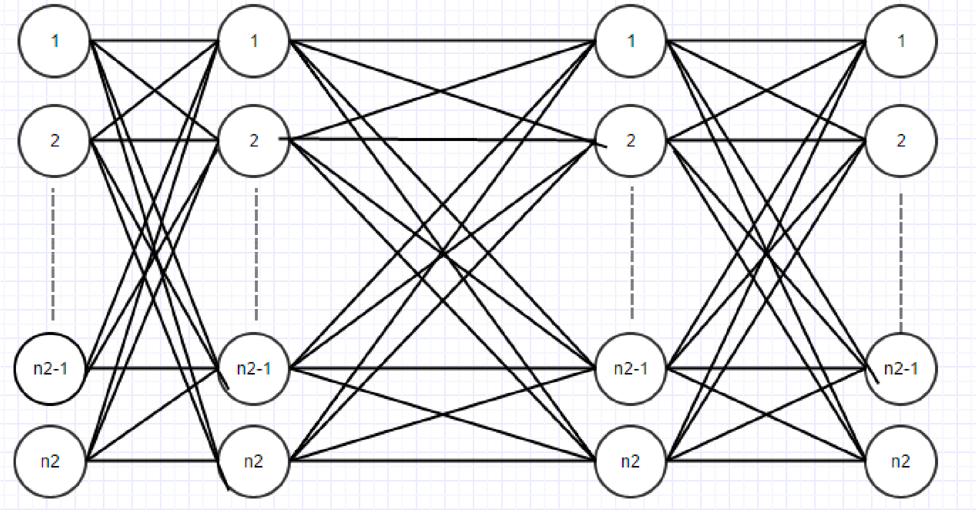


Fig.17, [2].

2.3.4: Constraints Added in Basic ACO

An ant can only move from left to right. Once a node is selected at certain column, it can pick the next node to move at, only from the column, next to the current column. Tour ends over last column. An ant during a tour visits only 'n' nodes (8 for 8-queen). The label of nodes in the tour cannot be the same. The node in a tour represents a cell of chessboard where a queen is to be placed. This restriction is added as if two nodes have same labels then 2 queens will be placed in same cell which is illegal. This essentially means that no two nodes in a tour will
be in the same row of the search space.

2.3.5: ACO Parameters Initialization and Equations

Initially, all the edges are assigned with the same small pheromone values. Arbitrary number of ants (swarm size) is created as multi-agent system. Pheromone value is modified on the basis of fitness value of a solution found by an ant. Fitness Value the fitness represents the number of positions at chessboard that satisfy the game constraint i.e. queens if placed on these positions will not kill any other queen at chessboard. In case of 8-queen problem the possible range of fitness values is 0-8. Heuristic Value and Evaporation: The probability equation is same as described above but the calculation of heuristic value is changed. Now, in Equation (1), heuristic value is not based on the distance between two vertices rather number of contradictions if node j is selected as next node. Contradictions represent the positions at chessboard where if queens are placed will kill each other. Note that node j is a chess position
at which a queen will be placed. The probability equation [2] for selecting a node is

$$p_{i,j} = \frac{[\tau_{i,j}]^{\alpha}[\eta_{i,j}]^{\beta}}{\sum_{k \in S}[\tau_{i,k}]^{\alpha}[\eta_{i,k}]^{\beta}}$$

Further, value of α and β is initialized with a random number generated in range 0-2. The evaporation is done on constant rate in our solution, decaying the pheromone value on all the edges is done after the completion of a single iteration. A single iteration is completed when all the ants complete their tour. The evaporation is done by subtracting the current pheromone value of an edge from a small constant.

2.4 Solving N-Queen Problem by Backtracking

4 queens problem is the simplest problem namely "n-queens problem". The basic idea is how to place n queens on n by n board, so that they don't attack each other. The complexity of solving the problem increases with n. We will briefly

© 2016, IJARCSSE All Rights Reserved

introduce solution by backtracking. First let's explain what is backtracking? The board should be regarded as a set of constraints and the solution is simply satisfying all constraints. For example Q_1 attacks some positions, therefore Q_2 has to comply with these constraints and take place, not directly attacked by Q_1. Placing Q_3 is harder, since we have to satisfy constraints of Q_1 and Q_1. Going the same way we may reach point, where the constraints make the placement of the next queen impossible. Therefore we need to relax the constraints and find new solution. To do this we are going backwards and finding new admissible solution. To keep everything in order we keep the simple rule last placed, first displaced. In other words if we place successfully queen on the ith column but cannot find solution for $(i+1)^{th}$ queen, then going backwards we will try to find other admissible solution for the ith queen first. This process is called backtrack

Let's discuss this with example. For the purpose of this handout we will find solution of 4 queens problem.

