DIFFERENT PERSPECTIVES OF THE N-QUEENS PROBLEM

Cengiz Erbas*, Seyed Sarkeshik†, Murat M. Tanik**

Department of Computer Science and Engineering

Southern Methodist University, Dallas, TX 75275-0122

Abstract

The N-Queens problem is a commonly used example in computer science. There are numerous approaches proposed to solve the problem. We introduce several definitions of the problem, and review some of the algorithms. We classify the algorithms for the N-Queens problem into 3 categories. The first category comprises the algorithms generating all the solutions for a given N. The algorithms in the second category are designed to generate only the fundamental solutions [34]. The algorithms in the last category generate only one or several solutions but not necessarily all of them.

0. INTRODUCTION

The N-queens problem, which was germinated from the 8-Queens problem, has been studied for more than a century [14,15,17]. The problem introduced by a chess player Max Bazzel in 1848, and 2 years later, in 1850, it was suggested in Illustrirte Zeitung [15,17]. Ever since 1850, the problem attracted the attention of several famous mathematicians including Gauss [14,15], Polya [22], and Lucas [19]. During the last three decades, the problem is discussed in the context of computer science and used as an example of backtracking algorithms [5,16,30], permutation generation [5,25,26,27], divide and conquer paradigm [1], program development methodology [21,35], constraint satisfaction problems [20,24], integer programming [13], specification [28], and neural networks [31]. Some practical applications of the N-Queens problem such as parallel memory storage scheme's [9], VLSI testing, traffic control [29], and deadlock prevention [32], are also mentioned in the literature.

In this paper, we introduce several definitions of the problem, and review some of the algorithms related to the N-Queens problem. We classify the algorithms for the N-Queens problem into 3 categories. The first category comprises the algorithms generating all the solutions for a given N. The algorithms in the second category are designed to generate only the fundamental solutions. The algorithms in the last category generate only one or several solutions but not necessarily all of them.

1. DEFINITIONS OF THE N-QUEENS PROBLEM

The N-Queens problem can be defined as follows: Place N queens on a N by N chessboard, one queen on each square, so that no queen captures any other, that is, the board configuration in which there exists at most one queen on the same row, column and diagonals. Figure 1 illustrates a solution to the 8-Queens problem.

Solutions to the N-Queens problem can be represented by a permutation. For example, the solution given in Figure 1 can be represented by the following permutation.

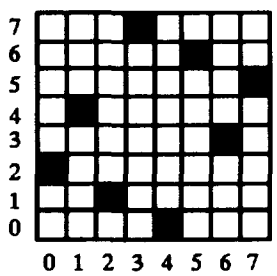

Figure 1. A Solution to the 8-Queens problem.

$$\begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 2 & 4 & 1 & 7 & 0 & 6 & 3 & 5 \end{bmatrix}$$

There are several different definitions for the N-Queens problem. Each of the following definitions, provides a different perspective of the same problem.

*Graduate Student in CSE. **Associate Professor of CSE
† Graduate Student in University of Texas at Arlington.

1.1 Algebraic Definition

Let A be the set of integers such that A = {0,1,2,..., N-1}. Define B as the Cartesian product of A, i.e., B = A*A. Then, B = {(0,0),(0,1),..., (1,0),..., (N-1,0),..., (N-1,N-1)}. Select N elements of B and call this set as C. The set C represents a solution to the problem if for the Cartesian product of C, C*C, each element $((i_k, j_k), (i_l, j_l))$ either satisfies all of the following conditions or fails all of them (for all l, k = 0, 1,..., N-1):

1. $i_k \neq i_l$ {not on the same column},
2. $j_k \neq j_l$ {not on the same row},
3. $i_k + j_k \neq i_l + j_l$ {not on the same diagonal},
4. $i_k - j_k \neq i_l - j_l$ {not on the same diagonal}.

1.2 Modular Arithmetic Definition

Let A = $\{0,1,2,...,N^2-1\}$. Select N elements from A and call this subset B. The set B corresponds to a solution if each element (X,Y) ∈ B*B (where X ≠ Y) satisfy the following conditions (where | | denotes the absolute value, and / denotes integer division) [5]:

1. (X mod N) ≠ (Y mod N) {not on the same column},
2. X/N ≠ Y/N {not on the same row},
3. X/N + (X mod N) ≠ Y/N + (Y mod N) {not on the same diagonal},
4. | X/N - (X mod N) | ≠ | Y/N - (Y mod N) | {not on the same diagonal}.

1.3 Constraint Satisfaction Problem

The N-Queens problem can be rephrased as a constraint satisfaction problem [20,24]. The general constraint satisfaction problem consists of a tuple $\{N, R_0, R_1, ..., R_{N-1}, P_{01}, P_{02}, ..., P_{N-2,N-1}\}$ such that: a) N is an integer denoting the number of variables $(x_0, x_1, ..., x_{N-1})$, b) R_i is a set of possible values that can be assigned to a variable such as x_i, c) P_{ij} is a subset of $R_i * R_j$ and represents the constraints. For N-Queens problem, $R_0=R_1=...=R_{N-1}=R$, and $P_{ij} = \{ R_i * R_j - P'_{ij} \}$ where $P'_{ij} = \{ (x,y) \in R_i * R_j \ni |x-y| = |i-j| \cup x+y = i+j \cup x = y \}$. It is obvious that $P_{ij} = P_{ji}$. A solution to the problem is an assignment A = $(a_0, ..., a_{N-1})$, where a) for all i, a_i is an element of R. b) $(a_i, a_j) \in P_{ij}$ for all $i \neq j$ where $0 \leq i \leq j \leq N-1$.

