Efficient optimization to the N Queens Problem

Hadj Mohammed Djamel
Department of Computer Science
USTHB University
Algiers, Algeria
hadjmohammeddjamel@gmail.com

***Abstract*—We present an efficient solution to the N-Queens puzzle using the heuristic algorithm proposed in the paper. The N-Queens problem become intractable for large values of N and this placed in NP (No deterministic Polynomial) class problem because of their high complexity (eg: O(N!) , O(2^N)) they cannot solved in a realistic time frame using deterministic technique, very efficient AI (Artificial Intelligence) search algorithm solving strategies, even classic Local Search algorithm developed so far cannot find a solution for large size N-Queens problem in a reasonable time, in this paper we present a new optimization that combine the Local Search algorithm and our proposed heuristic algorithm which is based on a specific generated N-Queens Starting states. This efficient optimization is capable of finding a solution for extremely large size Queens number in O(N), we give the execution statistics for this hybrid algorithm with number of Queens up to 19,000,000 in less than one second.**

***Index Terms*—Artificial Intelligence(AI), Hybrid Search, Heuristic Algorithm, The N Queens problem**

I. Introduction

The classical N-Queens Problem, first proposed by German chess enthusiast Max Bezzel in 1848 for the standard 8×8 board, and both solved and generalized to larger boards by Franz Nauck in 1850 [1]. The problem is to place n queens on an $N \times N$ chessboard, so that no two queens can capture each other. That is, no two queens are in the same row, column, or diagonal [2]. Note that the problem is solvable in constant time since there is a solution for all $n > 3$ [4]. A simple solution to the N-queens problem, with the use of a sequential search algorithm, consists in Generate all possible placement combinations of queens on the board and choosing a configuration that satisfies the given constraints.

In this paper we give a new probabilistic optimization to the classic local Search algorithm, that based on specific N-Queens starting states generated by the proposed heuristic algorithm. This optimization is capable of providing a solution for extremely large size of number Queens, by using a heuristic algorithm with a time complexity of O(N). Our results show that the program find solution in O(N) with a reasonable iteration number, using a control parameter *P* which is a positive random integer (*P*$\in$ $[1..N]$), these parameter are determined by statistical tests (See $Table I, II$). In our proposed approach, instead of giving random starting states as initial solution for the Local Search algorithm we propose a specific configuration of the initial N-Queens starting states. We will confirm in the experimental section that this combination provides excellent performance.

II. Related work

In recent years, the heuristic search algorithms have received considerable research attention in the AI field. The N Queens problem have been implemented by many researchers. However, they found that the N Queen problem can be successfully solved using heuristic algorithm and the quality of the solutions generated depend on the quality of the heuristic algorithm [5], [6].

Several authors have proposed the use of different search strategy techniques to solve the n-queen problem. These methods include Tabu search algorithm and genetic algorithm [9], Global Parallel Genetic Algorithm (GPGA) [10]. GPGA requires the master process which distributes tasks among the slaves to be executed in parallel. This algorithm based on three basic biological principles: selection, crossover, and mutation. Also, reference [11] presents a probabilistic algorithm; a gradient-based heuristic is successfully applied to the n-queens problem for large values of n. The main idea is to Generate a random permutation of queens and swapping a pair of queens to minimize the number of collisions. In [3]. introduced a distributed solution for n-queens problem using dP systems, where the components are P systems with active membranes. Another study [7] [8] that aims at solving the N Queen problem, and variations of it, using atoms with cavity-mediated long-range interactions.

Recently, this problem has been solved with N up to about 26 in which has 2.23 x 1016 solutions, in 271 days [12]. In this paper we show how a new heuristic algorithm can be used to solve the N Queen problem, and has an optimal running time complexity.

III. Solving the N-Queens problem

A. Method

let

1) N is the number of Queens.
2) $queens[1..N]$ its N Queens Array, the structure we use to represent the chess board.
3) $column_i \longrightarrow queens[i]$.
4) $row_i \longrightarrow i$.
5) All solutions to the N-Queens problem has exactly one Queen in each row. As a result, we can represent our solution using an Array $queens[1..N]$, where $queens[]$ array indicate if column j in row i contain a queen or not, $queens[i] = j$ means queen at *i-th* row is placed at *j-th* column. Let's have an example (See $Fig.$ 1).