Algorithm:

- Start with one queen at the first column first row
- Continue with second queen from the second column first row
- Go up until find a permissible situation
- Continue with next queen

We place the first queen on A1:

4				
3				
2				
1	*Q*			
Start	*A*	*B*	*C*	*D*

4				
3				
2				
1	Q			
start	A	B	C	D

Fig.18

Note the positions which Q_1 is attacking. So the next queen Q_2 has two options: B3 or B4.

4				
3		*Q*		
2				
1	*Q*			
start	*A*	*B*	*C*	*D*

4				
3		*Q*		
2				
1	*Q*			
start	*A*	*B*	*C*	*D*

Fig.19

We choose the first one B3.

Again with shaded we show the prohibited positions. It turned out that we cannot place the third queen on the third column (we have to have a queen for each column). In other words we imposed a set of constraints in a way that we no longer can satisfy them in order to find a solution. Hence we need to revise the constraints or rearrange the board up to the state which we were stuck. Now we may ask a question what we have to change. Since the problem happened after placing Q_2 we are trying first with this queen. We know that there were to possible places for Q_2. B3 gives problem for the third queen, so there is only one position left – B4

4		*Q*		
3				
2				
1	*Q*			
start	*A*	*B*	*C*	*D*

4		*Q*		
3				
2				
1	*Q*			
start	*A*	*B*	*C*	*D*

Fig.20

As you can see from the new set of constraints (the shaded positions) now we have admissible position for Q_3, but it will make impossible to place Q_4 since the only place is D3. Hence placing Q_2 on the only one left position B4 didn't help. Therefore the one step backtracking was not enough. We need to go for second backtrack. The reason is that there is no position for Q_2, which will satisfy any position for Q_4 or Q_3. Hence we need to deal with the position of Q_1. We have started from Q_1 so we will continue upward and placing the queen at A2

4				
3				
2	*Q*			
1				
start	*A*	*B*	*C*	*D*

4				
3				
2	*Q*			
1				
start	*A*	*B*	*C*	*D*

Fig.21

© 2016, IJARCSSE All Rights Reserved

Now it is easy to see that Q_2 goes to B4, Q_3 goes to C1 and Q_4 takes D3:

4		*Q*		
3				*Q*
2	*Q*			
1			*Q*	
start	*A*	*B*	*C*	*D*

Fig.22

To find this solution we had to perform two backtracks. So what now? In order to find all solutions we use backtrack. Start again in reverse order we try to place Q_4 somewhere up, which is not possible. We backtrack to Q_3 and try to find admissible place different from C1. Again we need to backtrack. Q_2 has no other choice and finally we reach Q_1. We place Q_1 on A3:

4				
3	*Q*			
2				
1				
Start	*A*	*B*	*C*	*D*

4			*Q*	
3	*Q*			
2				*Q*
1		*Q*		
start	*A*	*B*	*C*	*D*

Fig.23

Continuing further we will reach the solution on the right. Is this distinct solution? No it is rotated first solution. In fact for 4×4 board there is only one unique solution. Placing Q_1 on A4 has the same effect as placing it on A1. Hence we explored all solutions how implement backtrack in code. Remember that we used backtrack when we cannot find admissible position for a queen in a column. Otherwise we go further with the next column until we place a queen on the last column. Therefore your code must have fragment

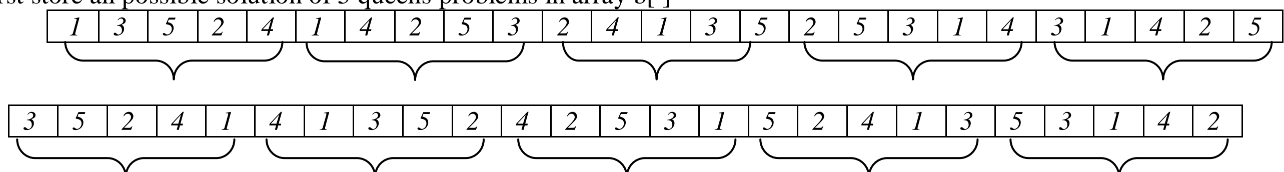
III. ALGORITHMS FOR RANKING AND UN-RANKING FUNCTIONS OF N-QUEENS PROBLEM

3.1 Process for Generating Solutions:-

Input- Number of queens.
Output- All possible solutions.
Take input as no of queens;
If no of queens is either 0 or 1 or 3
Return no solutions;
Else
Solutions exist;
If Solution Exist
Take first row
Check each column,
If any column of this row is safe
Mark it as solution and go to next row
Continue this process till next safe place,
If there is no safe place in this row
Come back to previous row and check next column of this row
Continue this process until no of marks position is equal to number of queens and print this solution,
This is the first solution, for next solution continue this process from position of next queen.

