*Journal of the Nigerian Association of Mathematical Physics*
*Volume 10 (November 2006)*, **107 -114**
*© J of NAMP*

# The turning points in the solution of *n*-queens problem using backtracking method

[1]***S. C. Chiemeke*** and [2]***E. O. Osaghae***
**[1]Department of Computer Science, University of Benin, Benin City, Nigeria.**
**e-mail: schiemeke@yahoo.com**
**[2]Department of Computer Science, Delta State Polytechnic, P.M.B 03, Otefe-Oghara, Delta State, Nigeria.**

**Abstract**

**Conventional backtracking method has been the generally accepted method for solving *n*-queens problem. However, this method may prolong execution time for fairly large n-queens (example, *n* = 30) and most cases, failed to find solution to large *n* queens problem. In this paper, we asserted that, even/odd numbered values of n-queens problem can affect the corresponding solutions of the standard backtracking. We also observed that, using a set of even and odd numbers, the odd number experience a turning point before the even numbers. The algorithm of the standard backtracking method was implemented in *C* programming language and, we used Microsoft Notepad as our output file to display the arrangement of the queens.**

## 1.0 Introduction

The *n*-queens problem is an old one that has been around for more than a century. Originally known as the 8-Queens problem, it has been studied by many famous mathematicians over the years, including the great German mathematician Karl Friedrich Gauss (1777-1855). The 8-Queens problem is a classic combinatorial problem whose purpose is to place eight queens on an 8 x 8 chessboard so that no two “attack”, i.e. so that no two of them are on the same row, column or diagonal. All solutions to the 8-queens problem can be represented as 8-tuples ($x_1$, …, $x_8$) where $x_i$ is the column on which queen *i* is placed. The explicit constraints using this formulation are $S_i$ = {1, 2, 3, 4, 5, 6, 7, 8}, $1 \le i \le$ n. Therefore the solution space consists of $8^8$ 8-tuples. Figure 1 shows a sample solution for 8-queens problem. The implicit constraints for this problem are that no two $x_i$’s can be the same (i.e. all queens must be on different columns) and no two queens can be on same diagonal. The problem was later generalized to *n* by *n* boards in 1850 by Franz Nauck. The goal of n-queens problem is to find a placement of n queens on an *n* x *n* chessboard, such that no one queen can be taken by any other. The solution space for n-queens consists of all n! permutations of the *n*-tuple (1, 2, …, *n*). While it has been well known that the solution to the n-queens problem is *n*, numerous solutions have been published since the original problem has been proposed [2]. Many of these solutions rely on providing a specific formula for playing queens or transposing smaller solutions sets to provide solutions for larger values of *n*.

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

**Figure 1: One Solution to the 8-queens problem**

One of the well-known methods for solving *n*-queens problem is called *backtracking* [3], [5], [6] [7], [8] and [11]. *Backtracking* is a method of trying to put the queens on the board squares one after the other. If one queen threatens the newly introduced queen, we withdraw the queen and search for another position. If we cannot find a solution, we choose to remove a queen already positioned, assign it another position that has not yet been used, and start the search again. This strategy is based on a trial-and-error algorithm. Backtracking ensures correctness by enumerating all possibilities. For backtracking to be efficient, we must prune the search space. Other methods of solving *n*-queens problem apart from backtracking are also available [10] and [13].

Although, backtracking method is an efficient searching method of combinatorial problems it is unable to solve large size of *n*-queens problem. In [4] and [11], the authors noted that the backtracking search method is not able to solve large size of *n*-queens problem. Recent results also indicate that we may not solve the *n*- queens problem where *n* is up to about 100. This is due to the exponential growth of the search load in backtracking [1], [4], [5], [13]. This work is motivated from personal observation of the results generated by the conventional backtracking problem. In this work, we present an alternative method for solving the n-queens problem. This approach is based on the standard backtracking method using the even/odd numbered values of n-queens.

