International Journal on
Advanced Science
Engineering
Information Technology

Vol.11 (2021) No. 5
ISSN: 2088-5334

Development of an Application for Interactive Research and Analysis of the N-Queens Problem

Velin Kralev [a,*], Radoslava Kraleva [a], Dimitar Chakalov [a]

[a] *Department of Informatics, South-West University "Neofit Rilski", Blagoevgrad, 2700, Bulgaria*
*Corresponding author: * velin_kralev@swu.bg*

Abstract*— This paper presents a study on the N-Queens Problem. Different approaches to its solution discussed in the scientific literature are analyzed. The implementation of an algorithm based on the backtracking method is also presented. The algorithm is optimized to find solutions in a specific subset of configurations among all possible ones. With this approach, the computational complexity of the algorithm is reduced from exponential to quadratic. In this way, the algorithm finds all possible solutions in a shorter time: fundamental and their symmetrical equivalents. The methodology for conducting the experiments is presented. The purpose of the study, the tasks to be performed, and the conditions for conducting the experiments are presented as well. In connection with the research, an application that implements the presented algorithm has been developed. This application generated all the results obtained in this study. The experimental results show that with a linear increase in the number of queens (equivalent to a quadratic increase in the number of fields on the board, the number of recursive calls made by the algorithm increases exponentially. Similarly, the number of possible solutions, as well as the execution time of the algorithm (in the different modes of the application - internal, interactive, and combined), also increases exponentially. However, the algorithm's execution time in the internal mode is significantly shorter than in the other two modes - interactive and combined. The future guidelines for the study are presented.

***Keywords*— N-queens problem; backtracking algorithm; decision problem; software development; application programs.**

Manuscript received 19 Feb. 2021; revised 21 May 2021; accepted 10 Jun. 2021. Date of publication 31 Oct. 2021.

I. INTRODUCTION

The N-Queens problem (NQP) is a classical, combinatorial optimization problem that has been actively studied in the recent years [1]–[4]. This problem was originally known as Eight Queen Problem (EQP). The size of a standard chessboard is a square measuring 8*8 squares. In solving this problem, the case in which the size of the board is greater than or equal to 4*4 squares is usually considered. It is easy to check that if the size of the board is less than or equal to 3*3 the problem has no solution.

A condition of the task for chess queens is to place Q (or N) queens on a square board so that none of them attacks any of the others. In other words, the goal is to place the pieces on the board so that they do not endanger each other (which in turn means that there can be at most one queen on each horizontal, vertical, and diagonal). The large number of scientific publications related to the research and analysis of this problem leads to the use of already known benchmark solutions in another scientific research [5].

There is a wide variety of possible methods for solving NQP and NQP's different formulations [6], [7]. Many of these methods are discussed in the scientific literature, comparing different approaches and publishing improved versions of existing algorithms [8], [9]. Published results show that genetic algorithms [10] and their modifications based on improved genetic operations [11] generate a large part of all possible solutions at a predetermined board size. In other scientific publications, a comparative analysis is made between different approaches and algorithms comparing the number of generated solutions and the time for their generation [12]. Some efficient metaheuristic approaches [13], [14] show that they can find solutions for NQP in a very short time. These approaches are based on combining evolutionary algorithms with different heuristic techniques. The same heuristic approaches are successfully used in solving many classes of combinatorial optimization problems [15], [16]. These studies show that combined approaches, in some cases, have significantly better computational times than basic heuristic techniques. Approaches based on neural networks Montoro *et al.* [17] and Lapushkin [18] are also used in solving NQP.

Other studies show that the N-Queens problem can be successfully solved by other techniques, such as simulated annealing [19], methods based on local search [20], heuristic and meta-heuristic approaches [21], [22]. In order to improve the speed of solving NQP, parallel algorithms are being developed, such as those based on multicore architecture [23], those based on parallel computing [24], and others based on specific computational models [25]. To increase productivity, methods of using the hardware capabilities of the computer systems on which the respective algorithms are executed are used [26], [27]. Also, methods based on accelerated execution in solving the N-Queens problem by using the ability for communication between the cores and threads of the CPU [28] are used. Software products that implement the NQP task in the form of a game (puzzle) have been developed. One such application used in the field of education is presented in [29].

Fig. 1 shows two (asymmetric) solutions of the N-Queens problem at a board size of 9 x 9. With this board size all possible solutions are 352 of which only 46 are unique (i.e., asymmetric).