Example: Let N = 4. Then, $R_0=R_1=R_2=R_3=\{0,1,2,3\}$ and,
P_{01} = {(0,2), (0,3), (1,3), (2,0), (3,0), (3,1)},
P_{02} = {(0,1), (0,3), (1,0), (1,2), (2,1), (2,3), (3,0), (3,2)},
P_{03} = {(0,1), (0,2), (1,0), (1,2), (1,3), (2,0), (2,1), (2,3), (3,1), (3,2)},
P_{12} = {(0,2) (0,3), (1,3), (2,0), (3,0), (3,1)},
P_{13} = {(0,1), (0,3), (1,0), (1,2), (2,1), (2,3), (3,0), (3,2)},
P_{23} = {(0,2) (0,3), (1,3), (2,0), (3,0), (3,1)}.

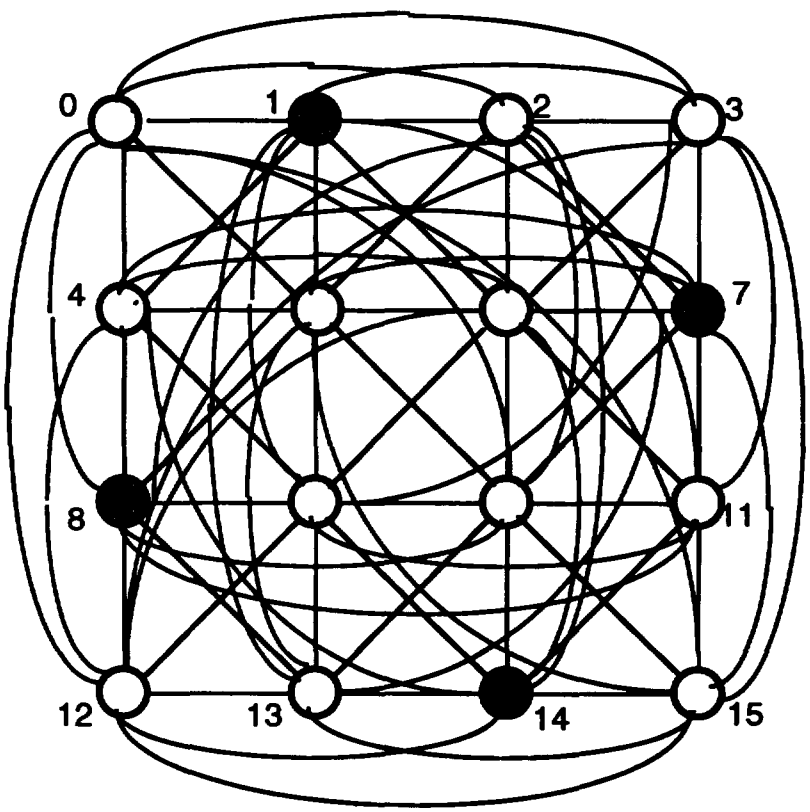

Figure 2. Graph Representation of the N-Queens Problem as a Maximum Stable Set Problem

A solution set is any set such as A = {1, 3, 0, 2} that satisfies all of the conditions. Several other representations of the N-Queens problem as a constraint satisfaction problem is given in [20].

1.4 Maximum Stable Set Problem

N-Queens problem can also be viewed as a maximum stable (independent) set problem. Maximum independent set problem is defined as follows: Given a graph G = (V,E), where V denotes a set of vertices and E denotes the edges between them, a set $V' \subseteq V$ is defined to be **stable** if no two vertices in V' are joined by an edge in E. The **stability number** of G is defined as the maximum cardinality of a stable set [4]. In order to map the N-Queens problem to the maximum stable set problem, view the board as a graph in which each square is represented by a vertex and any two vertices (squares) on the same row, column, or diagonal are connected by an edge as in Figure 2. The stability number of this graph is N (for $N \geq 4$). Each of the maximum stable set of this graph is also a solution to the N-Queens problem.

1.5 Maximum Clique Problem

Once the problem is defined as a stable set problem, it can also be defined as a clique problem [13]. Clique problem is defined as the following: Given a graph G = (V, E), where V denotes a set of vertices and E denotes the edges between them, a set $V' \subseteq V$ is defined as a **clique** if every vertex in V' is joined by an edge in E. Any clique with the largest possible cardinality is called **Maximum Clique**. To map the N-Queens problem to the maximum clique problem, the board is viewed as a graph. But this time, an edge is defined between two vertices that are not on the same column, row, or diagonal. Any clique of size N is a solution to the N-Queens problem. Graph representation of the 4-Queens problem for this definition is given in Figure 3.

