POLİTEKNİK DERGİSİ

JOURNAL of POLYTECHNIC

ISSN: 1302-0900 (PRINT), ISSN: 2147-9429 (ONLINE)
URL: http://dergipark.org.tr/politeknik

A linear time pattern based algorithm for n-queens problem

N-vezir problemi için lineer zamanlı örüntü temelli algoritma

***Yazar(lar) (Author(s))**: Bergen KARABULUT[1], Atilla ERGÜZEN[2], Halil Murat ÜNVER[3]*

ORCID[1]: 0000-0003-0755-1289

ORCID[2]: 0000-0003-4562-2578

ORCID[3]: 0000-0001-9959-8425

Bu makaleye şu şekilde atıfta bulunabilirsiniz(To cite to this article): Karabulut B., Ergüzen A. and Ünver H. M., "A linear time pattern based algorithm for n-queens problem", ***Politeknik Dergisi***, 25(2): 615-622, (2022).

Erişim linki (To link to this article): http://dergipark.org.tr/politeknik/archive

DOI: 10.2339/politeknik.762967

A Linear Time Pattern Based Algorithm for N-Queens Problem

Önemli noktalar (Highlights)

- *This study proposes a new algorithm for a benchmark problem.*
- *The proposed algorithm produces solution(s) for all n values.*
- *The proposed algorithm runs in linear time with O(n) time complexity.*
- *Even very large n values, it produces solution(s) without using complex calculations or searching.*
- *This study provide an effective solution to the handled problem.*

Aim

The purpose of this study is to develop an algorithm producing solution(s) in large n values of n-queens problem.

Design & Methodology

A pattern based approach that produces at least one unique solution for all n values (n>3) was detected and it used to create an algorithm.

Originality

The developed algorithm can produce solutions without using complex calculations or searching. In this respect, the proposed method provides efficiency compared to similar studies.

Findings

*The developed algorithm with **O(n)** time complexity produces quite faster solution to n-queens problem and even in some values this algorithm produces (n-1)/2 unique solutions in linear time.*

Conclusion

In this study, a linear time pattern based algorithm was proposed for the n-queens problem. The developed method provides an important contribution in terms of producing solutions for large values in linear time.

Declaration of Ethical Standards

The authors of this article declare that the materials and methods used in this study do not require ethical committee permission and/or legal-special permission.

A Linear Time Pattern Based Algorithm for N-Queens Problem

Araştırma Makalesi / Research Article

Bergen KARABULUT*, Atilla ERGÜZEN, Halil Murat ÜNVER

Computer Engineering Department, Engineering Faculty, Kırıkkale University, 71450 Yahşihan, Türkiye

(Geliş/Received : 06.07.2020 ; Kabul/Accepted : 17.12.2020 ; Erken Görünüm/Early View : 18.12.2020)

ABSTRACT

The n-queens problem is the placing of *n* number of queens on an *nxn* chessboard so that no two queens attack each other. This problem is important due to various usage fields such as VLSI testing, traffic control job scheduling, data routing, dead-lock or blockage prevention, digital image processing and parallel memory storage schemes mentioned in the literature. Besides, this problem has been used as a benchmark for developing new artificial intelligence search techniques. However, it is known that backtracking algorithms, one of the most frequently used algorithms to solve this problem, cannot produce all solutions in large *n* values due to the exponentially growing time complexity. Therefore, various methods have been proposed for producing one or more solutions, not all solutions for each *n* value. In this study, a pattern based approach that produces at least one unique solution for all *n* values (*n*>3) was detected. By using this pattern, a new algorithm that produces at least one unique solution for even very large *n* values in linear time was developed. The developed algorithm with ***O(n)*** time complexity produces quite faster solution to n-queens problem and even in some values, this algorithm produces (*n*-1)/2 unique solutions in linear time.

Keywords: N-queens problem, constraint satisfaction problem, NP-hard, NP-complete, optimization problem.

