A Novel Approach to Solving N-Queens Problem

Md. Golam KAOSAR
Department of Computer Engineering
King Fahd University of Petroleum and Minerals
Dhahran 31261, KSA

and

Mohammad SHORFUZZAMAN and Sayed AHMED
Department of Computer Science
University of Manitoba
Winnipeg, MB R3T 2N2, Canada

ABSTRACT

Placing *n* mutually non-attacking queens on an *n* by *n* chessboard is a classical problem in the artificial intelligence (AI) area. Over the past few decades, this problem has become important to computer scientists for providing solutions to many useful and practical applications. In this paper, we present a novel approach to solving the *n*-queens problem in a considerably less execution time. The proposed technique works by directly placing the queens on the board in a regular pattern based on some queen-movement rules.

Keywords: Artificial Intelligence, backtracking, horse movement, divide-and-conquer, permutation.

1. INTRODUCTION

The *n*-queens problem, originating from the 8-queens problem, has been studied for more than a century. The *n*-queens problem, originally introduced in 1850 by Carl Gauss, may be stated as follows: find a placement of *n* queens on an *n* by *n* chessboard so that no two queens attack each other (i.e., so that no two are in the same row, column or diagonal). This classical combinatorial search problem has traditionally been used for exploring new AI search strategies and algorithms. To date, this problem has found many scientific and engineering applications including VLSI routing and testing, maximum full range communication, parallel optical computing, and many more.

Solutions to the *n*-queens problem can be represented as an *n*-tuples ($q_1, q_2, q_3, \ldots q_n$) since each queen must be on a different row and column. Position of a number in the tuples represents the column position of a queen while the value of the number represents the row position of the queen (counting from the top). Figure 1 shows a 5-tuples, which represents a solution to the 5-queens problem. Empirical observations show that the number of solutions increases exponentially with increasing *n* [2]. Table 1 gives the number of solutions vs. the number of queens, based on the empirical results [1].

		Q		
Q				
			Q	
	Q			
				Q

(2, 4, 1, 3, 5)

Figure 1: Solution to 5-queens problem

n	Number of Solutions
6	4
7	40
8	92
9	352
10	724
11	2680
12	14200

Table 1: The number of solutions for the *n*-queens problem

The *n*-queens problem has three variants: finding one solution, finding a family of solutions, and finding all solutions. This paper presents a direct placement algorithm for solving *n*-queens problem. The proposed algorithm gives only one solution for a particular value of *n.* The algorithm works by directly placing the queens on the chessboard according to some queen-movement rules and does not require any kind of searching or backtracking.

The remainder of the paper is organized as follows. Section 2 reviews the related work done in the literature. Section 3 then presents the proposed technique for solving *n*-queens problem. Finally, the paper ends with conclusions and future research directions in section 5.

2. RELATED WORK

Numerous solutions to *n*-queens problem have been published since the original problem was proposed. This section presents a representative set of approaches that attempts to solve the *n*-queens problem.

A number of search based algorithms have been developed to generate all possible solution sets for a given *n* by *n* board. Backtracking [3, 4, 5] algorithms are among the most widely used search algorithms for solving *n*-queens problem that systematically generate all possible solutions. 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. To explore all possible vectors possible vectors, the algorithm starts by placing a queen on the first column of the first row and continues placing the queens on the other rows**,** while maintaining the constraints that are imposed by the previously placed queens. If the algorithm reaches a row for which all squares are already attacked by the other queens, it backtracks to the previous row and explores other vectors. In practice, however, backtracking approaches provide a very limited class of solutions for large size boards because it is difficult for a backtracking search to find solutions that are significantly distinct in the solution space [2].

Several authors have proposed other efficient search techniques to overcome this problem. These methods include search heuristic methods [3] and local search and conflict minimization techniques [2]. A probabilistic algorithm utilizes gradient-based heuristic search [6]. 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 collision 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.

Permutation generation algorithms [7] provide a trial and error solution to the *n*-queens problem. 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. One problem with these algorithms is that they do not test the partial arrangements.

Topor [8] develops an algorithm that uses a direct method for generating fundamental solutions. The solutions which can be transformed to each other by rotations form a group. This 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.

Recently, advances in research in the area of neural networks have led to several solutions to the *n*-queens problem using neural networks [9, 10]. Specifically, the use of Hopfield networks has been applied to the *n*-queens problem by Mandziuk [10]. The Hopfield neural network is a simple artificial network which is able to store certain patterns in a manner similar to the brain in that the full pattern for a given problem can be recovered if the network is presented with only partial information. The ability of neural networks to adapt and learn from information has applications in optimization problems beyond the *n*-queens problem.