1.6 Integer Programming

N-Queens problem can be formulated as an integer programming problem. Let x_{ij} represent the squares on the board (where i, j = 0,1,..., N-1), then, the N-Queens problem can be mapped to this definition according to the following model [13]:

Maximize: $\sum_{i} \sum_{j} x_{ij}$ i, j = 0, 1, .. , N-1

Such that:

1. $\sum_{j=1} x_{ij} = 1$ i = 0, 1, 2,..., N-1

2. $\sum_{i=1} x_{ij} = 1$ j = 0, 1, 2,..., N-1

3. $\sum_{i+j=k} x_{ij} \leq 1$ k = 1, 2, 2N-3

4. $\sum_{i-j=k} x_{ij} \leq 1$ k = -N+2, -N+3, ..., N-2

$x_{ij} = 0$ or 1 for all i,j

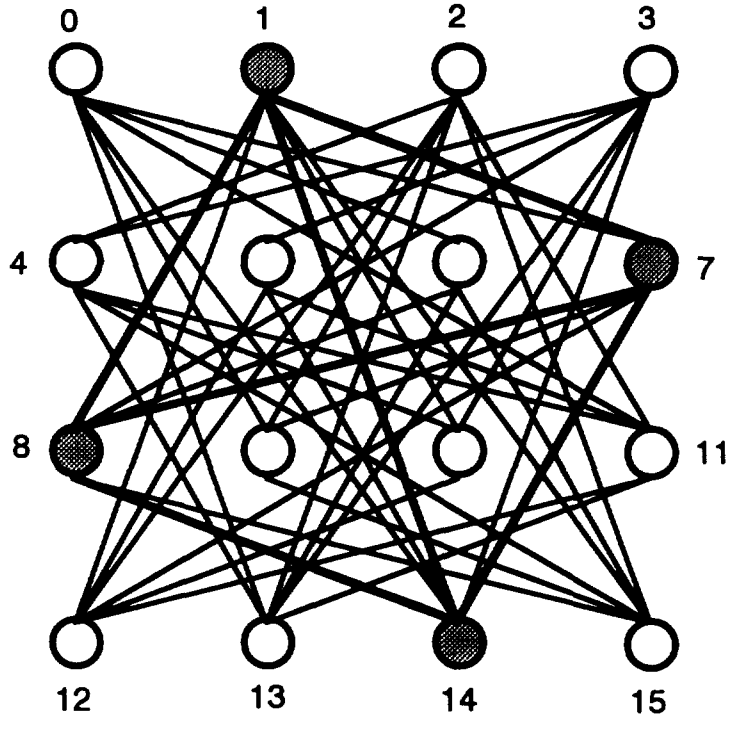

Figure 3. Graph Representation of the 4-Queens Problem as a Maximum Clique Problem

Constraint sets 1 and 2 do not allow more than one queen on each row or column while constraint sets 3 and 4 prevent collisions on the diagonals. Solving this problem will produce a solution to the N-Queens problem. In this case, the N variables (actually, the squares) that have the value of 1 in the optimal solution of the integer programming model will correspond to a solution.

2. ISOMORPHIC AND FUNDAMENTAL SOLUTIONS

The solutions to the N-Queens problem form a group. Having an initial solution I, some other solution can be obtained by simple rotations. The following 8 rotations can be applied to the solutions of the N-Queens problem [7].

R1 : A rotation of 90 degrees in the counterclockwise direction.

R2 : A rotation of 180 degrees in the counterclockwise direction.

R3 : A rotation of 270 degrees in the counterclockwise direction.

R4 : A rotation of 360 degrees in the counterclockwise direction. This is obviously the same as I.

D1 : A 3-dimensional rotation of 180 degrees about the vertical axis of the board.

D2 : A 3-dimensional rotation of 180 degrees about the horizontal axis of the board.

D3 : A 3-dimensional rotation of 180 degrees about the diagonal passing through the cells a_{ij}, $i \neq j$.

D4 : A rotation of 180 degrees in 3-dimensions about the principal diagonal axis, (i = j).

These rotations allow us to generate different views of the same solution. Two solutions are said to be equivalent (**isomorphic**) if either can be transformed into one another by one of the rotations mentioned . These rotations partition the solutions into a set of equivalence classes. A set of solutions consisting of exactly one representative from each equivalence class is called a set of **fundamental** solutions. Sets of fundamental solutions are of interest because they concisely describe the set of all solutions.

We would classify the algorithms related to the N-Queens problem into 3 categories. The first category comprises the algorithms that find all the solutions for a given N. The algorithms in the second category are designed to find only the fundamental solutions. Algorithms in the last category find only one or several solutions, but not necessarily all of them.

3. ALGORITHMS GENERATING ALL SOLUTIONS

The objective of these algorithms is to find all the solutions for a specified N. Determining the exact number of solutions is a difficult problem , and so far no closed-form formula has been found for the general case [36]. Table 1 gives the number of solutions vs. the number of queens, based on the empirical results.