3.2 Process for Finding Solution for Given Index:-

Store all possible solution in a character array;
Take input index of solution
Go to proper place of array
Print N×N elements of character array
This is required solution
Example: Find 6th solution of 5 queens problem.
First store all possible solution of 5 queens problems in array b[ ]



1	*3*	*5*	*2*	*4*	*1*	*4*	*2*	*5*	*3*	*2*	*4*	*1*	*3*	*5*	*2*	*5*	*3*	*1*	*4*	*3*	*1*	*4*	*2*	*5*

3	*5*	*2*	*4*	*1*	*4*	*1*	*3*	*5*	*2*	*4*	*2*	*5*	*3*	*1*	*5*	*2*	*4*	*1*	*3*	*5*	*3*	*1*	*4*	*2*

Fig.24

Now count 6th block of b[ ] which is (3,5,2,4,1) is required solution.

3.3 Process for Finding Index for Given Solution:-

Take input solution in a character array a[ ];
Compare N character of already stored solution and input solution;
If N×N elements are all equal
This is required result
If any one of the N×N elements do not match this means this is not a required solution
Go to next N×N elements comparison.

3.4 C Implementation of Ranking and Un-Ranking process

```
#include<stdio.h>
#include<math.h>
#include<ctype.h>
char s[500000000];
int k=1;
int count_solution=0;
void reverse_rank_function(int n)
{
   char a[500];
   int i,x=1,count,rank=1;
   top2:
   printf("enter %d*%d solution matrix in a single row\n\n",n,n);
   scanf("%s",a);
    top:
   count=0;
   for(i=0;i<n*n;i++)
   {
      if(a[i]==s[x])
      {
         count++;
         x++;
      }
      else
      x++;
   }
   if(count==n*n)
   {
      printf("rank of solution\n");
      printf("%d\n",rank);
     goto end;
   }
   else
   {
     rank++;
     if(rank>count_solution)
     {
      printf("this is wrong solution matrix\nplease give correct solution matrix\n\n");
      goto top2;
     }
   }
    goto top;
    end:

      printf("\n\n");
}
void ranking_function(int n)
{
   int m,i,j;
   int count=0;
   char *p=s+1;
   top1:
   printf("enter rank of solution\n\n");
```

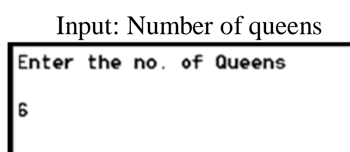
```
    scanf("%d",&m);
    if(m<1 || m>count_solution)
    {
    printf("wrong input--->please give correct input\n\n");
    goto top1;
    }
    m=m-1;
    j=n*n*m;
    p=p+j;
    for(i=0;i<n*n;i++)
    {
     printf("%c    ",*(p+i));
      count++;
      if(count%n==0)
      printf("\n\n");
     }
}
void print_grid(int n,int x[])
{
    char arr[20][20];
    int i,j;
    for(i=1;i<=n;i++)
    {
        for(j=1; j<=n; j++)
        {
            arr[i][j]='.';
        }
    }

    for(i=1;i<=n;i++)
    {
        arr[i][x[i]]='Q';
    }

    for( i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
        {
        //  if(arr[i][j]=='Q')
        //  printf("%d ",j);
            printf("%c    ",arr[i][j]);
            s[k]=arr[i][j];
            k++;
        }
            printf("\n\n");
    }
    count_solution++;
}

/* For checking Queens placement is safe or not */
int safetoplace(int x[],int k)
{      int i;
    for(i=1;i<k;i++)
    {
        if(x[i]==x[k]||i-x[i]==k-x[k]||i+x[i]==k+x[k])
        {
            return 0;
        }
    }
    return 1;
}
```