N-Vezir Problemi için Lineer Zamanlı Örüntü Temelli Algoritma

ÖZ

N-vezir problemi, *nxn* boyutundaki bir satranç tahtasına *n* adet vezirin herhangi iki vezir birbirine saldırmayacak şekilde yerleştirilmesidir. Bu problem literatürde değinilen VLSI testi, trafik kontrol işi planlama, veri yönlendirme, ölümcül kilitlenme ya da tıkanıklık önleme, dijital görüntü işleme ve paralel bellek depolama şemaları gibi çeşitli kullanım alanlarından dolayı önemlidir. Ayrıca bu problem, yeni yapay zekâ arama tekniklerinin geliştirilmesi için bir referans noktası olarak kullanılmaktadır. Fakat bilindiği üzere bu problemin çözümde sıklıkla kullanılan geri-izleme algoritmaları, katlanarak büyüyen zaman karmaşıklığından dolayı büyük *n* değerleri için tüm çözümleri üretememektedir. Bu nedenle, her bir *n* değeri için tüm çözümleri bulmak yerine bir veya daha fazla çözüm üretebilmek için çeşitli yöntemler önerilmiştir. Bu çalışmada, tüm *n* değerleri (*n*>3) için en az bir tane eşsiz çözüm üreten bir örüntü tespit edilmiştir. Bu örüntü kullanılarak, çok büyük *n* değerlerinde bile lineer zamanda en az bir tane eşsiz çözüm üreten yeni bir algoritma geliştirilmiştir. ***O(n)*** zaman karmaşıklığı ile geliştirilen algoritma, n-vezir problemine oldukça hızlı çözüm üretmektedir ve hatta bazı değerler için lineer zamanda (*n*-1)/2 adet eşsiz çözüm üretmektedir.

Anahtar Kelimeler: N-vezir problemi, kısıt sağlama problemi, NP-hard, NP-complete, optimizasyon problemi.

1. INTRODUCTION

The n-queens problem is the placement of *n* number of queens on an *nxn* chessboard in such a way that no two queens are on the same row, column, or diagonal. Therefore, there can be only one queen on each row, column and diagonal. This problem was initially suggested as 8-queens problem by a chess player Max Bezzel in 1848 [1]. It was later extended to the n-queens problem, with *n* queens on an *nxn* board. The various application fields of this problem, which has been studied more than a century ago, have been emphasized in the literature. Sosic and Gu [2] stated that the n-queens problem has practical applications in VLSI testing and traffic control. Waqas and Bhatti [3] have listed applications of n-queens problem and (n+1) queens problem for real world problem as job/shop scheduling, data routing, dead-lock or blockage prevention, efficient resource management in computer systems, task assignment in multiprocessors, digital image processing and parallel memory storage schemes. It has also been used in various practical applications. Wang et al. [4] presented pixel decimation technique using the n-queen lattice and presented an application for block-based motion estimation using this novel technique. Bell and Stevens [5], have presented one of the best survey about the n-queens problem and have mentioned applications of the problem in detail.

Many studies have been conducted to offer a solution to the n-queens problem which was suggested initially in 1848. Solutions have been sought for the problem by using various methods including optimization techniques, parallel programming, mathematical approaches, backtracking algorithms and different patterns. Different methods have been applied to solve

*Sorumlu Yazar (Corresponding Author)
e-posta : brgnkarabulut@gmail.com

this problem, which has importance for various fields. The studies in the literature can be divided into three categories; finding one solution, finding more than one solutions but not all and finding all solutions. One of the most frequently used method for solving n-queens problem is backtracking search that generate all possible solutions. Murali et al. [6] simulated the N-queens problem using backtracking algorithm. Güldal et al. [7] proposed a hybrid approach by integrating sets and backtracking. They have removed the threatened cells in order to decrease the number of trial and error steps. Due to the exponentially growing time complexity of backtracking search [8], it cannot produce all possible solutions in large *n* values [9]. For this reason, methods which produce one or more solutions, instead of all solutions for each *n* value were developed.

N-queens problem is also a combinatorial optimization problem [10] and this problem has been used as a benchmark for developing new AI search techniques [11, 12]. In the literature, there are different studies in which artificial intelligence and computational intelligence techniques have been used to solve the problem. Meng and Wu [13] presented a Hybrid Genetic Algorithm in their study to solve the problem. Khan et al. [14] was proposed a solution for n-queen problem based on Ant Colony Optimization. In their study Turky and Ahmad [15] have been used Genetic Algorithm to produce solution to the problem. Motameni [16] have used Gravitational Search Algorithm to solve n-queens problem. Maazallahi et al. [17], presented a DNA computing model based on Adleman-Lipton model to solve the n-queens problem. This model provides all solutions and solve problem in a polynomial time complexity. In other studies, Biogeography based optimization [18], particle swarm optimization Amooshahi et al. [19] were used to solve this problem.