Abramson and Yung [11] present a divide-and-conquer algorithm that provides a family of solutions by means of splitting the input into distinct subsets. Most recent study shows that genetic algorithms [12] can be used to solve *n*-queens problem. These algorithms are search and optimization procedures based on three basic biological principles: selection, crossover, and mutation. Potential solutions are represented as individuals that are evaluated using a fitness function that determines how close a wrong solution is to a correct one. A global parallel implementation of this algorithm is also presented to increase computation speed.

3. PROPOSED SOLUTION

We propose a direct placement method to solve the *n*-queens problem. Our algorithm is based on the principle of finding one solution for a particular value of *n* and adopts a very simple technique to place queens on the board. The technique is simple in the sense that it does not require any searching or backtracking to find the positions of queens. The remainder of this section describes the algorithm and pinpoints its key features.

After performing a considerable investigation, we come up with a series of numbers that indicate the values of *n*:

8, 9, 14, 15, 20, 21, 26, 27, 32, 33, 38, 39, 44, 45, 50…
It can be represented as:
Start with n=8 and

1. n, (n+1)
2. n=n+6, Repeat 1.

This series possesses a nice property that makes the placement of queens on the board much easier. For every fifth element (i.e., value of n) in the series, the arrangement of queens on the board follows the same rule. For example, the rule used to place queens when the value of n is 8, 20, or 32 is same. As every fifth element in the series holds the same arrangement of queens, we get four distinct secondary series like (i) 8, 20, 32, 44... (ii) 9, 21, 33, 45... (iii) 14, 26, 38, 50… and (iv) 15, 27, 39, 51… where each of these series has unique arrangement of queens. So, for the main series we have four types of arrangements. Besides, for all the values of n that do not fit in the series, there is only one unique arrangement of queens on the board. Therefore, our proposed algorithm considers five types of arrangements of queens for any value of n.

Arrangement 1

The secondary series 8, 20, 32, 44 ... fall under this arrangement of queens. This arrangement works as follows.
Step 1: Start placing a queen in position (2, 1). (We count rows and columns from the upper left corner of the n by n chessboard.)
Step 2: Place the next queen with a horse movement (row and column are increased by two and one respectively). Continue placing n/2 queens this way.
Step 3: Place a queen in position (1, n/2 + 2).
Step 4: Place remaining queens according to two movement rules alternatively: a reverse horse movement (row is increased by two and column is decreased by one) and a custom movement (row is increased by two and column is increased by three). Start placing with the reverse horse movement.

Table 2 shows the solution for 8-queens problem according to this arrangement. Numbers in the board represents queens.

					5		
1							
				6			
	2						
							7
		3					
						8	
			4				

Table 2: Solution for 8-queens problem

Arrangement 2

The secondary series 9, 21, 33, 45 ... fall under this arrangement of queens. This arrangement works as follows.
Step 1: Start placing a queen in position (1, 3).
Step 2: Place the next queen with a horse movement (row and column are increased by two and one respectively). Continue placing (n/2 +1) queens this way.
Step 3: Place a queen in position (2, n/2 + 5).
Step 4: Place remaining queens (except last two) according to two movement rules alternatively: a reverse horse movement (row is increased by two and column is decreased by one) and a custom movement (row is increased by two and column is increased by three). Start placing with the reverse horse movement.
Step 5: Place last two queens starting from position (n -1, 1) and a horse movement (row is decreased by two and column is increased by one).

Table 3 shows the solution for 9-queens problem according to this arrangement.

Arrangement 3

The secondary series 14, 26, 38, 50... fall under this arrangement of queens. This arrangement works as follows.
Step 1: Start placing a queen in position (2, 2).

		1						
								6
			2					
							7	
				3				
	9							
					4			
8								
						5		

Table 3: Solution for 9-queens problem

Step 2: Place the next queen with a horse movement (row and column are increased by two and one respectively). Continue placing n/2 queens this way.
Step 3: Place a queen in position (1, n/2 + 3).
Step 4: Place remaining queens (except the last one) according to two movement rules alternatively: a reverse horse movement (row is increased by two and column is decreased by one) and a custom movement (row is increased by two and column is increased by three). Start placing with the reverse horse movement.
Step 5: Place the last queen in position (n-1, 1).

Table 4 shows the solution for 14-queens problem according to this arrangement.

Arrangement 4

The secondary series 15, 27, 39, 51 ... fall under this arra-

									8				
	1												
								9					
		2											
											10		
			3										
										11			
				4									
													12
					5								
												13	
						6							
14													
							7						

Table 4: Solution for 14-queens problem

										8				
		1												
									9					
			2											
												10		
				3										
											11			
					4									
														12
						5								
													13	
							6							
	15													
								7						
14														

Table 5: Solution for 15-queens problem

-ngement of queens. This arrangement works as follows:
Step 1: Start placing a queen in position (2, 3).
Step 2: Place the next queen with a horse movement (row and column are increased by two and one respectively). Continue placing $n/2$ queens this way.
Step 3: Place a queen in position (1, $n/2 + 4$).
Step 4: Place remaining queens (except last two) according to two movement rules alternatively: a reverse horse movement (row is increased by two and column is decreased by one) and a custom movement (row is increased by two and column is increased by three). Start placing with the reverse horse movement.
Step 5: Place last two queens starting from position (n, 1) and a horse movement (row is decreased by two and column is increased by one).
Table 5 shows the solution for 15-queens problem according to this arrangement.