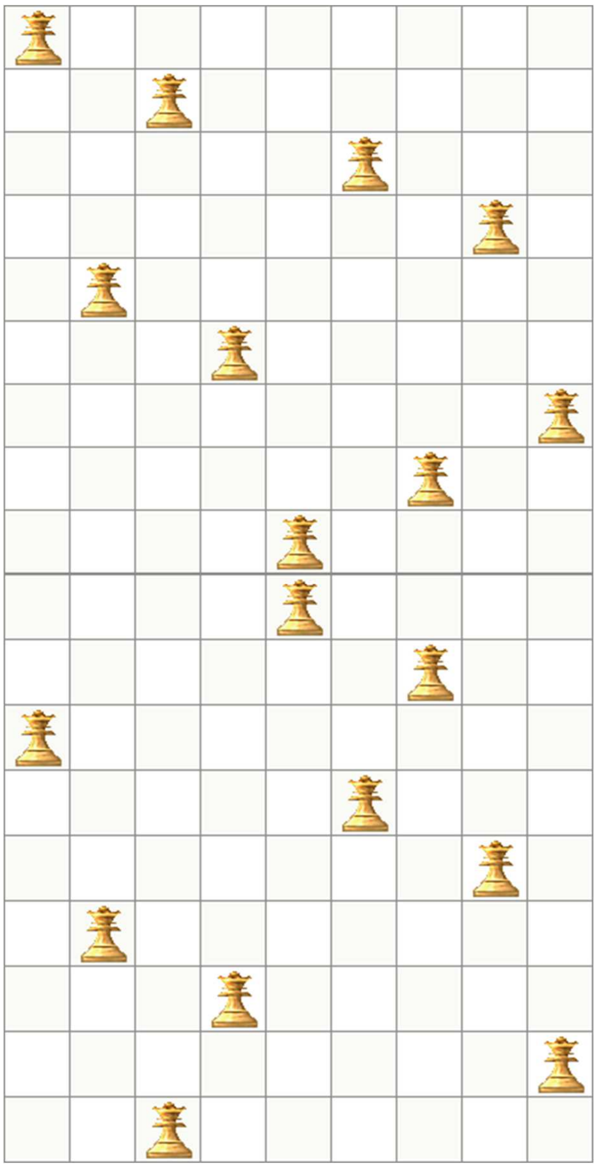


Fig. 1 Two solutions of the NQP with a board size of 9 x 9

The present study will use an approach to generate all solutions by examining only a certain subset among all possible candidates for a solution. When using this approach, it is very important to apply a method that reduces the number of analyzed individual cases of the problem. This can be done by omitting entire branches from the recursion tree, for which it is certain at an earlier stage that they will not lead to an acceptable solution. The idea of the algorithm that will be presented in the next section is just such one.

II. Material and Method

In this section, an implementation of a recursive algorithm for solving the NQP will be presented. This algorithm uses the backtracking approach. Initially, a variable named QueenCount is initialized with a value of 1. The next step is to check if QueenCount queens are successfully placed on the board. If this is true, a test is made to place the next queen on the board (i.e., QueenCount + 1). However, the next queen must be placed in such a square on the board that it does not fall into a common horizontal, vertical or diagonal with the queens already located on the board (which are exactly QueenCount in number). This is the step forward of the algorithm. If at the last (successful) placement of a queen on the board it turns out that the placed queens are exactly N (i.e., QeueenCount = N), then this means that an acceptable solution has been found. However, if there is no free position for the queen with the QueenCount number, then the algorithm performs a step back (backtracking), after which it starts looking for another suitable square for placing the queen with the number (QueenCount - 1).

Both local and global data structures (variables and arrays) can be used when performing an algorithm. The algorithm manipulates these structures (stored in the computer's RAM) during its execution. Global data structures are accessible from all functions (methods) of the application. Fig. 2 shows the global declarations of the variables and dynamic arrays that the NQP solution algorithm uses.

```
const
  Size = 8;
var
  Start, Finish: Cardinal;
  QueenCount: Integer;
  SolutionCount: UInt64;
  RecursionCount: UInt64;
  RowsArray: array[1..Size] of Byte;
  ColsArray: array[1..Size] of Byte;
  RightDiagonal: array[1..2 * Size-1] of Byte;
  LeftDiagonal: array[1..2 * Size] of Byte;
  Fields: array[1..Size, 1..Size] of Byte;
  Solution: array[1..Size, 1..Size] of Byte;
```
Fig. 2 Source-code of the global declarations

The size constant (of Integer type) determines the size of the board and the number of queens to be placed on the board. The variables Start and Finish (of Cardinal type) are used to record the time to find a solution (i.e., how long the algorithm has been running). The QueenCount variable (of Integer type) stores the current number of the queens placed on the board. The SolutionCount variable (of UInt64 type) can only accept positive values and store the current number of solutions found. The RecursionCount variable (also of UInt64 type) stores the current number of recursive calls that the algorithm has made. The 1-dimensional arrays RowsArray, ColsArray, RightDiagonal and LeftDiagonal (of Byte type) store the indexes of the fields on which queens are placed (respectively in row, column and both diagonals). The 2-dimensional Fields array (of Byte type) stores information about the "attacked" fields on the board. The values used are as follows: 1 - the field is "attacked", 0 - the field is not "attacked". The Solution array (also of type Byte) stores a partial or complete solution of the problem, as the values used are as follows 1 - there is a queen placed in this field and 0 otherwise.

Fig. 3 shows the source-code of the DisableFields method, which is used to mark as disabled all "attacked" fields on the board by the placed queens.

```
01 procedure DisableFields(ACol, ARow: Integer);
02 var
03   Col, Row, DCol, DRow: Integer;
04 begin
05   Solution[ACol, ARow] := 1;
06   for Col := 1 to Size do
07     if (Fields[Col, ARow] <> 1) then
08         Fields[Col, ARow] := 1;
09   for Row := 1 to Size do
10     if (Fields[ACol, Row] <> 1) then
11         Fields[ACol, Row] := 1;
12   DCol := ACol - Size; DRow := ARow - Size;
13   repeat
14     Inc(DCol); Inc(DRow);
15     if ((DCol >= 1) and (DRow >= 1) and
16         (DCol <= Size) and (DRow <= Size)) then
17       if (Fields[DCol, DRow] <> 1) then
18           Fields[DCol, DRow] := 1;
19   until ((DCol >= Size) or (DRow >= Size));
20   DCol := ACol + Size; DRow := ARow - Size;
21   repeat
22     Dec(DCol); Inc(DRow);
23     if ((DCol >= 1) and (DRow >= 1) and
24         (DCol <= Size) and (DRow <= Size)) then
25       if (Fields[DCol, DRow] <> 1) then
26           Fields[DCol, DRow] := 1;
27   until ((DCol <= 1) or (DRow >= Size));
28 end;
```

Fig. 3 Source-code of the DisableFields method

The DisableFields method receives the variables ACol and ARow as input parameters. These parameters contain the position (column and row) on which the next queen of the solution is placed. In the declarative part of the DisableFields method, the local variables Col, Row, DCol and DRow (of Integer type) are declared (line 03). The first code that the DisableFields method executes is to fix in the two-dimensional array the solution that a queen is already placed at the ACol, ARow position (line 05). The next code that executes the DisableFields method is to disable (i.e. mark as "attacked") all non-disabled fields from column ACol (lines 06-08), all non-disabled fields from row ARow (lines 09-11), all non-disabled fields from one of the left diagonals, which contains the field ACol, ARow (lines 12-19) and all non-disabled fields from one of the right diagonals, which also contains the field ACol, ARow (lines 20-27).