Table 1. The Number of Solutions for the N-Queens Problem

N	# of solutions	N	# of solutions
1	1	10	724
2	0	11	2,680
3	0	12	14,200
4	2	13	73,732
5	10	14	365,596
6	4	15	2,279,184
7	40	16	14,772,512
8	92	17	95,815,104
9	352	18	666,090,624

Empirical observations indicate that the number of solutions increases exponentially. In this section, we analyze the following algorithms:

1. Trial and Error Algorithms.
2. Backtracking Algorithms.
3. Permutation Generation Algorithms.

3.1 Brute-Force Trial and Error Algorithms

Consider a Cartesian product space $X_0 * X_1 * ... * X_{N-1}$, of N selection spaces $X_0, X_1, ..., X_{N-1}$ each of which contains a finite number of distinguishable values. The problem is to find a vector $(x_0, x_1, ...,x_{N-1})$ that satisfies the criterion function $f(x_0, x_1, ...,x_{N-1})$ where $x_i \in X_i$ for i = 0,1,..,N-1. Let M_i be the number of distinguishable values in X_i [16]. Then, number of different vectors in the Cartesian product space is $M = \prod M_i$. The brute-force trial and error algorithms generate all of the M possible vectors and test each of them according to the criterion function *f*. For the N-Queens problem, we have to place each queen on a separate row. Thus, the selection spaces and value of M_i are defined as follows:

$$X_0 = X_1 = ... = X_{N-1} = (0, 1, 2, ..., N-1)$$
$$M_0 = M_1 = ... = M_{N-1} = N$$

The criterion function is defined as: For each pair (x_i, x_j) of elements of vector $(x_0, x_1, ..., x_{N-1})$, the following conditions must be satisfied:

1. $x_i \neq x_j$
2. $i + x_i \neq j + x_j$
3. $i - x_i \neq j - x_j$

Brute-force trial and error algorithms are among the most inefficient algorithms because the number of different vectors to be tested is calculated as $M = \prod M_i = N^N$.

3.2 Backtracking Algorithms

The brute-force trial and error algorithms never test the criterion function until all of the elements of the solution vector (x_i's) have been selected. Instead, backtracking algorithms perceive the framework as a multi-stage decision problem. The basic idea of backtracking algorithms is to build up the solution vector one component at a time and test it according to the criterion function to determine whether the vector being formed still has a chance of

success. If a partial vector such as $(x_0,x_1,-,-,...,-)$ cannot be a partial solution, all possible (N^{N-2}) configurations starting with these two elements can be discarded [16].

N-Queens problem is among the most commonly used examples of backtracking algorithms [5,16,30]. To explore all possible vectors, we start by placing a queen on the first column of the first row and continue placing the queens on the other rows, while maintaining the constraints that are imposed by the previously placed queens. If we reach a row for which all squares are already attacked by the other queens, we backtrack to the previous row and explore other vectors. This algorithm can be considered as a tree traversal process.

Recursive algorithms are another way of implementing backtracking algorithms. Wirth uses this approach in the following algorithm to solve the N-Queens problem recursively [35].

Algorithm 1. Recursive Algorithm (Wirth's Algorithm)

```
procedure Trycolumn(j);
    var X[0..N-1], current_row;
begin
    current_row := 0;
    repeat
        if the configuration is safe then
            X[j] := i {Set Queen}
            If not lastcoldone then Trycolumn(j + 1)
            else Print(X);
            removequeen;
        current_row := current_row+ 1;
    until lastsquare
end
```

3.3 Permutation Generation Algorithms

These algorithms are based on generating all of the permutations of N different elements. The brute-force trial and error algorithm allows a configuration with more than one queen on each column before testing the solution. In order to abide this restriction, the board is represented by a vector of N different numbers. All possible permutations of these N numbers are generated and tested to see whether they are solutions to the problem or not. The complexity of this approach is O(N!). One problem with these algorithms is that they do not test the partial arrangements. However, we may apply the criterion function to the partial vectors. If the partial vector can not be a partial solution, any arrangement starting with the same partial vector can be eliminated. Rohl's Algorithm I use this approach [25,26].

Algorithm 2. Rohl's Algorithm I

```
procedure choose(k)
    var temp, i, pk;
begin
    temp := p[k];
    for i := k to N do
        p[k] := p[i];
        if the partial arrangement p[1..k] satisfies
            the conditions then
            p[k] := pk;
            p[i] := temp;
            if k <> N then choose(k+1)
            else print_solution(p);
            p[i] := p[k];
    p[k] := temp;
end
```

Rohl describes another algorithm which is very similar to his first algorithm [27]. This algorithm generates solutions to the N-Queens problem in a lexiographical order.