However, in recent years more algorithmic studies have been done and algorithms have been proposed for solution of the problem. Abramson and Yung [1], developed a new linear time divide and conquer algorithm to solve n-queens problem and a related problem that is the toroidal problem. Sosic and Gu [20], presented a new probabilistic local search algorithm. This algorithm runs in polynomial time and based on a gradient based heuristic. Their algorithm finds a solution for extremely large size n-queens problem. It is stated that previous AI search algorithms can find solution with *n* up to about 100, this method can find a solution with *n* up to 500,000. In another study, Sosic and Gu [11] developed two gradient-based heuristic based probabilistic local search algorithms called Queen Search 2 (QS2) and Queen Search 3 (QS3). QS2 and QS3 algorithms run in almost linear time and can find a solution for extremely large size n-queens problems. With QS3 algorithm, a solution can be found for 500,000 queens in approximately one and a half minutes. In their subsequent study, Sosic and Gu [9] presented a linear time algorithm. With this algorithm, a solution can be found for 3,000,000 queens in approximately 55 seconds. In a new study they did in the following years, Sosic and Gu [21] presented an efficient local search algorithm. This algorithm runs in linear time. They can find a solution for 3,000,000 queens using a workstation computer. Lijo and Jose [22], proposed an algorithm that has less computational complexity compared with backtracking algorithm. Their algorithm based on arithmetic progression and predicts potential candidate solutions. The proposed algorithm has reduced the time complexity by $O(n^3)$.

El-Qawasmeh and Al-Noubani [23], developed an algorithm working in linear time for the problem. Solutions have a modular structure and queens are placed in pre-calculated positions, allowing for producing a fast solution to the problem. They used 3 different strategies in which they divided chessboard into two parts. They have directly placed queens on non-overlapping points beginning from the top of the board for the first half and beginning from the bottom of the board for the second half. They obtained a unique solution for 75% of *n* values, and used the algorithm developed by Sosic for the remaining 25%. They proved their strategies with a software they developed and with mathematical operations. In conclusion, they developed an algorithm that works 100 faster in deterministic linear time in only 75% of *n* values for *n*>3 compared to Sosic's algorithm. Rohith et al. [24], obtained a unique solution in polynomial time for (*n*+1)x(*n*+1) chessboard by expanding *n*x*n* solution using a pattern that they observed.

In this study, a pattern-based algorithm running in linear time for n-queens was developed. A pattern that produces at least one unique solution for every consecutive 6 values where *n*>3 was observed. Besides, this pattern produces (*n*-1)/2 unique solutions in one in every consecutive six values. By using this pattern, a general algorithm was developed. With this algorithm, unique and fast solutions were obtained even for extremely large *n* values without resorting to complex calculations. The developed algorithm is iterative and it only assigns a position for queens in any *n* values. Computational complexity of the algorithm is $O(n)$.

2. METHODOLOGY

2.1. Notation for the N-Queens Problem

Any possible solution of the n-queens problem was represented as the n-tuple $(q_0, q_1, \ldots q_n)$. In this notation, q_i is a column position on which the queen in the i-th row is placed. Figure 1 shows an example of n-tuple notation for *n*=4.

	q_0		
			q_1
q_2			
		q_3	

(2, 4, 1, 3)

Figure 1. Solution to 4-queens problem

2.2. Observed Pattern

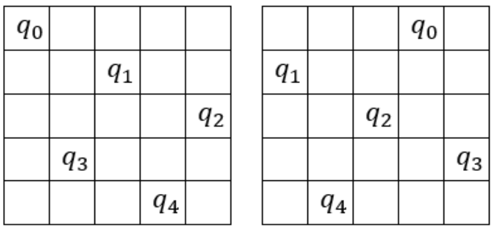
The observed pattern produces at least one unique solution for every value where *n*>3. It produces one unique solution in four, two unique solutions in one, three solutions in three and (*n*-1)/2 unique solutions in four in every consecutive twelve values. In Table 1 solutions of the pattern was presented. Firstly, the *n* values are divided into two categories according to whether they are even or odd. Then subcategories are determined by mode operation. There are 3 subcategories for both even and odd, so there are 6 subcategories in total. Solution sets were given for 6 subcategories in Table 1.