© 2016, IJARCSSE All Rights Reserved

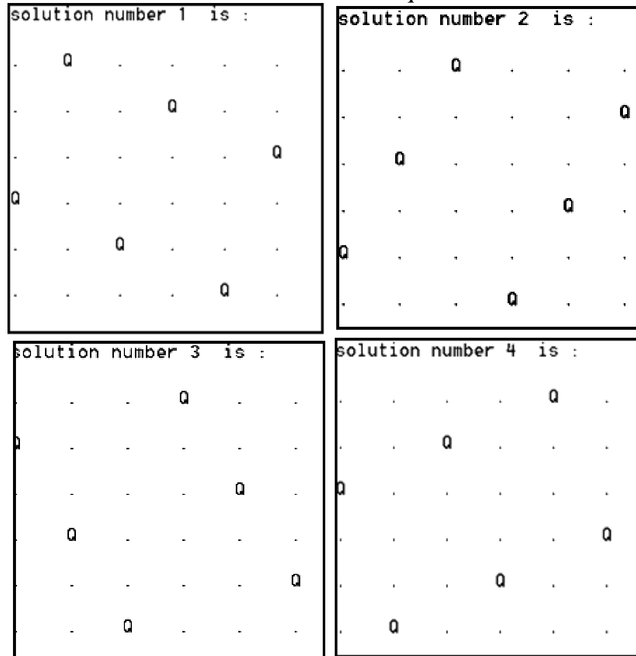
```
/* For printing the Queens and Placing them in Grid */
void nqueens(int n)
{
   int x[20];
   int count=0;
   int k=1;
   x[k]=0;
   while(k!=0)
   {
      x[k]=x[k]+1;
      while((x[k]<=n)&&(!safetoplace(x,k)))
      {
         x[k]=x[k]+1;
      }
      if(x[k]<=n)
      {
         if(k==n)
         {
            count++;
            printf("\nsolution number %d  is : \n\n",count);
            print_grid(n,x);
         }
         else
         {
            k++;
            x[k]=0;
          }
      }
      else
      {
         k--;
      }
   }
   printf("total number of solution=%d\n\n",count_solution);
   return;
}

/* Main Function */
int main()
{
   int n;
   printf("Enter the no. of Queens\n\n");
   scanf("%d",&n);
   if(n==0||n==2||n==3)
   {
      printf("no solution exist\n\n");
      goto end;
   }
   nqueens(n);
   ranking_function(n);
   reverse_rank_function(n);
end:
      printf("\n\n");

}
```

3.5 Output of Ranking

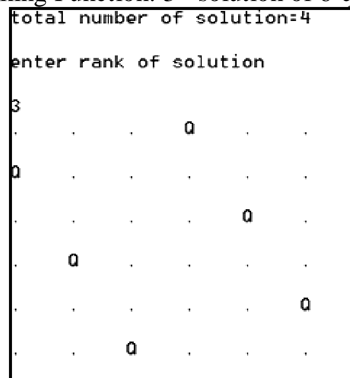
Input: Number of queens



Four solutions of 6-queens



Ranking Function: 3rd solution of 6-queens



IV. UN-RANKING FUNCTION

Means if rank of solution is given, then we have to find that particular solution. For example 8 queens problem has total 92 solutions, suppose you want to find 10th solution, then a function which find 10th solution among those 92 solutions is called un-ranking function. Suppose n is number of queens, r is rank for which solution is to be find then a function f

$f : S \rightarrow N$

Where S is the set of number of all solution, 1,2 .... ,92 in case of 8 queens.

N is set of natural number from 1 to factorial of n. n is fixed.

Function f is defined by

$f(n,r)=i$

Means that r^{th} solution is i^{th} permutation of (1,2,....n). Where i is defined as

$$i = \begin{cases} \text{Checked,} & \text{if } count = r \\ \text{Not exist,} & checked > n! \end{cases}$$

Where count = count permutation which is solution.

Checked = all checked permutation until count=r;

Example: find f(4,2), i.e. find second solution of 4 queens.

Initialise count =0;

And checked = 0;

Now generate permutation of (1,2,3,4)

Table 2

Permutation	Number	Solution or not	checked	count
(1,2,3,4)	1	Not	1	0
(1,2,4,3)	2	Not	2	0
(1,3,2,4)	3	Not	3	0

© 2016, IJARCSSE All Rights Reserved

(1,3,4,2)	4	Not	4	0
(1,4,2,3)	5	Not	5	0
(1,4,3,2)	6	Not	6	0
(2,1,3,4)	7	Not	7	0
(2,1,4,3)	8	Not	8	0
(2,3,1,4)	9	Not	9	0
(2,3,4,1)	10	Not	10	0
(2,4,1,3)	11	Yes	11	1
(2,4,3,1)	12	Not	12	1
(3,1,2,4)	13	Not	13	1
(3,1,4,2)	14	Yes	14	2

Now count =2 which is equal to r, then i=checked=14.
Thus required solution is 14th permutation of (1,2,3,4) which is (3,1,4,2);