Algorithm 3. Rohl's Algorithm II

```
procedure searchrow;
    var l, m, h, F[-1..N-1]
begin
    l := -1; h := F[-1];
    repeat
        if the square (m,h) is safe then
            place a queen to (m,h)
            F[l] := F[h];
            m := m + 1;
            if m = N then print_solution
            else searchrow;
            m := m - 1;
            remove the queen from (m,h)
            F[l] := h;
        l := h;
        h := F[h];
    until h < 0;
end
```

In this algorithm global array F is defined as an array [-1..N-1] of integer, and is initialized as follows: for k := 0 to N-1 do F[k-1] := k-1; F[N-1] := -1;

3.4 Comparison of the Algorithms Generating All Solutions

As discussed, brute-force, generating permutation, or other similar algorithms are very inefficient/ The following table shows the comparison of the run times for some of the more efficient algorithms that generate all solutions:

Table 2. Run Time Statistics For the Algorithms Generating All Solutions

	Algorithm*		
N	Wirth	Rohl I	Rohl II
8	0: 0.16**	0: 0.10	0: 0.10
9	0: 0.60	0: 0.43	0: 0.38
10	0: 2.69	0: 1.97	0: 1.53
11	0: 13.51	0: 9.72	0: 7.63
12	1: 13.54	0: 50.97	0: 39.75

* Run on CompuAdd 386SX, 20MHz using TURBO PASCAL 5.5

** All times are min: sec.hundredths of sec obtained using the GETTIME function call in DOS operating system.

As can be seen in Table 2, the most efficient algorithm generating all the solutions to the N-Queens problem is the Rohl's Algorithm II.

4. ALGORITHMS GENERATING FUNDAMENTAL SOLUTIONS

In this section, we discuss some of the algorithms that find the fundamental solutions. The number of fundamental solutions vs. the number of queens are given in Table 3.

Table 3. The number of Fundamental Solutions for the N-Queens Problem

N	# of solutions	N	# of solutions
1	1	9	46
2	0	10	92
3	0	11	341
4	1	12	1,787
5	2	13	9,233
6	1	14	45,752
7	6	15	285,053
8	12	16	1,846,955

As can be observed, the number of fundamental solutions increases exponentially. The following algorithms generate the fundamental solutions:

1. Generate and Test Algorithms
2. Partial Elimination of Symmetries (Naur's Algorithm)
3. An Approach with Group Properties (Topor's Algorithm)

4.1 Generate and Test Algorithms

Generate and test algorithms, use one of the algorithms generating all solutions, as the "generating" part of the algorithm. When a solution is generated, it is first tested to see whether it is fundamental or not. If it can be transformed to a previously found fundamental solution, it is discarded. Otherwise, it is added to the fundamental solutions list.

4.2 Partial Elimination of Symmetries (Naur's Algorithm)

Naur's Algorithm uses the central rows to eliminate some of the symmetries while for the other rows it still uses "generate and test" methods to discard the isomorphic solutions [21]. This algorithm has originally been written for 8-Queens. Although it is possible to generalize it for N-Queens, it is not significant to do so because the number of symmetries eliminated by the algorithm relatively decreases as the number of queens increases. For 8-Queens, this algorithm uses 4th and 5th rows to eliminate some of the symmetries. Figure 4 depicts the symmetries in these rows. A queen on column 0 of the 3th row has 7 symmetric squares. These symmetric squares are symbolized with "A"s in the figure. Similarly, "B"s, "C"s, and "D"s represents other symmetries on the board.

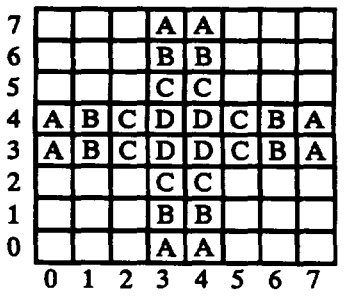

Figure 4. Symmetric Squares of Central Columns and Rows

In order to avoid generating isomorphic solutions, if a queen is tried on each of the rows 0,1, 2, and 3 of column 3, it could not be started in any other of the fields marked "A", "B", "C", "D", except when two or more queens can be placed in the positions having the same letter .

4.3 An Approach with Group Properties (Topor's Algorithm)

This algorithm uses a direct method for generating fundamental solutions [34]. We know that the solutions, which can be transformed to each other by rotations, form a group. Topor's algorithm utilizes the group property of the N-Queens problem to generate the fundamental solutions. The algorithm uses the concept of orbits. An orbit of a square under a group is defined as the set of squares to which the given square can be mapped by elements of the group. Given the symmetry group G, all the squares in the orbit of the candidate square under G are equivalent. Thus, placing a queen on any square in the orbit leads to an equivalent solution.

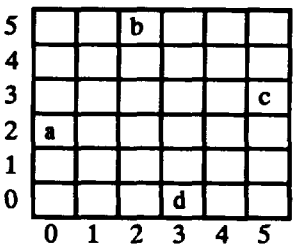

Figure 5. Distinct Selections of Topor's Algorithm

Topor's algorithm can be summarized as the following recursive algorithm:

Algorithm 4. Topor's Algorithm

```
procedure choose(c, S, G);
   var sq, seln, current_column;
begin
   current_column := c;
   while current_column already contains a queen do
      current_column := current_column + 1;
   for each safe square, sq, in current_column do
         for each square in seln do
```

```
        place queen on this square;
    if N queens have been placed then print solution
    else choose (current_column+1, S,
                 symmetry group of new solution);
    for each square in seln do
        remove queen from this square
end
```

5. ALGORITHMS GENERATING SOME SOLUTIONS

The algorithms in this category aim to find only one or several solutions to the problem. The following algorithms are in this category:

1. Nondeterministic Algorithms,
2. Backtracking Algorithms,
3. Probabilistic Algorithms,
4. Divide-and-Conquer Algorithm.