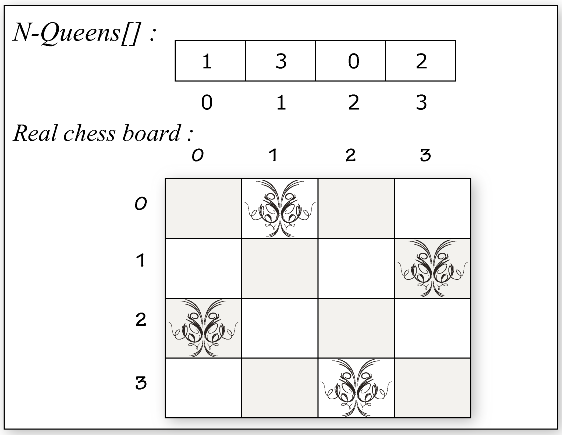

Fig. 1: Solution of the N-Queens problem with $N = 4$.

B. The global algorithm

The global algorithm consists of two parts:

1) Generate the initial N-Queens starting states: The first part is our heuristic algorithm proposed to solve the N-Queens problem. For 8-Queens problem, there are $4,426,165,368$ possible arrangements of queens on board out of which only 92 are correct solutions [13]. The aim is to minimize the number of attacked queens in the initial N-Queens starting states. The proposed Heuristic algorithm *Generat-Init-nQueens()*[1](See $Algorithm$: 1), generate the initial N-Queens starting states with improved parameter *P*. The running time of our heuristic algorithm is O(N), that places all N Queens in order, from the upper row to the lower row on an $N \times N$ *chess board* simulated in Array $queens[1..N]$.

Our optimization consists in building a starting states in such a way that may solve or reduce the number of collisions among the queens in polynomial time. The main idea behind this optimization is to put queens on the board row by row using the *P* value to locate the *j-th* column of the first row, starting at the top (See $Algorithm$: 1):

- The first queen can be placed in $queens[i] = 1$ knowing that i =*P* and i <=*N*.
- The second queen will be placed in $queens[i']$ = 2 knowing that i'=*i*+2 and i'<=*N*.
- The last queen configuration is $queens[i'']$ =N knowing that i''=*i'*+2 and i''<=*N*.
- If i >N than i get a new value depend on P as shown in $Algorithm$: 1.

If no collisions among Queens in the generated N-Queens starting states, no action is taken, that is, a solution is found in polynomial time; otherwise, selecting the N-Queens starting states that generated by heuristic algorithm as initial solution for local search algorithm to find a solution, in practice, the running time is approximately O(N). However, if the heuristic algorithm does not find a solution its guarantee that the generated N-Queens starting states has a reduced number of collisions. Finally, if the local search algorithm get stuck in a local minimum [14] with the given N-Queens starting states, a new value of P is generated and the heuristic algorithm starts a new search.

[1]The *Generat-Init-nQueens()* algorithm proposed by H.Djamel.

Algorithm 1: *Generat-Init-nQueens*

```
Input: $P$ its random Integer $\in [1..N]$ used to generate
       N-Queens starting states.
Output: $queens[0..N-1]$ is the initial starting states
        with the minimum number of conflicts.
column ← P;
row ← 0;
save ← column+1;
if $Mod(column, 2) = 0$ then
    while $row < N$ do
        if $column + 2 < N$ then
            $queens[row] = column + 2$;
            $column = column + 2$;
        else
            if $save = 2$ then
                $column = save$; $save = save + 1$;
            else
                $column = 1$; $save = 2$;
            $queens[row] = column$;
        $row \leftarrow row + 1$;
    if $queens[0] = queens[N-1]$ then
        $queens[N-1] = 0$;
else
    while $row < N$ do
        if $column + 2 < N$ then
            $queens[row] = column + 2; column =$
            $column + 2$;
        else
            if $save = 1$ then
                $column = save$; $save = save + 1$;
            else
                $column = 0$; $save = 1$;
            $queens[row] = column$;
        $row \leftarrow row + 1$;;
return $queens$;
```

Example 3.1: For the standard *8X8* board, if the input parameter P equal 2 than the initial starting states will be configured in board as shown in $Fig.$ 2.