## 2.0 Developing the algorithm for standard backtracking method

If we imagine the squares of the chessboard being numbered as the indices of the two dimensional array A(1:*n*, 1:*n*) then, we observed that for every element on the same diagonal which runs from the upper left to the lower right, each element has the same “row-column” value. Also, every element on the same diagonal which goes from the upper right left has the same “row + column” value. Suppose two queens are place at positions (*i, j*) and (*k, i*). Then, by the above, they are the same diagonal only if,

$i - j = k - l$ or $i + j = k + l$

This first equation implies

$j - l = i + k$

While the second implies

$j - l = k + i$

Therefore, two queens lies on the same diagonal if and only if $| j - l| = | i - k|$. Therefore the next queen to be placed on the chessboard should never be in the same diagonal, row or column of the preceding queens [1].

A careful study of the conventional backtracking method shows that even/odd numbered *n* has significant implications on the solution. This observation led us to the following rule:

- ➢ When the values of *n* queens is even numbers of 2, 4, …,30 and the values of odd queens are 1,3,…,29
- The values of odd numbered queens find faster solution to queens’ problem than the even numbered queens.
- When n is even/odd number, odd numbered *n*-queens experience a turning point in terms of faster execution time before the even numbered n queens do.

Implementing the above standard backtracking algorithm in Microsoft C program, we got the following computing time for different values of *n*. The result shows that, we have a faster solution to n-queens problem when the value of n is odd number. The results are presented in Table 1 below.

**Table 1: Solutions to *n*-queens problem using standard backtracking method.**

| **Even *n*** | **Time(seconds)** | **Odd *n*** | **Time(seconds)** |
|---|---|---|---|
| 2 | No Solution | 1 | No Solution |
| 4 | 0.0 seconds | 3 | No Solution |
| 6 | 0.0 seconds | 5 | 0.0 seconds |
| 8 | 0.00 seconds | 7 | 0.0 seconds |
| 10 | 0.00 seconds | 9 | 0.0 seconds |
| 12 | 0.01 seconds | 11 | 0.0 seconds |

| Even n | Time(seconds) | Odd n | Time(seconds) |
|---|---|---|---|
| 14 | 0.02 seconds | 13 | 0.0 seconds |
| 16 | 0.15 seconds | 15 | 0.02 seconds |
| 18 | 0.73 seconds | 17 | 0.08 seconds |
| 20 | 3.99 seconds | 19 | 0.05 seconds |
| 22 | 40.4 seconds | 21 | 0.23 seconds |
| 24 | 11.06 seconds | 23 | 0.64 seconds |
| 26 | 12.21 seconds | 25 | 1.37 seconds |
| 28 | 102.68 seconds | 27 | 14.48 seconds |
| 30 | 2160 seconds | 29 | 55.14 seconds |

## 3.0 Finding the turning point in the solution to *n* queens problem

Due to inability of most researchers to find efficient solution to large *n*-queens problem, the attention of most researches have been focused on the number of solutions of an arbitrary n queen problem. In our past researches, we religiously believe that, even/odd numbered value of *n* can affect the solution to most combinatorial problems and puzzles. After observation of the solutions to *n*-queens problem, especially the ones that generate the number of solutions to the problem, we found out two hidden facts. Using Microsoft C programming language to implement the conventional backtracking method, it is observed that there is a turning point for both even and odd numbers [9], [12], [17] and [18].
Figure 2 below, is a graph of even/odd number of n queens problem. The graph provides information on n-queens problem and the time in seconds. A cursory look at the graph shows that solutions to odd numbers of *n*-queens problem is faster than the solution to even number queens problem.