5.1 Nondeterministic Algorithms

Nondeterministic algorithms differ from their deterministic counterparts because of a special primitive called "*choice*". This primitive is used to make a selection among a number of choices. The deterministic counterparts of the nondeterministic algorithms are the backtracking algorithms. It is possible to translate a nondeterministic algorithm to a backtracking algorithm [6,12]. The specification of the nondeterministic algorithm can be stated as: Pick one square on each column, being careful not to pick two on the same row or diagonal. Write the number of the row you chose, and advance to the next column.

Algorithm 5. Nondeterministic Algorithm

```
procedure nondeterministic(N)
    var column, current-row;
begin
    current-row := 0;
    while current-row < N-1 do
        column := choice(N);
        place a queen to the column of the current-row;
        if this configuration is safe
          then current-row := current-row + 1;
        else failure;
    success;
end
```

5.2 Backtracking Algorithms

The general characteristics of backtracking algorithms have been discussed earlier. These algorithms can be modified to provide only one solution. These algorithms find the lexiographically first solution to the N-Queens problem. Empirical results show that the run-times for the backtracking algorithms which find only the first solution, increase exponentially [30].

5.3 Probabilistic Algorithms

The backtracking algorithm provides the solutions by placing the queens on the board systematically. However, it is known that the positions of the queens in most of the solutions do not follow a general pattern. This observation suggest a Las Vegas algorithm to place the queens randomly on the successive rows, providing that those queens placed on the board satisfy the problem conditions. The algorithm ends successfully if it manages to place all the queens on the board, otherwise; it fails. We may obtain different solutions by applying this probabilistic algorithm [5].

Algorithm 6. Las Vegas Algorithm

```
procedure LasVegas (N)
    var current_row
begin
    current_row := -1;
    while there are available squares on the next row
        (i.e., current_row + 1) or current_row < N-1 do
        choose one of the available squares randomly;
        place a queen to that square;
        current_row := current_row + 1;
    if current_row = N-1 then this is a solution
    else failure to reach a solution;
end
```

When the algorithm detects a failure, it starts from scratch again. We can improve the efficiency of this algorithm by combining it with the backtracking algorithm. This new algorithm places a number of queens on the board randomly. Then, it uses backtracking to add the remaining queens to the initial configuration.

Another probabilistic algorithm utilizes gradient-based heuristic search [29]. This algorithm is able to generate a solution for extremely large values of N. In this algorithm, a random permutation generator is used to place the queens on the board. A permutation guarantees that no two queens are either on the same row or on the same column. This generally produces collisions along the diagonals. The gradient-based heuristic is applied to all possible pairs of queens until there is no collisions along the diagonals. The main idea is to reduce the number of collisions by swapping a pair of queens. The swapping is actually performed if the swapping of two queens lessens the number of collisions. If this procedure does not reach a solution, a new permutation is generated and a new search is applied. The empirical results show that the number of permutations necessary to generate a solution is usually very small. Algorithm 7 is the gradient-based heuristic part of this algorithm [29].

Algorithm 7. Gradient-based Heuristic Search Algorithm

```
procedure
  var swaps_performed;
begin
   repeat
     swaps_performed := 0;
     for i := 0 to N-1 do
         for j := (i+1) to N-1 do
```

```
        if queen i or queen j is attacked then
            if swap(queen i, queen j) reduces collisions
             then
                perform_swap(queen i, queen j);
                swaps_performed := swaps_performed+1;
            end;
  until swaps_performed = 0;
end;
```

5.4 Divide-and-Conquer Algorithms

A similar problem to the N-Queens problem is known as N-Superqueens problem. This problem has been solved by Polya [22]. In this problem the queens are placed on a toroidal board instead of a planar board. The algorithm introduced in this section uses the connection between the N-Queens and the N-Superqueens problems.

A decomposition solution method to the N-Superqueens problem is introduced in [1]. According to this method, if N = A*B, where A-Superqueens and B-Superqueens are solvable, then, N-Superqueens problem can be reduced to solving A-Superqueens and B-Superqueens problems and combining them into a N*N board which is divided into A*A grid of B*B tiles, as exemplified in Figure 6. In Figure 6, N = 25, A = 5, and B = 5.

The general solution to the N-Queens problem is defined in terms of the concept of **knight-walk** and **D-solution**. If a solution to the N-Superqueens problem starts with a queen in the upper left-hand corner and proceeds with a queen in the third column of the second row, and with a queen in the fifth column of the third row, and so on, this solution is called knight-walk. If for N = A*B decomposition, A-Superqueens solution contains a superqueen in the upper left-hand corner and the solution of B is a knight-walk then this type of decomposition is called D-solution. It has been proved that [1]:

1. If N = A*B, and there is a solution to the N-Superqueens problem, then it is possible to construct a solution to the B*(A-1)-Queens problem.
2. If N = A*B, and C ≤ B, and there is a solution to the N-Superqueens and C-Superqueens problems, then it is possible to construct a solution to the B*(A-1)+C-Queens problem. The general approach follows the steps given below:

Step 1. Construct a D-solution to the A * B superqueens problem.
Step 2. Replace the top B rows and B leftmost columns of the decomposition with C rows and C columns.
Step 3. Place C-knight-walk to the C * C tile in the upper left-hand corner of the board.