An inefficient approach to solving NQP is to test all possible combinations of queen's placements on the board. Each time the required number of queens is positioned on the board, it is checked whether they all fulfill the condition of the task and if this is not fulfilled, the specific combination is rejected. However, this approach can be optimized. Since there can be at most one queen on each row and column of the board, it is sufficient to check only those combinations in which there is exactly one queen on each row and column. The algorithm can be improved by checking only those fields on the board that are not "attacked" by already placed queens. The implementation of the algorithm follows.

Fig. 4 shows the source-code (a) and the flowchart (b) of the FindSolutions recursive method. This method checks if it is possible to position the next queen on any of the free fields on the board. The number of the next queen is passed as an input parameter of the method - this is the variable QNumber.

```
procedure FindSolutions(QNumber: Byte);
begin
  Inc(RecursionCount);
  if QNumber > Size then begin
    QueenCount := Size;
    Inc(SolutionCount);
    for var Col: Integer := 1 to Size do
      for var Row: Integer := 1 to Size do
        if (RowsArray[Row] = Col) then
            DisableFields(Col, Row)
         else Solution[Col, Row] := 0;
    Exit;
  end;
  for var I: Integer := 1 to Size do begin
    if ((ColsArray[I] <> 0) and
     (RightDiagonal[QNumber + I] <> 0) and
     (LeftDiagonal[Size+QNumber-I] <> 0)) then
    begin
      ColsArray[I] := 0;
      RightDiagonal[QNumber + I] := 0;
      LeftDiagonal[Size+QNumber - I] := 0;
      RowsArray[QNumber] := I;
      FindSolutions(QNumber + 1);
      ColsArray[I] := 1;
      RightDiagonal[QNumber + I] := 1;
      LeftDiagonal[Size + QNumber - I] := 1;
    end;
  end;
end;
```

Fig. 4 (a) Source-code of the recursion-based method FindSolutions

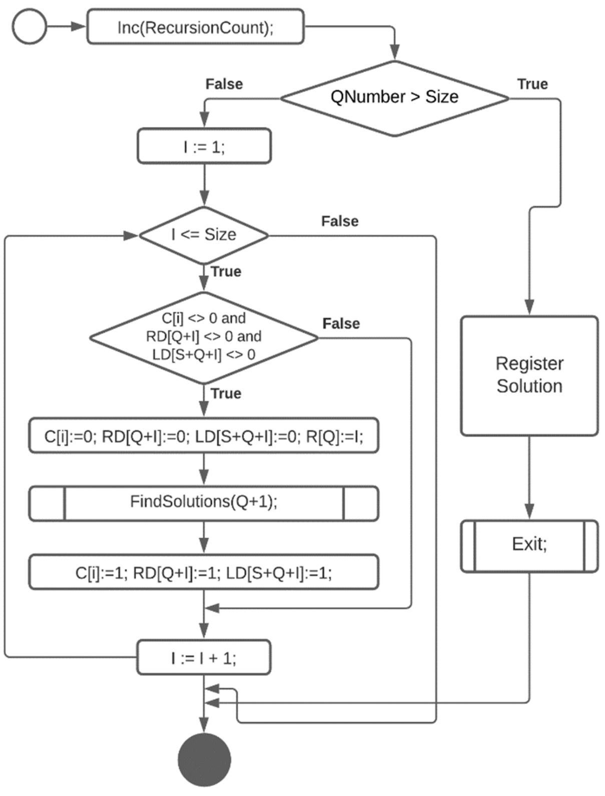


Fig. 4 (b) Flowchart of the recursion-based method FindSolutions

The number of recursive calls to FindSolutions is stored in the global variable RecursionCount. The first code that executes the FindSolutions method is to increase by 1 the value of the variable RecursionCount (line 03). Then, the method checks whether the required number of queens has already been placed on the board (line 04). If this is done, the number of solutions found is increased by 1 (line - 06) and the two-dimensional arrays Fields and Solutions are filled (lines 07 - 11). The two-dimensional Fields array stores information about the "attacked" fields, and the two-dimensional Solution array stores information about the current solution. Then, the execution of the method is terminated (line 12). In this interruption, the control of the computational process is actually transferred from the last call of the FindSolution method (stored in the stack) to the previous one. In this case, if there are more solutions to find, the recursive process will continue to find them. If the exact number of queens has not yet been placed on the board, the FindSolution method initializes a loop (lines 14 – 28). For each value (from 1 to size) of the local variable I checks whether there is no queen in the column with index I or in some of the two diagonals. If a free position is found, a queen is placed at that position and the FindSolution method (line 23) is called recursively. On this call, the FindSolution method receives as a parameter the number of the next queen (QNumber + 1). However, if no free position can be found for the current queen, the algorithm takes a step back (backtracking). In this way, the algorithm starts a new search process for another free position of the previous queen (i.e. the one with number QNumber - 1). This process is repeated until all possible solutions are generated.