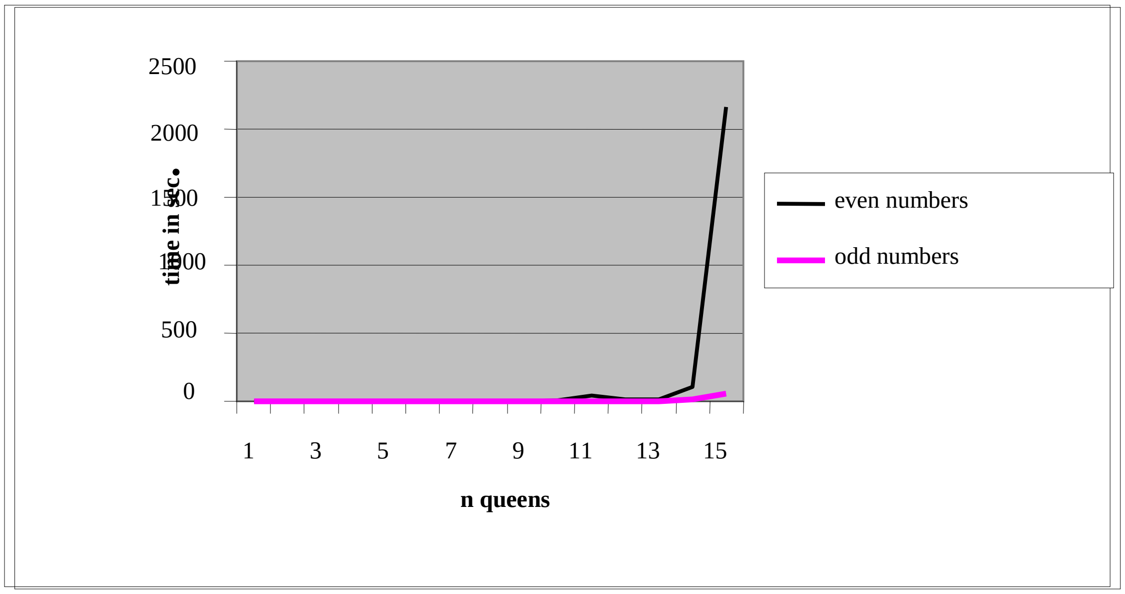

**Figure 2: The solutions of *n*-queens problem using even/odd n value (x-axis 1 cell represents 2 units and y-axis 1 row represents 500 seconds)**

Figure 3 depicts the graph of the solution of *n*-queens problem using a set of even numbers. The range of the even number is from 1 to 30 inclusive but there will be no solution to n when the value is 2. The information in the graph shows that the set of even number solutions experience a turning point when n is 24 and rise again when n is 26.

Figure 4 shows the graph of the solution of n-queens problem using a set of odd numbers. The range of the odd number is from 1 to 29 inclusive but there will be no solution to 1, and 3. The information in the graph shows that the set of odd number solutions experience a turning point when *n* is 19 and rise again when n is 21

In the final analysis, it was further observed that, the odd numbered n-queens solutions experienced the turning point before the even number. It is very clear that, the solutions to the n-queens problem can improve if n is an odd number. And, we can make an assertion based on the turning points that, because the odd numbered queens experiences a turning point before the even number that is why it is faster.

## 4.0 Implementation

To find a solution to *n*-queens problem and its performance in practice, the algorithm has been implemented using Microsoft Visual C++ programming language 6.0. The source codes is in C programming language, which is depicted in appendix A, and the results generated in tabular form are shown in appendix B. The specification of the computer system used for the implementation is: Intel Pentium II processor, 298 MHZ (speed of processor) and 54.0MB of RAM space.