This algorithm cannot produce solutions to the N-Queens problem for N = 8, 9, 14, 15, 26, 27, 38, 39.

Algorithm 8. Divide and Conquer Algorithm

```
procedure divide_and_conquer(N);
  var Even, A,B,C;
begin
    Even := false; {Solve for an odd number. If N is
            even, add one, solve for the resulting board,
            and drop the upper left hand corner}
    if (N is even) then begin
       N := N + 1;
       Even := true;
    end;
```

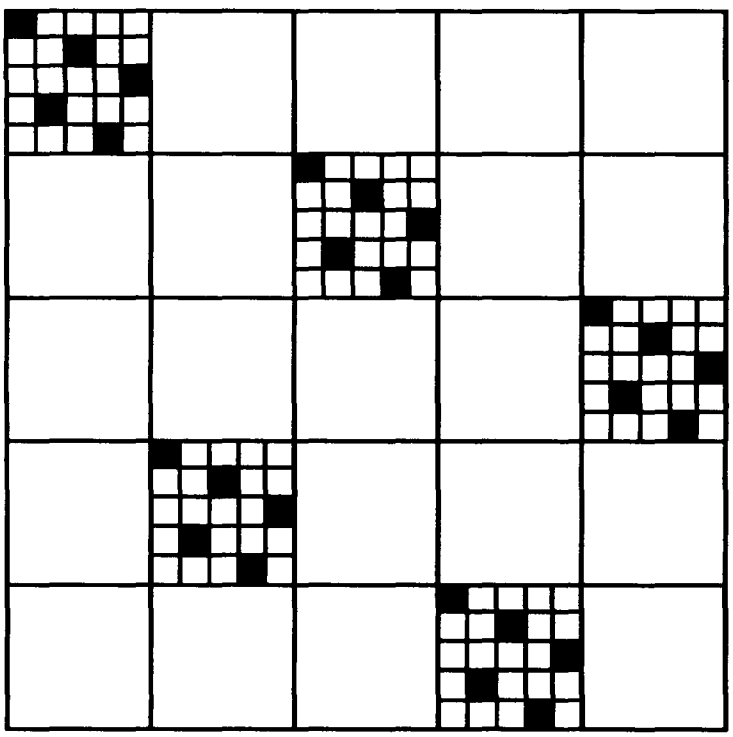

Figure 6. An example of a Decomposition solution.

```
    if (N mod 3 <> 0) then board := regular_solution(N)
    else  begin
        find B, C such that 4B + C = N;
        {Set A=5, find B and C}
        board := general_solution(N, 5, B, C);
    end;
    if (even) then board := board-minus-corner-queen;
    {Remove the upper left corner that was added above}
end
```

6. CONSTRUCTIVE APPROACHES

In the previous sections, we have analyzed several algorithms designed to solve the N-Queens problem. However, the problem has not only been addressed by algorithmic approaches. In addition, there are several other attempts intended to solve the problem using constuctive methods. The difference between the algorithmic and the constructive approaches is that the latter does not rely on search techniques, but generate solutions using predefined patterns. [2,18,11,23,8] are the examples in the literature to the constructive methods.

Furthermore, there are several other attempts aimed to provide an insight to the problem by showing its connections to the mathematical structures, such as polygons, circulants, and magic squares. In [8], Erbas and Tanik analyzed the N-Queens problem from the viewpoint of the polygons, and, demonstrated that star and compound polygons can be used to generate solutions to the N-Queens problem. The connections between the N-Queens problem and the magic squares constructed using the uniform step method are demonstrated in [10].

7. CONCLUSION

We have introduced several different definitions to the N-Queens problem. Each of these definitions provides a different perspective. We have classified the algorithms related to the N-Queens problem regarding their objectives and behaviors. We have also implemented each of these algorithms using Turbo Pascal. In addition, we have developed an integrated environment on which one can run experimentations with all these algorithms [3,33]. This environment is available upon request.

Acknowledgment

We would like to thank Travis A. Diamond, Waleed Hafiz, and Nader Rafraf for their stimulating discussions on queens. We would also like to recognize the efforts of James Allis in coding and documenting the user interface of our environment.

References

1. B. Abramson and M. Yung, "Divide and conquer under global constraints: A solution to the n-queens problem," *J. Parall. Dist. Comp.* 6 (1989) 649-662.

2. W.Ahrens, *Mathematische Unterhaltungen und Spiele,* vol. 1, Teubner, (1921).

3. J.A. Allis, W.M. Molaison, M.M. Tanik, *The N-Queens Workstation User's Manual and Technical Reference Manual,* SMU, TR 90-CSE-22, (June 1990).