4.1 C Implementation of Un-Ranking Function

```
main()
{
long long int n, r,count=0,i,j,k1,k2;
   long long int m;
   printf("enter number of queens\n");
   scanf("%lld",&n);
   printf("enter rank of solution\n");
   scanf("%lld",&r);
   m=fact(n);
   for(i=1;i<=m;i++)
   {
   long long   int a[n+1],b[n+1];
      for(j=0;j<=n;j++)
      a[j]=j;
      perm(i,n+1,a,b);
      k1=rule1(n+1,b);
      if(k1==1)
      {
         k2=rule2(n+1,b);
            if(k2==1)
            {

               count++;
            }
      }
      if(count==r)
      {
         printf("%lldth solution is=\n",r);
         for(i=1;i<=n;i++)
         printf("%lld ",b[i]);
         break;
      }
   }
}
```

4.2 Output of Un- Ranking Function

```
enter number of queens
10
enter rank of solution
100
100th solution is=
2 8 10 4 1 5 9 6 3 7
--------------------------------
Process exited with return value 0
Press any key to continue . . .
```

Here window shows that for 10 queens problem, 100th solution is (2,8,10,4,1,5,9,6,3,7).

© 2016, IJARCSSE All Rights Reserved

```
enter number of queens
15
enter rank of solution
100
100th solution is:
1 3 5 12 9 11 14 7 15 13 2 8 6 4 10
--------------------------------
Process exited with return value 0
Press any key to continue . . .
```

Here window shows that for 15 queens problem, 100th solution is (1,3,5,12,9,11,14,7,15,13,2,8,6,4,10).

V. CONCLUSIONS

N-queens problem is very interesting program among computer programmer as well as mathematician; there are several methods to solve this problem like Novel method, Genetic Algorithms, Ant Colony Optimization, backtracking etc. But there is no method for ranking and un-ranking function; in this project my work is to find a way to find ranking and un-ranking function. Since to develop these functions I need all solutions of n-queens problem in lexicographic order, so I have used Backtracking method to solve n-queens. To generate permutation of (1,2,3,4......n) I have used a new technique which is dependent on project Euler 24 which is very efficient to find i^{th} permutation directly among all n! Permutations. Also I have found two rules by applying which we can easily find that i^{th} permutation is solution of n-queens problem or not. In old method it is used very simple concept in which all solutions of n-queens problem are stored in an array and we can reach at desired place by counting all block , one block means one solution, but in new method I have used lexicographic order of permutation which is easy to understand and easy to implement. Finally I have written two function ranking and un-ranking function respectively.

REFERENCES

[1] Modified Genetic Algorithm for Solving n-Queens Problem -Jalaleddin Aghazadeh heris Faculty of Mathematics and Computer Science Allameh Tabataba University Tehran, Iran j.aghazadeh@st.atu.ac.ir Mohammadreza Asgari Oskoei Faculty of Mathematics and Computer Science Allameh Tabataba University Tehran, Iran oskoei@atu.ac.ir.

[2] Solution of n-Queen Problem Using ACO-Salabat Khan, Mohsin Bilal, M. Sharif, Malik Sajid, Rauf BaigNational University Of Computer and Emerging Science Islamabad, Pakistan Email: salabat.khan@nu.edu.pk Telephone: (+92)51-4532308

[3] International Journal of Computer Applications (0975 – 8887) Volume 43– No.12, an Unique Solution for N queen Problem April 2012

[4] Vishal Kesri, Prasant Ku. Pattnaik School of Computer Engineering KIIT University, India Vaibhav Kesri Department of Electrical Engineering NIT Kurukshetra, India.

[5] International Journal of Advanced Research in Computer Science and Software Engineering Solving N Queen Problem Using Various Algorithms – A Survey S. Pothumani Department of CSE, Bharath University,

[6] International Journal of Advanced Research in Computer Science and Software Engineering- A Novel Method for Solving N-Queens Problem by Avani Gupta , S.Ravi Rohith and Satya Pramodh Mazumdar SCSE, VIT University, India.

[7] Lexicographic generation of ordered tree by S. ZAKS , Department of Computer Science University of Illinois Urbana IL 61801 U.S.A.

[9] Hynek, J. Genetic Algorithms for the N-Queens Problem. Available at: http://nguyendangbinh.org/Proceedings/IPCV08/Papers/GEM4444.pdf

[10] 3rd N QUEENS ETSI PlugtestsTM CONTEST Counting the number of solutions Single and Distributed Program.

© 2016, IJARCSSE All Rights Reserved