The InitializeAndStart method starts the NQP solution search process (Fig. 5).

```
01 procedure InitializeAndStart;
02 var
03   I: Integer; Msg: String;
04 begin
05   QueenCount:=0; SolutionCount:=0;
06   RecursionCount:=0;
07   for I:=1 to Size do ColsArray[I]:=1;
08   for I:=1 to 2*Size-1 do RightDiagonal[I]:=1;
09   for I:=1 to 2*Size do LeftDiagonal[I]:=1;
10   Start := GetTickCount;
11   FindSolutions(1);
12   Finish := GetTickCount;
13   Msg := IntToStr(SolutionCount) + ', ' +
14          IntToStr(RecursionCount) + ', ' +
15          IntToStr(Finish-Start) + ' ms';
16 end;
```

Fig. 5 Source-code of the InitializeAndStart method

In the declarative part of the InitializeAndStart method, two local variables are declared - I and Msg (line 03). The integer variable I is used as the control variable for the loops initializing the arrays ColsArray, RightDiagonal and LeftDiagonal (lines 07 - 09). This method also initializes the variables QueenCount, SolutionCount, and RecursionCount by setting each of these variables to 0 (lines 05 - 06). Before calling the FindSolutions method, the Start variable stores the time to start the NQP solution search process (line 10). Once the FindSolution method has finished, the Finish variable stores the end time of the calculation process. The time is obtained from the GetTickCount function. This function returns the time from the start of the computer's operating system (in milliseconds). The difference between the values of the variables Finish and Start is actually the execution time of the FindSolution method (in milliseconds). Finally, the InitializeAndStart method concatenates in the string variable Msg the number of generated solutions (the SolutionCount variable), the number of recursive calls (the RecursionCount variable), and the execution time of the whole process.

The complexity of the algorithm, after optimization, remains exponential and is respectively O(n!).

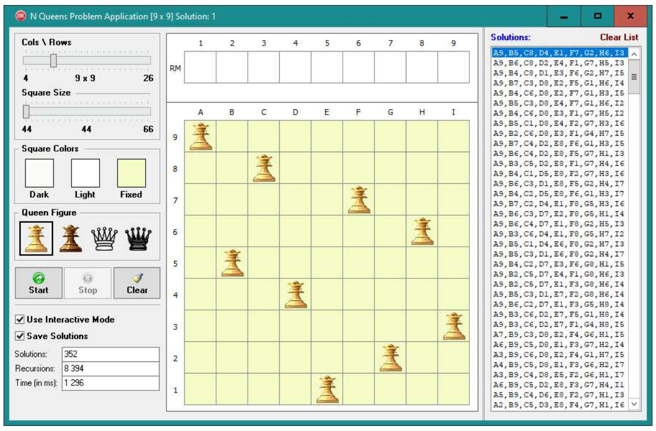
III. Result and Discussion

A. Development of an Application for Conducting Experiments

There are many programming languages and application development environments. Some of them provide the possibility to compile the same code for different operating systems. These integrated development environments enable the application's interface to be visually designed. However, the various functions of this application are implemented with event-oriented programming. In this way, different types of applications (such as mobile, desktop, services, etc.) can be developed in a short time. These applications can perform various tasks (data analysis, data processing, graphic design, etc.). In addition, these applications can be run on different target platforms (operating systems and servers). These capabilities of application development environments are referred to as Cross-Platform Application Development [30]–[32] or Multi-Device Application Development [33].

Application development is usually done in two stages - application design and programming. During the application design stage, the application's graphical user interface (GUI) is created. This is done by positioning (arranging) controls on the forms (also called windows) of the application [34]. Each form of an application can be considered as a container for controls. During the application design stage, the developer (designer) creates the layout of the application. During the programming stage, the developer creates the functionality of the application. This is done by implementing user functions with a specific purpose or by implementing event handlers. Event handlers are functions that are called by the application when certain events occur. The application receives the event as a message sent by the operating system.

For the purposes of the study, an application was developed to perform the planned experiments. The N Queens Problem Application is shown in Fig. 6.


N Queens Problem Application [9 x 9] Solution: 1

Cols \ Rows: 9 x 9
Square Size: 44

[✓] Use Interactive Mode
[✓] Save Solutions

Solutions: 352
Recursions: 8 394
Time (in ms): 1 296

Solutions:

```
A9,B5,C8,D4,E1,F7,G2,H6,I3
A9,B6,C8,D2,E4,F1,G7,H5,I3
A9,B4,C8,D1,E3,F6,G2,H7,I5
A9,B7,C3,D8,E2,F5,G1,H6,I4
A9,B4,C6,D8,E2,F7,G1,H3,I5
A9,B5,C3,D8,E4,F7,G1,H6,I2
A9,B4,C6,D8,E3,F1,G7,H5,I2
A9,B5,C1,D8,E4,F2,G7,H3,I6
A9,B2,C6,D8,E3,F1,G4,H7,I5
A9,B7,C4,D2,E8,F6,G1,H3,I5
A9,B6,C4,D2,E8,F5,G7,H1,I3
A9,B3,C5,D2,E8,F1,G7,H4,I6
A9,B4,C1,D5,E8,F2,G7,H3,I6
A9,B6,C3,D1,E8,F5,G2,H4,I7
A9,B4,C2,D5,E8,F6,G1,H3,I7
A9,B7,C2,D4,E1,F8,G5,H3,I6
A9,B6,C3,D7,E2,F8,G5,H1,I4
A9,B6,C4,D7,E1,F8,G2,H5,I3
A9,B3,C6,D4,E1,F8,G5,H7,I2
A9,B5,C1,D4,E6,F8,G2,H7,I3
A9,B5,C3,D1,E6,F8,G2,H4,I7
A9,B4,C2,D7,E3,F6,G8,H1,I5
A9,B2,C5,D7,E4,F1,G8,H6,I3
A9,B2,C5,D7,E1,F3,G8,H6,I4
A9,B5,C3,D1,E7,F2,G8,H6,I4
A9,B6,C2,D7,E1,F3,G5,H8,I4
A9,B3,C6,D2,E7,F5,G1,H8,I4
A9,B3,C6,D2,E7,F1,G4,H8,I5
A7,B9,C3,D8,E2,F4,G6,H1,I5
A6,B9,C5,D8,E1,F3,G7,H2,I4
A3,B9,C6,D8,E2,F4,G1,H7,I5
A4,B9,C5,D8,E1,F3,G6,H2,I7
A3,B9,C4,D8,E5,F2,G6,H1,I7
A6,B9,C5,D2,E8,F3,G7,H4,I1
A5,B9,C4,D6,E8,F2,G7,H1,I3
A2,B9,C5,D3,E8,F4,G7,H1,I6
```