4. C. Berge, *Graphs and Hypergraphs,* North-Holland Publishing Company, New York, (1973).

5. G. Brassard, and P. Bratley, *Algorithmics Theory and Practice,* Prentice-Hall, New Jersey, (1988).

6. J. Cohen, "Non-Deterministic Algorithms," *Computing Surveys,* Vol. 11, No. 2, (June 1979) 79-94.

7. C. Erbas, and M.M. Tanik, *N-Queens Problem and Its Algorithms,* SMU, TR 91-CSE-8, (February 1991).

8. C. Erbas, and M.M. Tanik, *N-Queens Problem and Its Connections to the Polygons,* SMU, TR 91-CSE-21, (June, 1991).

9. C. Erbas, and M.M. Tanik, "Storage Schemes for Parallel Memory Systems and the N-Queens Problem," *The 15th ASME ETCE Conference, Computer Applications Symposium,* Houston, Texas, (January 26-30, 1992).

10. C. Erbas, N. Rafraf, and M.M. Tanik, *Magic Squares Constructed by the Uniform Step Method Provide Solutions to the N-Queens Problem,* SMU, TR 91-CSE-25, (August 1991).

11. B.J. Falkowski, and L. Schmitz, "A note on the queens' problem," *Inform. Process. Lett.* 23 (1986) 39-46.

12. R.W. Floyd, "Nondeterministic Algorithms," *Journal of ACM,* Vol. 14, No. 4, (October 1967) 636-644.

13. L.R. Foulds, and D.G. Johnson, "An application of graph theory and integer programming: Chessboard non-attacking puzzles," *Math. Mag.* 57 (2) (March 1984) 95-104.

14. C.F. Gauss, *Briefwechsel zwischen C.F. Gauss und H.C. Schumacher,* herausgeg, von Peters, 6. Band, Altona (1865) 105-122.

15. J. Gingsburg, "Gauss's arithmetization of the problem of n queens," *Scripta Math.* 5 (1939) 63-66.

16. W.S. Golomb, and L.D. Baumert, "Backtrack Programming," *J. ACM* 12 (4) (1965) 516-524.

17. S. Gunther, "Zur mathematisches theorie des Schachbretts," *Archiv der Mathematik und Physik* 56 (1874) 281-292.

18. E.J. Hoffman, J.C. Loessi, and R.C. Moore, "Constructions for the Solution of the m Queens Problem," *Mathematics Magazine,* (March-April 1969), 66-72.

19. E. Lucas, *Recreations mathematiques,* 1891, Reprinted by A.Blanchard, Paris, 1960.

20. B. Nadel, "Representation selection for constraint satisfaction: A case study using n-queens," *IEEE Expert,* June (1990) 16-23.

21. P. Naur, "An experiment on program development," *BIT* 12(1972) 347-365.

22. G. Polya, "Uber die 'doppelt-periodischen' losungen des n-damen-problems," in W. Ahrens, *Mathematische Unterhaltungen und Spiele,* (1918) 364-374.

23. M. Reichling, "A Simplified Solution of the N queens' problem," *Information Processing Letters,* 25(1987) 253-255.

24. I. Rivin and R. Zabih, "An Algebraic Approach to Constraint Satisfaction Problems," *Proceedings of the International Conference on Artificial Intelligence (IJCAI-9)*, vol. 1, (1989), 284-289.

25. J.S. Rohl, "Generating Permutation by choosing," *The Computer Journal*, Vol. 21, Number 4, (1978), pp. 302-305.

26. J.S. Rohl, "Letter to the Editor," *The Computer Journal*, Vol. 22, Number 2, (1979) 191.

27. J.S. Rohl, "A faster lexiographical n-queens algorithm," *Inform. Process. Lett.* 17 (1983) 231-233.

28. D.R. Smith, "KIDS: A semiautomatic program development system," *IEEE Trans. Soft. Eng.* 16 (9) (1990) 1024-1043.

29. R. Sosic and J. Gu, "A polynomial time algorithm for the n-queens problem," *SIGART* 1 (3) (1990) 7-11.

30. H.S. Stone and J.M. Stone, "Efficient search techniques - An empirical study of the n-queens problem," *IBM J. Res. Develop.* 31 (4) (July 1987) 464-474.

31. Y. Takefuji, *Neural Network Parallel Computing*, Kluwer Academic Publishers, Hingham, MA, (1992).

32. M.M. Tanik, *A graph model for deadlock prevention*, Phd Dissertation, Texas A&M University (1978).

33. M.M. Tanik, K. Townsend, W. Cheng, and S.L. Stepoway, *Graphics Programming with Turbo Pascal*, Wordware, Dallas, TX, (1992).

34. R.W. Topor, "Fundamental Solutions of the Eight Queens Problem," *BIT*, Vol. 22, (1982) 42-52.

35. N. Wirth, "Program development by stepwise refinement," *Comm. ACM*, 14 (4) (1971) 221-227.

36. A.M. Yaglom and I.M. Yaglom, *Challenging Mathematical Problems with Elementary Solutions*, Holden-Day, (1964) 92-98.