Shift operation: In this operation an existing solution is handled. In existing solution's set, the last queen is taken first and all other queens are shifted one position to the right. When this operation is applied to an existing solution, a new unique solution is obtained. However, this operation can be performed only ((*n*-1)/2)-1 times. So, totally (*n*-1)/2 solutions are obtained. An example was given in Figure 2. As shown in Figure 2.a, solution set of *n*=5 queens is (1, 3, 5, 2, 4). Shift operation can be applied in this value one times and new unique solution set is (4, 1, 3, 5, 2). So, two unique solutions are produced.

Table 1. Solution sets

Residue class		Solution(s)
n is even	(n+1) mod 3 = 0 & (n/2) mod 2 = 0 (e.g., n= 8, 20, 32, 44 …)	Solution 1= (2, 4, 6, … n, 3, 1, 7, 5, … (n-1), (n-3)) Solution 2= (4, 6, …n, 3, 1, 7, 5, … (n-1), (n-3), ***2***)
	(n+1) mod 3 = 0 & (n/2) mod 2 ≠ 0 (e.g., n= 14, 26, 38, 50 …)	Solution 1 = (***(n-1)***, 2, 4, … n, 3, 1, 7, 5, … (n-3), (n-5)) Solution 2 = (***n***, 3, 5, … (n-1), ***1***, 4, 2, 8, 6, … (n-2), (n-4)) Solution 3= (***(n-5)***, ***(n-1)***, 2, 4, … n, 3, 1, … (n-7), (n-9), ***(n-3)***)
	(n+1) mod 3 ≠ 0 (e.g., n= 4, 6, 10, 12, 16, 18 …)	Solution 1= (2, 4, … n, 1, 3, … (n-1))
n is odd	n mod 3 = 0 & (n-1) mod 4 = 0 (e.g., n= 9, 21, 33, 45 …)	Solution 1 = (***(n-1)***, 5, 3, 9, 7, … n, (n-2), 2, 4, … (n-3), ***1***) Solution 2 = (***1***, ***(n-1)***, 5, 3, 9, 7, … n, (n-2), 2, 4, … (n-3)) Solution 3 = (***(n-2)***, 4, 2, 8, 6, … (n-1), (n-3), 1, 3, … (n-4), ***n***)
	n mod 3 = 0 & (n-1) mod 4 ≠ 0 (e.g., n= 15, 27, 39, 51 …)	Solution 1 = (***(n-1)***, 3, 5, 7, … n, 4, 2, 8, 6, … (n-3), (n-5), ***1***) Solution 2 = (3, 5, 7, … n, 4, 2, 8, 6, … (n-3), (n-5), ***1***, ***(n-1)***) Solution 3 = (***(n-1)***, 3, 5, 7, … (n-2), ***1***, 4, 2, 8, 6, … (n-3), (n-5), ***n***)
	n mod 3 ≠ 0 (e.g., n= 5, 7, 11, 13, 17, 19 …)	Solution 1 = (1, 3, … n, 2, 4, … (n-1)) Solution 2 = ((n-1), 1, 3, … n, 2, 4, … (n-3)) ***(Shift operation)*** ⋮ Solution (n-1)/2 = (4, 6, … (n-1) ,1, 3, … n, 2) ***(Shift operation)***

**Non-consecutive numbers in the solution sets typed as bold and italic.*
**Shift operation: Create new solution by shifting the previous solution one column right.*



(1, 3, 5, 2, 4)
(a)

(4, 1, 3, 5, 2)
(b)

Figure 2. a) 1st solution of 5-queens b) 2nd solution of 5-queens

2.3. Proposed Algorithm

The proposed algorithm is based on a pattern and does not contain search operation. The positions of queens on the board are predetermined and the solution set is generated directly by the pattern. The algorithm works for *n*>3.

The main function calls the decision function called *determineSolutionFunction()* if *n* is greater than 3.

Main Function: *nQueens*

Set *n* = get the board size from user
if *n* >3 **then**
  Call *determineSolutionFunction*(*n*);
else quit

Function: *determineSolutionFunction*(*n*)

if *n* is even **then**
if (*n+1*) mod 3 is equal to zero **then**
if *n/2* mod 2 is equal to zero **then**
Call *Function1*(*n*)
else
Call *Function2*(*n*)
else
Call *Function*3(*n*)
else
if *n* mod 3 is equal to 0 **then**
if (*n-1*) mod 4 is equal to 0 **then**
Call *Function4*(*n*)
else
Call *Function5*(*n*)
else
Call *Function6*(*n*)

In decision function, a solution function is selected according to the mode operation and the program branches to the appropriate function. Six functions; *Funtion1*, *Funtion2*, *Funtion3*, *Funtion4*, *Funtion5* and *Funtion6* are defined for the 6 different subcategories determined in the pattern. By using these functions, solution sets for related *n* values are produced.