Fig. 6 Session of work whit the N Queens Problem Application

The developed application provides some important functions. In the [Cols \ Rows] section, the board size can be set, with possible values ranging from 4 to 26. In the [Square Size] section, the field size of the board can be set, with possible values ranging from 44 to 66. In the [Square Colors] section, different colors can be selected for the "dark", "light" and "attacked" fields on the board. In the [Queen Figure] section, the image for the queens can be selected. The [Start] and [Stop] buttons start and stop the solution search process, respectively. When the [Use Interactive Mode] checkbox is checked, a new solution will be displayed immediately. When the [Save Solutions] checkbox is checked, the new solution will be saved (added) in the [Solutions] list. The text labels "Solutions", "Recursions" and "Time (in ms)" show the number of solutions found, the number of recursive calls and the elapsed time for finding all solutions, respectively. When a solution is selected from the "Solutions" list, the application immediately visualizes it.

B. Experimental results

The N Queens Problem Application generated the experimental results. The application was run on a computer with 32-bit Windows 10 Pro operating system and hardware configuration as follows: processor: Intel(c) Core i5-10400 (2.9/4.3 GHz, 12M); RAM memory: 8 GB DDR4 2666 MHz.

For conducting the experiments, 14 (fourteen) sizes of boards were selected, respectively from the standard size 8x8 to 21x21. Table 1 shows summary information for the input data.

TABLE I
SUMMARY INFORMATION FOR THE INPUT DATA

No	Board Size	Fields Count	Fundamental Solutions (F)	All Solutions (A)	Ratio F/A	Ratio F/(F-1)
1	8 x 8	64	12	92	0.130	2.0000
2	9 x 9	81	46	352	0.131	3.8333
3	10 x 10	100	92	724	0.127	2.0000
4	11 x 11	121	341	2680	0.127	3.7065
5	12 x 12	144	1787	14200	0.126	5.2405
6	13 x 13	169	9233	73712	0.125	5.1668
7	14 x 14	196	45752	365596	0.125	4.9553
8	15 x 15	225	285053	2279184	0.125	6.2304
9	16 x 16	256	1846955	14772512	0.125	6.4793
10	17 x 17	289	11977939	95815104	0.125	6.4852
11	18 x 18	324	83263591	666090624	0.125	6.9514
12	19 x 19	361	621012754	4968057848	0.125	7.4584
13	20 x 20	400	4878666808	39029188884	0.125	7.8560
14	21 x 21	441	39333324973	314666222712	0.125	8.0623

The "Ratio (F/A)" column shows the ratio between the number of fundamental solutions and the number of all solutions. It can be seen that after a board size of 12x12, the values of this ratio are 12.5%. This ratio shows that the number of fundamental (asymmetric) solutions represents 12.5% of the number of all possible solutions. Also, this value shows that the number of symmetric solutions is 8 times greater than the number of fundamental solutions. This can be calculated from the reciprocal value A/F. This dependence is a consequence of the fact that there are exactly 8 ways to turn and/or rotate a square symmetrically. Two ways horizontally, two ways vertically; two ways on the main diagonal and two ways on the secondary diagonal. The "Ratio (F/(F–1))" column shows the increase in the number of fundamental solutions relative to the number of fields on the board (respectively the size of the board). It can be seen that these values increase linearly in contrast to the number of fundamental solutions, which increase exponentially.

The purpose of this study is to analyze the three different modes - internal, interactive and combined, in terms of the application execution time. Table 2 shows the results of the experiments. The values in the "Internal", "Interactive" and "Combined" columns are arithmetic mean of four different application runs (for each of the modes).

TABLE IIIII
RESULTS OF THE EXPERIMENTS PERFORMED

Fields Count	All Solutions	Recursion Calls	Time Internal	Time Interactive	Time Combined
64	92	2057	0.02 s	0.24 s	0.38 s
81	352	8394	0.02 s	1.02 s	1.45 s
100	724	35539	0.03 s	2.66 s	3.22 s
121	2680	166926	0.14 s	10.17 s	13.08 s
144	14200	856189	0.75 s	55.35 s	1 min. 46 s
169	73712	4674890	4.08 s	7 min. 19 s	19 min. 54 s
196	365596	27358553	23.84 s	43 min. 20 s	2 h. 1 min
225	2279184	171129072	2 min. 37 s	5 h. 19 min	15 h. 27 min
256	14772512	1141190303	20 min. 14 s	17 h. 48 min	> 3 days
289	95815104	7473578112	2 h. 43 min	N/A	N/A

For each of the input sizes, four tests were made in the three different modes of application - internal, interactive and combined. In the internal mode, the application does not visualize and does not save the found solutions. Both the "Use Interactive Mode" and "Save Solutions" checkboxes are unchecked in this mode. The "Use Interactive Mode" checkbox is checked in the application's interactive mode, but the "Save Solutions" checkbox is unchecked. In this mode, when a solution is found, it is immediately visualized. Both checkboxes - "Use Interactive Mode" and "Save Solutions" are checked in combined mode. In this way, when a solution is found, it is immediately visualized and stored into the "Solutions" list.