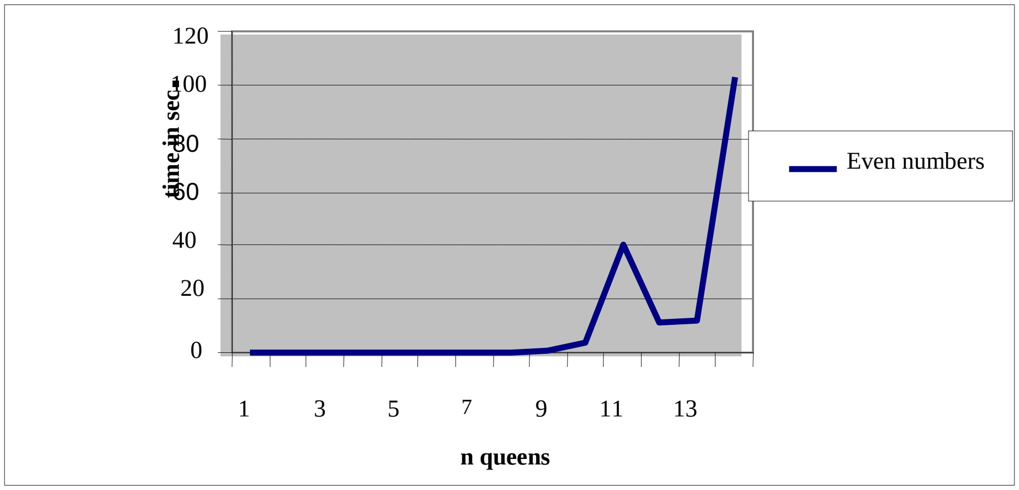

**Figure 3: The solution to n-queens problem when n is even (x-axis 1 cell represents 2 units and y-axis 1 row represents 20 seconds)**

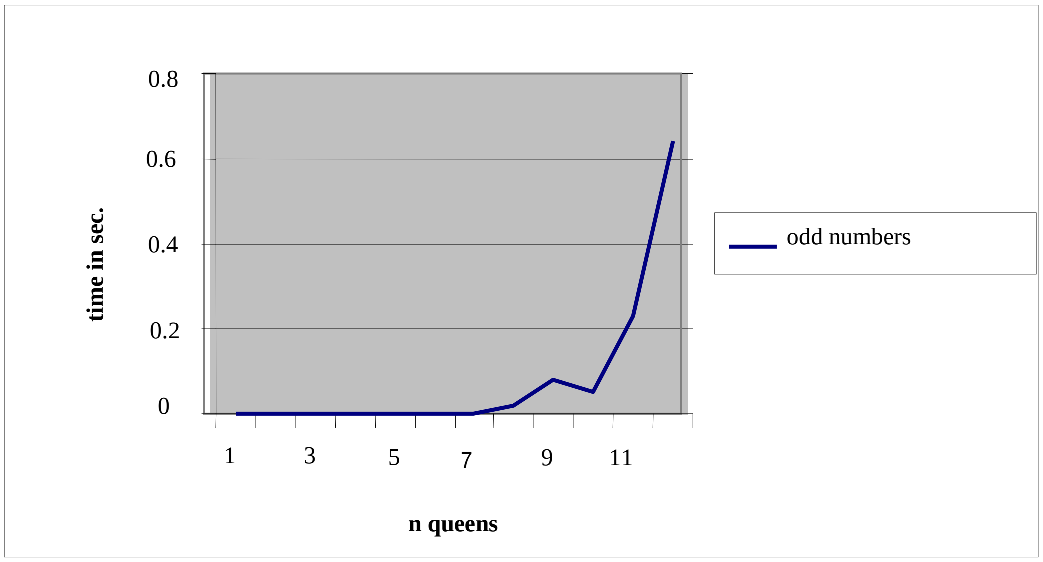

**Figure 4: The solution to n-queens problem when n is odd (x-axis 1 cell represents 2 units and y-axis 1 row represents 0.2 seconds)**

## 5. 0 Discussion of Results

A solution for a chessboard of size n = 17, 19, 22 and 24 obtained from the standard backtracking method of solving n queens problem as shown in Table 1 is displayed. This selected numbers are numbers before the turning points and when it turns in the even/odd numbers.