Procedure: *Function1*(*n*)

```
Set number=2, index=0
while number is smaller than or equal to n do
Set solutionArray[index] = number
Increment index by one and number by two
end while
Set tempIndex= index
Set number=1
while number is smaller than n do
Set solutionArray[index] = number
Increment index by one and number by two
end while
while tempIndex is smaller than n-1 do
Increase solutionArray[tempIndex] by two
Decrease solutionArray[tempIndex+1] by two
Increase tempIndex by two
end while
Call printSolution(solutionArray)
Set newSolutionArray[n-1] = solutionArray[0]
for i= 0, 1, 2, …, n-2 do
Set newSolutionArray[i] = solutionArray[i+1]
end for
Call printSolution(newSolutionArray)
```

Procedure: *Function2*(*n*)

```
Set number=2, index=1
Set solutionArray[0] = n-1
while number is smaller than or equal to n do
Set solutionArray[index] = number
Increment index by one and number by two
end while
Set tempIndex=index
Set number=1
while index is smaller than n do
Set solutionArray[index] = number
Increment index by one and number by two
end while
while tempIndex is smaller than n-1 do
Increase solutionArray[tempIndex] by two
Decrease solutionArray[tempIndex+1] by two
Increase tempIndex by two
end while
Call printSolution(solutionArray)
Copy SolutionArray to tempSolutionArray
for i= 0, 1, 2, …, n-1 do
if solutionArray[i] is equal to 0 then
Set solutionArray[i]=1
else
Increase solutionArray[i] by one
end for
Call printSolution(solutionArray)
Set newSolutionArray[0]=tempSolutionArray[n-1]
for i= 0, 1, 2, …, n-2 do
Set newSolutionArray[i+1]=tempSolutionArray[i]
end for
Call printSolution(newSolutionArray)
```

Procedure: *Function3*(*n*)

```
Set number=2, index=0
while number is smaller than or equal to n do
Set solutionArray[index] = number
Increase index by one and number by two
end while
Set number=1
while number is smaller than n do
Set solutionArray[index] = number
Increment index by one and number by two
end while
Call printSolution(solutionArray)
```

Procedure: *Function4*(*n*)

```
Set number=3, index=1
Set solutionArray[0] = n-1, solutionArray[n-1] = 1
while number is smaller than or equal to n do
Set solutionArray[index] = number
Increase index by one and number by two
end while
Set tempIndex=index
Set number=2
while number is smaller than n-2 do
Set solutionArray[index] = number
Increase index by one and number by two
end while
Set index=1
while index is smaller than tempIndex do
Increase solutionArray[index] by two
Decrease solutionArray[index+1] by two
Increase index by two
end while
Call printSolution(solutionArray)
```

```
Set newSolutionArray[0] = solutionArray[n-1]
for i=1, 2, 3, …, n-1 do
newSolutionArray[i] = solutionArray[i-1]
end for

Call printSolution(newSolutionArray)

for i= 0, 1, 2, …, n-1  do
if solutionArray[i]-1 is equal to 0 then
Set solutionArray[i]=n
else
Decrease solutionArray[i] by one
end for

Call printSolution(solutionArray)
```

Procedure: *Function5*(*n*)

```
Set number=3, index=1
Set solutionArray[0] = n-1,
while number is smaller than or equal to n do
Set solutionArray[index] = number
Increase index by one and number by two
end while

Set tempIndex=index
Set number=2
while number is smaller than n-1 do
Set solutionArray[index] = number
Increase index by one and number by two
end while

while tempIndex is smaller than n-1 do
Increase solutionArray[tempIndex] by two
Decrease solutionArray[tempIndex+1] by two
Increase tempIndex by two
end while

solutionArray[n-1] = 1
Call printSolution(solutionArray)

Set newSolutionArray[n-1] = solutionArray[n]
for i= 0, 1, 2, …, n-2 do
Set newSolutionArray[i] = solutionArray[i+1]
end for

Call printSolution(newSolutionArray)
Set tempValue= solutionArray[n/2]
Set solutionArray[n/2]= solutionArray[n-1]
Set solutionArray[n-1]= tempValue

Call printSolution(newSolutionArray)
```