2) Local Search solving strategy: In the second part, we choose to use the local search procedure [15]. The N-Queens problem is easy for local search because solutions are densely distributed. The local search algorithm based on replaces the current solution by a neighboring solution of better quality. The local search algorithm is used as an alternative solution to solve any collisions among queens in the initial N-Queens

TABLE I: The estimated time(second) required to solve the N-queens problem with N up to $500,000$.

Number of Queens N	100	1000	2000	10000	100000	500000
Time of the 1st run	<0,001	<0,001	0,02	0,01	0,01	1,62
Time of the 2nd run	<0,001	0,01	0,02	<0,001	0,02	2,28
Time of the 3rd run	<0,001	<0,001	0,01	<0,001	0,01	5,25
Time of the 4th run	<0,001	<0,001	0,02	0,01	0,01	1,8
Time of the 5th run	<0,001	<0,001	0,02	<0,001	0,01	3,52
Time of the 6th run	<0,001	<0,001	0,01	<0,001	0,01	3,74
Time of the 7th run	<0,001	<0,001	0,02	<0,001	0,02	1,83
Time of the 8th run	<0,001	<0,001	0,02	<0,001	0,01	1,72
Time of the 9th run	<0,001	<0,001	0,02	0,01	0,01	2,08
Time of the 10th run	<0,001	<0,001	0,02	<0,001	0,01	1,72
Ave Time to Find Solution	<0,001	<0,001	0,02	<0,001	0,01	2,56

TABLE II: The estimated time(second) required to solve the N-queens problem with N up to $19,000,000$.

Number of Queens N	7000000	10000000	12000000	15000000	17000000	19000000
Time of the 1st run	0,41	0,35	0,4	0,57	0,82	0,94
Time of the 2nd run	0,4	0,37	0,4	0,49	0,75	1,12
Time of the 3rd run	0,41	0,37	0,42	0,5	0,79	0,99
Time of the 4th run	0,39	0,38	0,41	0,49	0,84	0,93
Time of the 5th run	0,39	0,34	0,43	0,51	0,78	0,94
Time of the 6th run	0,41	0,39	0,4	0,5	0,76	0,99
Time of the 7th run	0,41	0,36	0,4	0,49	0,82	0,96
Time of the 8th run	0,41	0,38	0,4	0,55	0,81	0,99
Time of the 9th run	0,4	0,36	0,41	0,53	0,8	1
Time of the 10th run	0,43	0,35	0,39	0,49	0,79	0,93
Ave Time to Find Solution	0,41	0,37	0,41	0,51	0,80	0,98

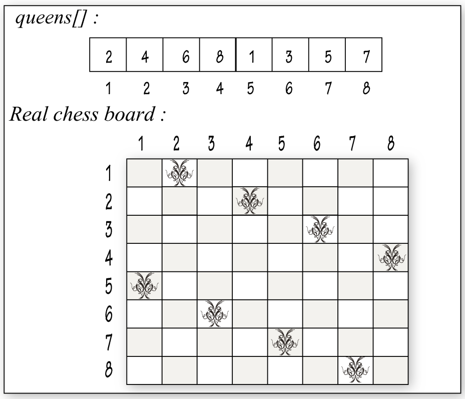

Fig. 2: The initial N-Queens starting states with collisions for the standard *8X8* board and P equal 2.

starting states that may occur generally on the diagonals.

IV. Results

As illustrated in $Fig.$ 2, the number of collisions for the *8X8* board with P equal 2 is *6*, so the number of collisions which must be resolved in the second part equal *6*.
Among the algorithm features we have studied, we observed a large Number of *NXN* board generated in The proposed heuristic algorithm (See $Algorithm$: 1) have initial collisions equal *0*. In other words, the heuristic algorithm success in finding a valid solution without calling Local search algorithm.