**Solution for *n* queens problem when *n* =17**

```
|Q| | | | | | | | | | | | | | | | | | |
| | |Q| | | | | | | | | | | | | | | | |
| | | | |Q| | | | | | | | | | | | | | |
| |Q| | | | | | | | | | | | | | | | | |
| | | |Q| | | | | | | | | | | | | | | |
| | | | | | | | |Q| | | | | | | | | | |
| | | | | | | | | | | | |Q| | | | | | |
| | | | | | | | | | | | | | |Q| | | | |
| | | | | | | | | | | | | | | | |Q| | |
| | | | | | | | | | | | | | | | | | |Q|
| | | | | | |Q| | | | | | | | | | | | |
| | | | | | | | | | | | | | | |Q| | | |
| | | | | | | | | | | | | | | | | |Q| |
| | | | | | | | | | |Q| | | | | | | | |
| | | | | |Q| | | | | | | | | | | | | |
| | | | | | | |Q| | | | | | | | | | | |
| | | | | | | | | |Q| | | | | | | | | |
| | | | | | | | | | | |Q| | | | | | | |
| | | | | | | | | | | | | |Q| | | | | |
```

**Solution for *n* queens problem when *n* =19**

```
|Q| | | | | | | | | | | | | | | | | | |
| | |Q| | | | | | | | | | | | | | | | |
| | | | |Q| | | | | | | | | | | | | | |
| |Q| | | | | | | | | | | | | | | | | |
| | | |Q| | | | | | | | | | | | | | | |
| | | | | | | | |Q| | | | | | | | | | |
| | | | | | | | | | | | |Q| | | | | | |
| | | | | | | | | | | | | | |Q| | | | |
| | | | | | | | | | | | | | | | |Q| | |
| | | | | | | | | | | | | | | | | | |Q|
| | | | | | |Q| | | | | | | | | | | | |
| | | | | | | | | | | | | | | |Q| | | |
| | | | | | | | | | | | | | | | | |Q| |
| | | | | | | | | | |Q| | | | | | | | |
| | | | | |Q| | | | | | | | | | | | | |
| | | | | | | |Q| | | | | | | | | | | |
| | | | | | | | | |Q| | | | | | | | | |
| | | | | | | | | | | |Q| | | | | | | |
```

**Problem when *n* = 22**

```
|Q| | | | | | | | | | | | | | | | | | | | | |
| | |Q| | | | | | | | | | | | | | | | | | | |
| | | | |Q| | | | | | | | | | | | | | | | | |
| |Q| | | | | | | | | | | | | | | | | | | | |
| | | |Q| | | | | | | | | | | | | | | | | | |
| | | | | | | | | |Q| | | | | | | | | | | | |
| | | | | | | | | | | | | |Q| | | | | | | | |
| | | | | | | | | | | | | | | | |Q| | | | | |
| | | | | | | | | | | | | | | | | | | |Q| | |
| | | | | | | | | | | | |Q| | | | | | | | | |
| | | | | | | | | | | | | | | | | | |Q| | | |
| | | | | | | | | | | | | | | | | | | | | |Q|
| | | | | | | | | | | | | | | | | |Q| | | | |
| | | | | | | |Q| | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | |Q| |
| | | | | | | | | | | |Q| | | | | | | | | | |
| | | | | | | | |Q| | | | | | | | | | | | | |
| | | | | |Q| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | |Q| | | | | | |
| | | | | | |Q| | | | | | | | | | | | | | | |
| | | | | | | | | | |Q| | | | | | | | | | | |
| | | | | | | | | | |Q| | | | | | | |
```