Procedure: *Function6*(*n*)

```
Set number=1, index=0
while number is smaller than or equal to n do
Set solutionArray[index] = number
Increase index by one and number by two
end while

Set number=2
while number is smaller than n do
Set solutionArray[index] = number
Increase index by one and number by two
end while

Call printSolution(solutionArray)

while newSolutionArray[n-1] is not equal to 2 do
Set newSolutionArray[0] = solutionArray[n-1]
for i=1, 2, 3, …, n-1 do
Set newSolutionArray[i] = solutionArray[i-1]
end for

Call printSolution(newSolutionArray)
Copy newSolutionArray to solutionArray

end while
```

2.4. Control of Solutions

A correct solution requires that no two queens share the same row, column, or diagonal. If the place of the q_i queen is $B[X_1, Y_1]$ and the place of the q_{i+1} is $B[X_2, Y_2]$, then;

- $X_1 \neq X_2$ (not in the same row)
- $Y_1 \neq Y_2$ (not in the same column)
- $|X_1 - X_2| \neq |Y_1 - Y_2|$ (not in the same diagonal)

If all the requirements above are achieved, it means that no two queens threaten each other. The correctness of the solution can be checked whether all requirements for *n* value queens placed in *n*x*n* board are achieved. To control solution a check function called *solutionCheck* was defined. This function is used to check whether the solutions are correct.

Procedure: *solutionCheck*(*solutionArray*: array[1…n] of integer)

```
for k=0, 1, 2, …, n-1 do
for i=k+1, k+2, …, n-1 do
if  solutionArray[k] is equal to solutionArray[i]) or |k-i| is equal to
    |solutionArray[k]-solutionArray[i]  then
return false
end for
end for
return true
```

3. RESULTS AND DISCUSSION

In this study, a method which runs in linear time and produces at least one unique solution for the solution of n-queens problem has been studied. In the conducted study, a pattern was observed which produce at least one unique solution for *n*>3 in 6 different categories. This pattern produces different numbers of solutions for each

Table 2. Number of solutions

Number of queens *n*	**Number of solutions**
4	1
5	(n-1)/2=2
6	1
7	(n-1)/2=3
8	2
9	3
10	1
11	(n-1)/2=5
12	1
13	(n-1)/2=6
14	3
15	3

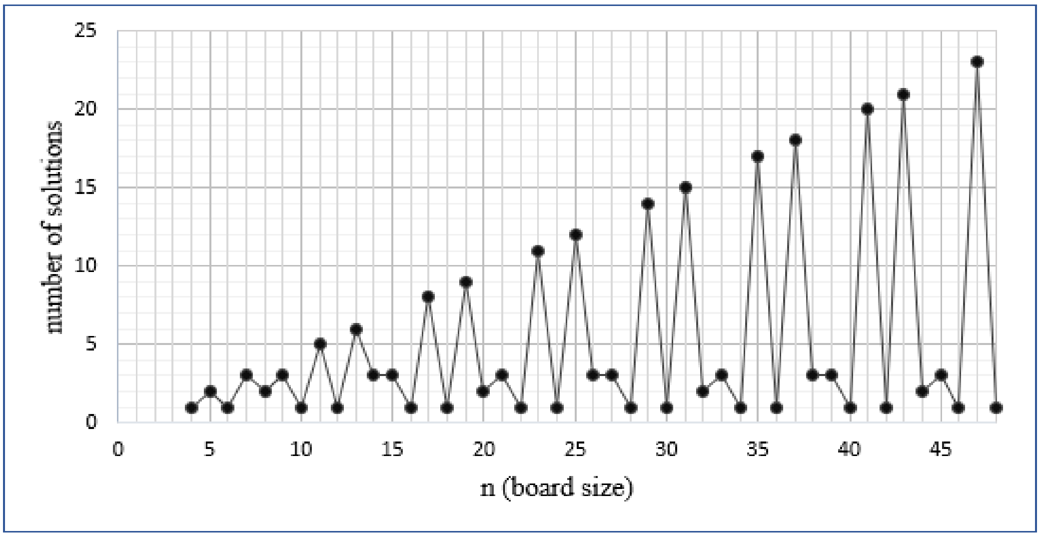


Figure 3. Number of solutions

category. In Table 2, number of solutions obtained with the pattern for related *n* values was presented. Only for the first 12 numbers, number of solutions are given in Table 2, since these number of solution numbers are repeated every consecutive 12 values. As stated in the table, a fixed number of solutions are obtained at some values. However, in some values (n-1)/2 solutions are obtained.