Fig. 7 shows the influence of the number of fields (x-axis) on the number of all solutions (y-axis).

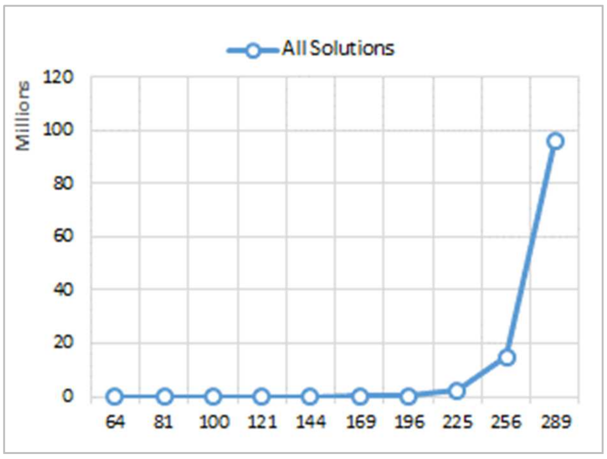


Fig. 7 Influence of the number of fields on the number of all solutions

Fig. 8 shows the influence of the number of fields (x-axis) on the number of recursive calls (y-axis).

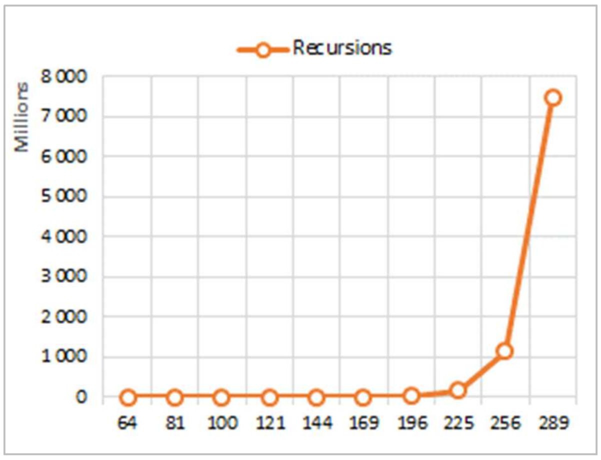


Fig. 8 Influence of the number of fields on the number of recursive calls

From the charts of Fig. 7 and Fig. 8, with a linear increase in the size of the board (respectively the number of fields on the board), the number of all solutions and the number of recursive calls increase exponentially. The ratio between the number of recursive calls and the number of all solutions changes from 22 times (for an 8 x 8 board) to 78 times (for a 17 x 17 board). Fig. 9 shows the influence of the number of fields (x-axis) on the algorithm's execution time (y-axis) in the internal mode.

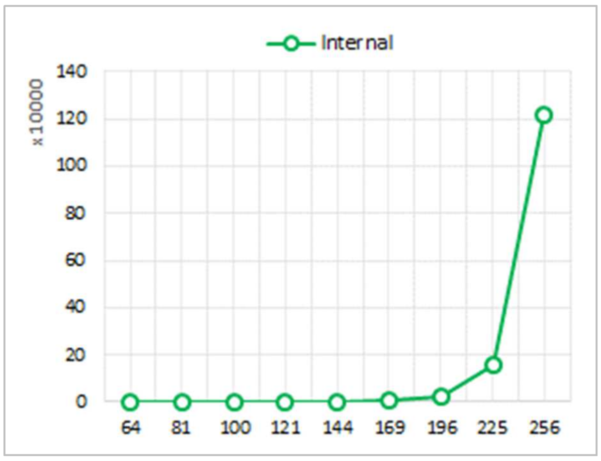


Fig. 9 Influence of the number of fields on the execution time of the algorithm in the internal mode

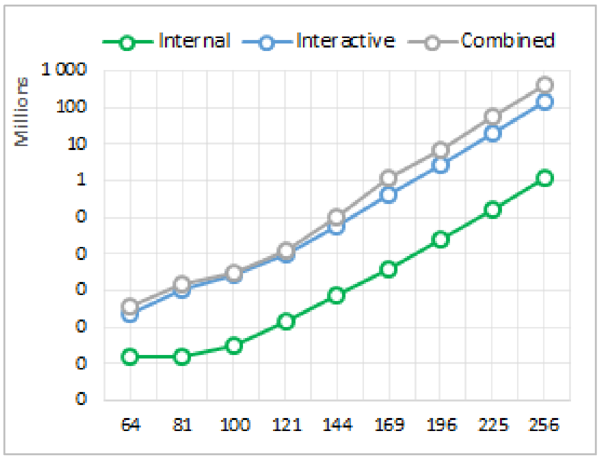


Fig. 10 Influence of the number of fields on the algorithm's execution time, summarized in the three modes.

Fig. 10 shows the influence of the number of fields (x-axis) on the execution time (y-axis) summarized in the three modes. From the charts of Fig. 9 and Fig. 10, with a linear increase in the number of fields on the board, the execution time on the algorithm in the internal mode increases exponentially. It is important to specify that the values on the y scale in Fig. 10 are on a logarithmic scale. This chart shows that the combined mode and the interactive mode are comparable in terms of the algorithm's execution time. In contrast, the internal mode is significantly faster, with differences in values of the order of hundreds of times. Therefore, the internal mode should be preferred in future studies of the N-Queens problem.