Table I and Table II were obtained from 10 sample algorithm runs. In the tables below, the computational complexity is measured by using execution time. For probabilistic reasons, execution time stabilizes independent of the increase in problem complexity.

This statistic highly depend on the parameter P which initialed $P = 1$ for all tests.

From the tables, one can infer that the execution time strongly depend on the parameter P. The algorithm is probabilistic. That is, if the algorithm could not find a solution with the given value of P, a new value is generated and the algorithm starts a new search. In addition, to confirm that combining the Local Search algorithm with our proposed heuristic algorithm produces a significant improvement, we execute the local search algorithm starting from an initial random population. The algorithm can find a solution for N up to 500 in a reasonable time (7min). When N is too high the local search algorithm becomes inefficient to find a solution to the N-Queens problem in reasonable time.

V. Conclusion

The proposed heuristic algorithm can solve the N-Queens problem in a fraction of second with the appropriate value of P. Also, for different values of N and specific value of parameter P, those *NXN* boards can be successfully resolved using our heuristic algorithm only, in O(N). As a part of future work will make a Combinations of our heuristic algorithm and others search algorithm solving strategies and will include the proposed heuristic algorithm for solving the N-Queens problem which systematically generates all possible solutions for extremely large size N-Queens problems.

References

[1] Dealy, S. (2004). Common Search Strategies and Heuristics With Respect to the N-Queens Problem CS504 Term Project.

[2] Cengiz Erbas, Seyed Sarkeshik, and Murat M. Tanik. Different perspectives of the N-Queens problem. In ACM annual conference on Communications, pages $99 - 108, 1992$.

[3] Bu$\tilde{n}$o, K. C., Cabarle, F. G. C., Calabia, M. D., & Adorna, H. N. (2018). Solving the N-Queens problem using dP systems with active membranes. Theoretical Computer Science, 736, 1-14.

[4] Gent, I. P., Jefferson, C., & Nightingale, P. (2017). Complexity of n-queens completion. Journal of Artificial Intelligence Research, 59, 815-848.

[5] POTHUMANI, Solving N. Queen Problem Using Various Algorithms-A Survey. International Journal of Advanced Research in Computer Science and Software Engineering, 2013, vol. 3, no 2

[6] FARHAN, Ahmed S., TAREQ, Wadhah Z., et AWAD, Fouad H. Solving N Queen Problem using Genetic Algorithm. International Journal of Computer Applications, 2015, vol. 122, no 12.

[7] Torggler, V., Aumann, P., Ritsch, H., & Lechner, W. (2018). A quantum n-queens solver. arXiv preprint arXiv:1803.00735.

[8] Draa, A., Meshoul, S., Talbi, H., & Batouche, M. (2011). A quantum-inspired differential evolution algorithm for solving the N-queens problem. neural networks, 1(2).

[9] Martinjak, Ivica, and Marin Golub. ”Comparison of heuristic algorithms for the n-queen problem.” Information Technology Interfaces, 2007. ITI 2007. 29th International Conference on. IEEE, 2007.

[10] Boikovic, Marko, Marin Golub, and Leo Budin. ”Solving n-Queen problem using global parallel genetic algorithm.” International Conference on Computer as a tool EUROCON 2003. 2003.

[11] 2015 (pp. 102-104). IEEE. Sosic, Rok, and Jun Gu. ”A polynomial time algorithm for the n-queens problem.” ACM SIGART Bulletin 1.3 (1990): 7-11.

[12] Vaughan, N. (2015, July). Swapping algorithm and meta-heuristic solutions for combinatorial optimization n-queens problem. In Science and Information Conference (SAI),

[13] Eight Queen Puzzle,[Online]Available:http://en.wikipedia.org/wiki/Eight queens puzzle [Accessed:July.20, 2009].

[14] Selman, B., Kautz, H. A., Cohen, B. (1994, October). Noise strategies for improving local search. In AAAI (Vol. 94, pp. 337-343).

[15] Johnson, D. S., Papadimitriou, C. H., Yannakakis, M. (1988). How easy is local search?. Journal of computer and system sciences, 37(1), 79-100.