**Solution for *n* queens problem when *n* = 24**

```
|Q| | | | | | | | | | | | | | | | | | | | | | | |
| | |Q| | | | | | | | | | | | | | | | | | | | | |
| | | | |Q| | | | | | | | | | | | | | | | | | | |
| |Q| | | | | | | | | | | | | | | | | | | | | | |
| | | |Q| | | | | | | | | | | | | | | | | | | | |
| | | | | | | | |Q| | | | | | | | | | | | | | | |
| | | | | | | | | | |Q| | | | | | | | | | | | | |
| | | | | | | | | | | | | |Q| | | | | | | | | | |
| | | | | | | | | | | | | | | | | |Q| | | | | | |
| | | | | | | | | | | | | | | | | | | | | |Q| | |
| | | | | | | | | | | | | | | | | | |Q| | | | | |
| | | | | | | | | | | | | | | | | | | | | | |Q| |
| | | | | | | | | | | | | | | | | | | |Q| | | | |
| | | | | | | | | | | | | | | | | | | | | | | |Q|
| | | | | | | | | |Q| | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | |Q| | | |
| | | | | |Q| | | | | | | | | | | | | | | | | | |
| | | | | | | |Q| | | | | | | | | | | | | | | | |
| | | | | | | | | | | |Q| | | | | | | | | | | | |
| | | | | | | | | | | | | | | |Q| | | | | | | | |
| | | | | | | | | | | | |Q| | | | | | | | | | | |
```

A cursory look at both solutions, we found that execution time of standard method for odd numbers 17 and 19 are 0.08 and 0.05 seconds respectively, while the even numbers 22 and 24 are 40.4 and 11.06 seconds respectively. The results also show that: (i) the probability of placing a queen (Q) in any row or column of the chessboard is 1/n and (ii) the probability of Q being in any cell of the chessboard is $1/n^2$. A look at the output for both odd/even placements of the queens shows that the queens are concentrated on the left and right along the diagonals of the chessboard.

## 6.0 Conclusion

In this paper, we have studied the n-queens problem by implementing the conventional backtracking problem. We particularly studied Horowitz and Sahni [1] backtracking algorithm, and implemented this algorithm, using separate values of even and odd numbers. Doing this, we found out that, the odd numbers n find solution quicker than the even number n. A computer program in Microsoft C was used to implement the algorithm. We presented the turning points of the solutions of n-queens problem using even/odd numbered n in order to improve on the conventional. The presentation for n = 17, 19, 22 and 24 were used to display the turning points of n-queens problem using odd/even numbered n. Further work should be carried out to see how the odd/even numbered n can affect the arrangements of the queens on the chessboard. It will however not be right to assume or try to imply in this paper that the best has been done in reducing the speed of execution of solution to *n*-queens problem.

## APPENDIX A

**C program, which computes the Conventional backtracking method of n Queens**

```
#include <stdio.h>
#include<math.h>
#include<time.h>
#define TRUE 1
#define FALSE 0
int b4 = FALSE;
int place(int k);
int col[5100];
clock_t t0, t1;
int t=1, count, c, s;
FILE *fp;

int main()
{
int b, n, r;
int row[5100];
printf("  Enter the value of N-Queens: ");
scanf("%d", &n);

if(n < 4)
{
printf("\n");
printf("  Impossible Solution! \n");
printf("  The execution of this program has been terminated. \n");
printf("  To run this program again, enter an integer");
printf(" value greater than 3: ");
return 0;
}

t0=clock();
printf("  Please Wait...\n");
fp=fopen("Queen.txt", "w");
count =1;
r=1; col[r]=0;

while(r>0)
 {
  col[r]=col[r]+1;
  while((!place(r))&&(col[r]<=n))col[r]= col[r] + 1;
  while((place(r))&&(col[r]<=n))
  {
```

```
                         row[r]=col[r];
                         r= r + 1;
                         col[r]=1;   }
                b4= FALSE;

                if(col[r]>n)
               {
                          r= r - 1; row[r]=col[r]+1;
                          b= row[r];
                          if(b==n)b4= TRUE;
                }

               if(b4)
               {
                          r= r - 1; row[r]=col[r]+1;
               }

               if(r==(n+1))r=-1;
 }

  if((n==4)||(n==6))row[1]=2;
 fprintf(fp,"\n\n The solution when n = %d\n",n);
 fprintf(fp,"\n------------------------------\n");
 r=1;
 c=1;
 fprintf(fp,"\n");

 for(r=1;r<=n;r= r + 1)
   {
                fprintf(fp,"\n|");
                for(c=1;c<=n;c= c + 1)
                         if(row[r]==c) fprintf(fp,"Q|");
                                  else
        fprintf(fp," |");
               }
              t1=clock();
              printf("Time: %f sec\n",(t1-t0)/(float)CLOCKS_PER_SEC);
              fclose(fp);
  printf("The Queens configuration is displayed in Queen.txt\n");
              return 0;
}
int place(int k)
{
 int i=1;
 while(i<k)
              {
               if(((col[i]==col[k]))||
                                  (abs(col[i]-col[k]))==(abs(i-k)))
                           return FALSE;
                         i= i + 1;
              }
              return TRUE;
```