IV. Conclusion

In this paper, a study of the N-Queens Problem was presented. Different approaches to its solution, which are discussed in detail in the scientific literature, were analyzed. The implementation of an algorithm based on the backtracking method was also presented. The algorithm was optimized to find solutions in a specific subset of configurations among all possible ones. With this approach, the computational complexity of the algorithm was reduced from exponential to quadratic. In this way, the algorithm finds in a shorter time all possible solutions, both fundamental and their symmetrical equivalents. The definitions of the various global variables and the dynamic data structures - vectors (arrays) and matrices (two-dimensional arrays) that the algorithm uses were also described. The source codes of the implemented methods (procedures and functions) were presented and analyzed. The method for measuring the execution time of the algorithm used by the start-up procedure takes into account the multitasking mode of the operating system.

The methodology for conducting the experiments was presented. The purpose of the study, the tasks to be performed and the conditions for conducting the experiments were presented as well. As part of the methodology, 14 board sizes were presented, from 8 x 8 (standard chessboard) to 21 x 21, respectively. The ratios between the number of fundamental solutions and the number of all solutions for each of the selected board sizes were calculated. For all solutions, the incremental step between every two consecutive values was calculated. In connection with the research, an application was developed that implements the presented algorithm. Its main functions were summarized. All results obtained in this study were generated by this application.

The experimental results showed that with a linear increase in the number of queens (equivalent to a quadratic increase in the number of fields on the board), the number of recursive calls made by the algorithm increased exponentially. Similarly, the number of possible solutions, as well as the execution time of the algorithm (in the different modes of the application - internal, interactive and combined), also increased exponentially. However, the execution time of the algorithm in the internal mode was significantly shorter than in the other two modes - interactive and combined. The ratio between the number of recursive calls and the number of all solutions was also calculated. This ratio varied between 22 times (with a board size of 8 x 8) to 78 times with a board size of 17 x 17.

ACKNOWLEDGMENT

This work was supported by the South-West University "Neofit Rilski" in Bulgaria.

REFERENCES

[1] K. Pratt, "Closed-form expressions for the n-queens problem and related problems," *International Mathematics Research Notices*, vol. 2019, no. 4, pp. 1098-1107, Feb. 2019, 10.1093/imrn/rnx119.

[2] K. C. Buño, F. G. C. Cabarle, M. D. Calabia, and H. N. Adorna, "Solving the N-Queens problem using dP systems with active membranes," *Theoretical Computer Science*, vol. 736, pp. 1-14, Aug. 2018, 10.1016/j.tcs.2017.12.013.

[3] M. A. Ayub, K. A. Kalpoma, H. T. Proma, S. M. Kabir, and R. I. H. Chowdhury, "Exhaustive study of essential constraint satisfaction problem techniques based on N-Queens problem," *in Proc. 20th International Conference of Computer and Information Technology, ICCIT 2017*, Dhaka, Bangladesh, 2018, pp. 1–6.

[4] M. Plauth, F. Feinbube, F. Schlegel, and A. Polze, "Using Dynamic Parallelism for Fine-Grained, Irregular Workloads: A Case Study of the N-Queens Problem," *in Proc. 2015 3rd International Symposium on Computing and Networking, CANDAR 2015*, Hokkaido, Japan, 2016, art. no. 7424747, pp. 404-407, 10.1109/CANDAR.2015.26.

[5] A. F. J. Al-Gburi, S. Naim, A. N. Boraik, "Hybridization of bat and genetic algorithm to solve N-queens problem," *Bulletin of Electrical Engineering and Informatics*, vol. 7, no. 4, pp. 626-632, Dec. 2018, 10.11591/eei.v7i4.1351.

[6] O. Kolossoski, L. C. Matioli, E. M. R. Torrealba, and J. G. Silva, "Modular knight distance in graphs and applications on the n-queens problem," *Discrete Mathematics*, vol. 343, no. 12, art. no. 112136, Dec. 2020, 10.1016/j.disc.2020.112136.

[7] M. Bača, S. C. López, F. A. Muntaner-Batle, and A. Semaničová-Feňovčíková, "New Constructions for the n-Queens Problem," *Results in Mathematics*, vol. 75, no. 1, art. no. 41, Mar. 2020, 10.1007/s00025-020-1166-9.

[8] A. Alhassan, "Build and conquer: Solving N queens problem using iterative compression," *in Proc. International Conference on Computer, Control, Electrical, and Electronics Engineering 2019, ICCCEEE 2019*, Sudan, 2019, art. no. 9070976.

[9] G. Zheng and Y. Xu, "A Hybrid Chemical Reaction Optimization Algorithm for N-Queens Problem," *Advances in Intelligent Systems and Computing*, vol. 1274 AISC, pp. 128-137, 2021, 10.1007/978-981-15-8462-6_15.

[10] I. A. Humied, "Solving N-Queens problem using subproblems based on genetic algorithm," *IAES International Journal of Artificial Intelligence*, vol. 7, no. 3, pp. 130-137, Sep. 2018, 10.11591/ijai.v7.i3.pp130-137.

[11] V. Jain and J. S. Prasad, "Solving N-queen problem using genetic algorithm by advance mutation operator," *International Journal of Electrical and Computer Engineering*, vol. 8, no. 6, pp. 4519-4523, Dec. 2018, 10.11591/ijece.v8i6.pp.4519-4523.

[12] A. K. Dubey, V. Ellappan, R. Paul, and V. Chopra, "Comparative analysis of backtracking and genetic algorithm in n queen's problem," *International Journal of Pharmacy and Technology*, vol. 8, no. 4, pp. 25618-25623, Dec. 2016.

[13] P. N. Sharief and B. S. Saini, "Metaheuristic techniques on N-Queen problem: DE VS ABC," *International Journal of Applied Engineering Research*, vol. 10, no. 55, pp. 4240-4244, 2015.

[14] E. Masehian, H. Akbaripour, and N. Mohabbati-Kalejahi, "Landscape analysis and efficient metaheuristics for solving the n-queens problem," *Computational Optimization and Applications*, vol. 56, no. 3, pp. 735-764, Dec. 2013, 10.1007/s10589-013-9578-z.