The number of obtained solutions is presented graphically in Figure 3. When the Figure 3 is examined, it is seen that in some values, a fixed number of solutions are obtained, whereas in some values, the number of solutions increases linearly.

A linear time algorithm was developed using the specified pattern. In order to test the developed algorithm, a program was created on the NetBeans programming platform using the Java programming language. The developed program was run on a personal computer with i7 2.40GHz processor. With the help of the program, we have investigated the time needed to obtain a unique solution. The obtained results for some values were given in Table 3.

Various studies have been done to solve this problem. One of the best known of these studies was done by Sosic and Gu [22]. Sosic and Gu stated that they could find a solution less than 55 seconds using a workstation for 3,000,000 queens in their recent work. The developed method in this study is able to produce at least one unique solution for all values as in the study of Sosic and Gu [22]. In addition, as shown in Table 3, our algorithm developed in this study is able to produce solutions in less than 1 second at much larger values of 3,000,000.

Table 3. Test results of proposed algorithm

Number of queens n	**Time to find a solution (seconds.miliseconds)**
4,000,000	0.15
5,000,000	0.29
6,000,000	0.32
7,000,000	0.28
8,000,000	0.54
9,000,000	0.28
10,000,000	0.28

In a study conducted in the following years, El-Qawasmeh and Al-Noubani [23] presented an algorithm that can generate a solution 100 times faster than Sosic's algorithm. Queens' positions are predetermined in the study and there is no need for any calculation or searching operation. In this way, the researchers have achieved speed increase. However, their algorithm is able to produce a correct solution of 75% rather than the full value of *n*. They used Sosic's algorithm in 25% values that they cannot produce solution. Our developed algorithm predetermines queens' positions similar to the algorithm of El-Qawasmeh and Al-Noubani, and does not require search operation or any computation. Moreover, our algorithm is able to produce a solution of all *n* values (100%). In addition, in most values (about 83%) can produce more than one unique solution. Even in some values (about 33%), (*n*-1)/2 unique solutions can be produced.

4. CONCLUSION

N-queens problem is an important problem with its applications in various fields and subject to new studies. However, it is also a difficult problem due to the time complexity that grows exponentially. It is known that backtracking algorithms, one of the most frequently used algorithms to solve this problem, cannot produce all solutions in large *n* values due to this time complexity problem. It is not possible to get all possible solutions for large values. For this reason, the development of methods that can produce one or more solutions but not all for large values has become important.

In this study, a linear time pattern based algorithm was proposed for the n-queens problem. The developed algorithm produce at least one unique solution for all *n* values and it produces solution(s) quite faster with ***O(n)*** time complexity. Besides, for most values (about 67%) it can produce more than one solution. Moreover, for some values (about 33%) it can produce (*n*-1)/2 unique solutions. The developed method provides an important contribution in terms of producing solutions for large values in linear time. In addition, for many values it can produce solutions without using complex calculations or searching. In this respect, the proposed method provides efficiency.

DECLARATION OF ETHICAL STANDARDS

The authors of this article declare that the materials and methods used in this study do not require ethical committee permission and/or legal-special permission.

AUTHORS' CONTRIBUTIONS

Bergen KARABULUT: Developed the theoretical framework, designed the algorithm, performed the experiments and analyse the results.

Atilla ERGÜZEN: Developed the theoretical framework, designed the algorithm, performed the experiments and analyse the results.

Halil Murat ÜNVER: Developed the theoretical framework, wrote the manuscript.

CONFLICT OF INTEREST

There is no conflict of interest in this study.

REFERENCES

[1] Abramson B., and Yung M.M., "Construction through decomposition: A linear time algorithm for the n-queens problem", ***Colombia University Computer Science Technical Reports***, CUCS-129-84, (1984).

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

[3] Waqas M., and Bhatti A.A., "Optimization of N+ 1 Queens Problem Using Discrete Neural Network", ***Neural Network World***, 27(3): 295, (2017).