Arrangement 5

The values of n that do not fit in the main series fall under this arrangement of queens. This arrangement works as follows.

Step 1: Start placing a queen in position (2, 1).
Step 2: Place the next queen with a horse movement (row and column are increased by two and one respectively). Continue placing $n/2$ queens this way.
Step 3: Place a queen in position (1, $n/2 +1$).
Step 4: Place remaining queens with the horse movement.

Table 6 shows the solution for 7-queens problem according to this arrangement.

The algorithm works by checking the value of n whether it fits into the series and then using the appropriate arrangement type accordingly. Checking is done by generating the series. The series is generated using the general expression mentioned earlier. If the value of n fits into the series, the appropriate arrangement type is found by taking modulus of n by 4. Arrangement type 1, 2, 3, and 4 are chosen for the modulus result 0, 1, 2, and 3 respectively. If the value of n does not fit into the series, arrangement type 5 is chosen.

			4			
1						
				5		
	2					
					6	
		3				
						7

Table 6: Solution for 7-queens problem

We have verified the soundness of the proposed algorithm by checking the conflict between any two queens. To avoid a conflict, it is obvious that each queen must be in a separate row and column from each of the others. This condition can be checked by using the *n*-tuples representation of a solution. Earlier we mentioned that any solution can be expressed as an *n*-tuples ($q_1, q_2, \ldots, q_n$), where q_i (with i=1, 2, 3… n) gives the row of the queen in column i. Thus, the solution from Table 6 (for example) can be expressed as follows. Row and column conflict can be detected by merely comparing the values of q_i and i respectively.

$q_{i=}$	2	4	6	1	3	5	7
i	1	2	3	4	5	6	7

To assure that we do not have any diagonal conflict, we require that for all $1 \le i \le j \le n$

$$\left|q_j - q_i\right| \neq j - i.$$

4. CONCLUSIONS

This paper investigated the property of a nice series and presented a fast algorithm that works by directly placing the queens on the board using that series. Four types of queen arrangements are considered for solving the problem that fits into the series and a unique arrangement works for any value of *n* that does not fit into the series. We have verified the soundness of the algorithm by checking conflict between any two queens. We claim our proposed algorithm to be the one to solve the *n*-queens problem in considerably less execution time. The future work would be to investigate for producing a family of solutions within the boundary of direct placement of queens.

5. REFERENCES

[1] C. Erbas, S. Sarkeshik, and M. M. Tanik. "Different perspectives of the *n*-queens problem". In **Proceedings of the ACM 1992 Computer Science Conference**, 1992.

[2] R. Sosi and J. Gu. "Efficient local search with conflict minimization: A case study of the *n*-queens problem". **IEEE Transactions on Knowledge and Data Engineering**, Vol. 6, No. 5, pp. 61-68, 1994.

[3] Laxmikant V. Kale. "An almost perfect heuristic for the *n* non-attacking queens problem". **Information Processing Letters**, 34:173-178, 1990.

[4] B.A. Nadel. "Representation selection for constraint satisfaction: A case study using *n*-queens". **IEEE Expert**, June, pp. 16-23, 1990.

[5] H.S. Stone and J.M. Stone. "Efficient search techniques - An empirical study of the *n*-queens problem". **IBM Journal of Research and Development**, 31: 464-474, 1987.

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

[7] J.S. Rohl. "Generating permutation by choosing". **The Computer Journal,** Vol. 21, No 4, pp.302-305, 1978.

[8] R. W. Topor. "Fundamental solutions of the eight queens problem". **BIT,** Vol. 22, pp. 42-52, 1982.

[9] O. Shagrir. "A neural net with self-inhibiting units for the *n*-queens problem". **International Journal of Neural Systems**, Vol. 3, No. 3, pp. 249-252, 1992.

[10] J. Mandziuk. "Solving the *n*-queens problem with a binary Hopfield-type network synchronous and asynchronous model". **Biological Cybernetics**, Vol. 72, No. 5, pp. 439-446, 1995.

[11] Bruce Abramson and Moti Yung. "Divide and conquer under global constraints: A solution to the *n*-queens problem". **Journal of Parallel and Distributed Computing**, 6:649-662, 1989.

[12] M. Bozikovic, M. Golub and L. Budin. "Solving *n*-queen problem using global parallel genetic algorithm". **EUROCON**, Vol 2, pp 22-24, 2003.