[15] V. Kralev, R. Kraleva, and S. Kumar, "A modified event grouping based algorithm for the university course timetabling problem," *International Journal on Advanced Science, Engineering and Information Technology*, vol. 9, no. 1, pp. 229-235, 2019, 10.18517/ijaseit.9.1.6488.

[16] V. Kralev, "Different applications of the genetic mutation operator for symetric travelling salesman problem," *International Journal on Advanced Science, Engineering and Information Technology*, vol. 8, no. 3, pp. 762-770, 2018, 10.18517/ijaseit.8.3.4867.

[17] F. Arroyo Montoro, S. Gómez-Canaval, K. Jiménez Vega, and A. Ortega De La Puente, "A Linear Time Solution for N-Queens Problem Using Generalized Networks of Evolutionary Polarized Processors," *International Journal of Foundations of Computer Science*, vol. 31, no. 1, pp. 7-21, Jan. 2020, 10.1142/S0129054120400018.

[18] A. A. Lapushkin, "Application of Hopfield neural network to the N-queens problem," *Advances in Intelligent Systems and Computing*, vol. 449, pp. 115-120, 2016, 10.1007/978-3-319-32554-5_15.

[19] V. M. Saffarzadeh, P. Jafarzadeh, and M. Mazloom, "A hybrid approach using particle swarm optimization and simulated annealing for N-queen problem," *World Academy of Science, Engineering and Technology*, vol. 43, pp. 974-978, 2010.

[20] H. Motameni, S. Bozorgi Hossein, M. Ali Shaban Nezhad, G. Berenjian, and B. Barzegar, "Solving N-queen problem using gravitational search algorithm," *Life Science Journal*, vol. 10, no. 1, pp. 37-44, Mar. 2013.

[21] D. Chatham, "The n queens problem with forbidden squares," *Utilitas Mathematica*, vol. 111, pp. 199-210, 2019.

[22] P. Prudhvi Raj, P. Shah, and P. Suresh, "Faster Convergence to N-Queens Problem Using Reinforcement Learning," *Communications in Computer and Information Science*, vol. 1290, pp. 66-77, 2020, 10.1007/978-981-33-6463-9_6.

[23] S. Saxena, N. Sharma, and S. Sharma, "Parallel computing in genetic algorithm (GA) with the parallel solution of n Queen's Problem based on GA in multicore architecture," *International Journal of Applied Engineering Research*, vol. 10, no. 17, pp. 37707-37716, 2015.

[24] C. Jianli, C. Zhikui, W. Yuxin, and G. He, "Parallel genetic algorithm for N-Queens problem based on message passing interface-compute unified device architecture," *Computational Intelligence*, vol. 36, no. 4, pp. 1621-1637, Nov. 2020, 10.1111/coin.12300.

[25] J. Cao, Z. Chen, Y. Wang, and H. Guo, "Parallel Implementations of Candidate Solution Evaluation Algorithm for N-Queens Problem," *complexity*, vol. 2021, art. no. 6694944, 2021, 10.1155/2021/6694944.

[26] Y. Azuma, H. Sakagami, and K. Kise, "An efficient parallel hardware scheme for solving the N-queens problem," *in Proc. 2018 IEEE 12th International Symposium on Embedded Multicore/Many-Core Systems-on-Chip, MCSoC 2018*, Hanoi, Viet Nam, 2018, art. no. 8540208, pp. 16-22.

[27] F. J. De Souza and F. L. De Mello, "N-Queens Problem Resolution Using the Quantum Computing Model," *IEEE Latin America Transactions*, vol. 15, no. 3, art. no. 7867605, pp. 534-540, Mar. 2017, 10.1109/TLA.2017.7867605.

[28] A. Maroosi and R. C. Muniyandi, "Accelerated execution of P systems with active membranes to solve the N-queens problem," *Theoretical Computer Science*, vol. 551, no. C, pp. 39-54, 2014, 10.1016/j.tcs.2014.05.004.

[29] Y. Sasaki, M. Fukui, and T. Hirashima, "Development of iOS software n-queens problem for education and its application for promotion of computational thinking," *in Proc. 2019 IEEE 8th Global Conference on Consumer Electronics, GCCE 2019*, Osaka, Japan, 2019, art. no. 9015331, pp. 563-565.

[30] K. Vassallo, L. Garg, V. Prakash, and K. Ramesh, "Contemporary technologies and methods for cross-platform application development," *Journal of Computational and Theoretical Nanoscience*, vol. 16, pp. 3854-3859, 2019, 10.1166/jctn.2019.8261.

[31] M. Cuadros, A. De la Fuente, R. Villalta, and A. Barrientos, "Cross-platform enterprise application development framework for large screen surfaces," *Smart Innovation, Systems and Technologies*, vol. 140, pp. 161-169, 2019, 10.1007/978-3-030-16053-1_15.

[32] M. K. Yahya-Imam, S. Palaniappan, and S. M. Ghadiri, "Investigation of methodical framework for cross-platform mobile application development: Significance of Codename One," *International Journal of Computer Aided Engineering and Technology*, vol. 11, no. 4-5, pp. 439-448, 2019, 10.1504/IJCAET.2019.100443.

[33] P. S. Mendez, J. C. Silva, and J. L. Silva, "Multi-screen and multi-device game development," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10272 LNCS, pp. 74-83, 2017, 10.1007/978-3-319-58077-7_7.

[34] M. L. Hamzah, A. A. Purwati, E. Rusilawati, and Hamzah, "Rapid application development in design of library information system in higher education," *International Journal of Scientific and Technology Research*, vol. 8, no. 11, pp. 153-156, Nov. 2019.