## References

[1] Horowitz E. and Sahni .S (1978); *fundamentals of computer Algorithms, computer,* science press, Inc, Maryland, USA, Pp. 1- 33, 322 - 340.

[2] Hoffman, E. J., Loessi, J. C. and Moore, R. C. (1969). *Construction for the solutions of the n-quuens problem.* Mathematics Magazine, pp. 66-72.

[3] *Eight Queens Problem*: "http://www.recipeland.com/encyclopaedia/index.php/Eight_queens_puzzle"

[4]] Harold S. Stone and Janice M. Stone (1987). *Efficient search techniques - an empirical study of the n-queens problem.* IBM Journal Research Development, 31:464--474.

[5] Steve Skiena (May, 2003); *Backtracking*, http://wwwcs.sunysb.edu/~skiena/392/lectures/index.html.
[6] *The n-Queens Problem*; http://www2.ilog.com/preview/Discovry/samples/index.html
[7] Somers .J (1993); *The N queens Problem: a study in optimization*,http://www.jsomers.com/nqueen _demo/nqueens.html
[8] Ruskey .F. and Ruskey .S. (June, 1996); Information on the n Queens Problem, http://theory.cs.uvic.ca/~cos/amof/e-queenI.htm.
[9] Biter J. R. and Reingold (1975); Backtrack Programming techniques, Communications of the ACM, Vol. 18, pp. 651-656.
[10] Latavac .C. and Ruggiero .J. (2002); The n-Queens Problem, INFORMS Transactions on Education, Vol. 2, No.3, http://pubs.informs.org/Vol2No3/LetavacRuggiero/
[11] Keld H. Cback: A Simple Tool For Backtrack Programming in C, http://www.akira.ruc.dk/~keld/research/CBACK/CBack-1.0/DOC/CBACK_PAPER.pdf
[12] Allison .L. (1999); N Queens Problem (Number of Solutions), http://www.csse.monash.edu.au/~lLoyd/
[13] Antipolis .I. S. (2005); N-queens World Record, University of Nice Sophia Antipolis-CNRS, http://www.i3s.unice.fr/
[14]] Bratko I. (1990); Prolog Programming For Artificial Intelligence, Addison-Wesley, Second Edition.
[15] Kernighan B. W. and Ritchie D. M.(1988); The C Programming Language, Second Edition, Prentice Hall.
[16] Dr. Dobby's Journal (May 2004); The Optimal Queens, Vol. 30, No5, http://penguin.ewu.edu/~trolfe/Queens/optQueen.doc
[17] Ikpotokin, F.O., Chiemeke, S.C., and Osaghae, E. O. (2004); Alternative Approach to the Optimal Solution of Tower of Hanoi, Journal of Institute of Mathematics and Computer Sciences, India. vol. 15, No.2, 229-244.
[18] Ikpotokin, F.O., Chiemeke, S.C., and Osaghae, E. O. (2004): Generalized Recursive Optimal Solution for the Multi-Peg Towers of Hanoi, Journal of Institute of mathematics and computer Sciences, India. vol. 15, No.2, 297-309.
[19] J. A. Campbell (1985): Implementation of Prolog. Halsted Press.