[4] Wang C.N., Yang S.W., Liu C.M., and Chiang, T., "A hierarchical decimation lattice based on N-queen with an application for motion estimation", ***IEEE signal processing letters***, 10(8): 228-231, (2003).

[5] Bell J., and Stevens B., "A survey of known results and research areas for n-queens", ***Discrete Mathematics***, 309(1): 1-31, (2009).

[6] Murali G., Naureen S., Reddy Y.A., Reddy M.S., JNTUA-Pulivendula J.P., and JNTUA-Pulivendula J.P. "Graphical Simulation of N Queens Problem", ***International Journal of Computer Technology and Applications***, 2(6), (2011).

[7] Güldal S., Baugh V., and Allehaibi S., "N-Queens solving algorithm by sets and backtracking", ***In SoutheastCon, IEEE***, 1-8 (2016).

[8] Lijo V.P. and Jose J.T., "Solving N-Queen Problem by Prediction", ***IJCSIT International Journal of Computer Science and Information Technologies***, 6(4): 3844-3848, (2015).

[9] Sosic R. and Gu J., "3,000,000 queens in less than one minute", ***ACM SIGART Bulletin***, 2(2): 22-24, (1991).

[10] Matsuda S., "Theoretical characterizations of possibilities and impossibilities of Hopfield neural networks in solving combinatorial optimization problems", ***In Neural Networks, 1994. IEEE World Congress on Computational Intelligence***, ***1994 IEEE International Conference***, (7): 4563-4566, (1994).

[11] Sosic R. and Gu, J., "Fast search algorithms for the n-queens problem", ***IEEE Transactions on Systems, Man, and Cybernetics***, 21(6): 1572-1576, (1991).

[12] Hu X., Eberhart R.C. and Shi Y., "Swarm intelligence for permutation optimization: a case study of n-queens problem", ***In Swarm intelligence symposium, 2003, SIS'03, Proceedings of the 2003 IEEE***, 243-246, (2003).

[13] Meng F. and Wu S., "Research of hybrid genetic algorithm in n-queen problem based on HCI", ***In Intelligent Information Technology Application, 2008. IITA'08. Second International Symposium on, IEEE,*** (2): 3-7, (2008).

[14] Khan S., Bilal M., Sharif M., Sajid M. and Baig R., "Solution of n-queen problem using aco", ***In Multitopic Conference, 2009. INMIC 2009. IEEE 13th International***, 1-5, (2009).

[15] Turky A. and Ahmad A., "Using Genetic Algorithm for Solving N -Queens Problem", ***In 2010 International Symposium on Information Technology***, IEEE, Kuala Lumpur, 745-747, (2010).

[16] Motameni H., Bozorgi S.H., Nezhad M.A.S., Berenjian G. and Barzegar B., "Solving N-Queen problem using Gravitational Search Algorithm", ***Life Science Journal-Acta Zhengzhou University Overseas Edition***, 10(1): 37-44, (2013).

[17] Maazallahi R., Niknafs A. and Arabkhedri P.A., "Polynomial-time DNA computing solution for the n-queens problem", ***Procedia-Social and Behavioral Sciences***, 83, 622-628, (2013).

[18] Habiboghli A. and Jalali T., "A Solution to the N-Queens Problem Using Biogeography-Based Optimization", ***IJIMAI***, 4(4): 20-26, (2017).

[19] Amooshahi A., Joudaki M., Imani M. and Mazhari N., "Presenting a new method based on cooperative PSO to solve permutation problems: A case study of n-queen problem", ***In Electronics Computer Technology (ICECT)***, 2011 3rd International Conference on, IEEE, (4): 218-222, (2011).

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

[21] Sosic R. and Gu J., "Efficient local search with conflict minimization: A case study of the n-queens problem", ***IEEE Transactions on Knowledge and Data Engineering***, 6(5): 661-668, (1994).

[22] Lijo V. and Jose J.T., "Solving N-Queen Problem by Prediction" ***Int. J. Comput. Sci. Inf. Technol***, (6), 3844-3848, (2015).

[23] El-Qawasmeh E. and Al-Noubani K., "Reducing the Time Complexity of the N-Queens Problem", ***International Journal on Artificial Intelligence Tools***, 14(03):545-557, (2005).

[24] Rohith S., Gupta A. and Pramodh S., "A Novel Method for Solving N-Queens Problem", ***Int. J. Adv. Res. Comput. Sci. Softw. Eng.***, 3(10), (2013).