New Approach to Solve N-Queen Problem with Parallel Genetic Algorithm

Monire Taheri Sarvetamin[1*], Amid Khatibi Bardsiri[2]

1 Department of Computer engineering, Islamic Azad University, Kerman, Iran
2 Assistant Professor, Department of Computer engineering, Islamic Azad University, Kerman, Iran

Abstract
Over the past few decades great efforts were made to solve uncertain hybrid optimization problems. The n-Queen problem is one of this such problem that many solutions have been proposed for. The traditional methods to solve this problem are exponential in terms of runtime and are not acceptable in terms of space and memory complexity. In this study, parallel genetic algorithms are proposed to solve the n-Queen problem. Parallelizing island genetic algorithm and Cellular genetic algorithm was implemented and run. The results show that this algorithm has the ability to find related solutions to this problem. The algorithms are not only faster but also they lead to better performance even without the use of parallel hardware and just running on one core processor. Good comparisons were made between the proposed method and serial genetic algorithms in order to measure the performance of the proposed method. The experimental results show that the algorithm has high efficiency for large-size problems in comparison with genetic algorithms, and in some cases it can achieve super linear speedup. The proposed method in the present study can be easily developed to solve other optimization problems.

Keywords:
Parallel Genetic Algorithms, Island Genetic Algorithm, Cellular Genetic Algorithm, N-Queen Problem.

1. Introduction

The N-Queen problem is a well-known CSP [1] problem. It includes putting n queens on a chessboard so that no two queens threaten each other. The number of possible combinations of n queens on a chessboard is plentiful. The traditional methods of solving this problem were all based on back-tracking. The time for searching back-tracking is exponential and cannot support solving large-scale n-Queen problems. Although then-Queen problem has little practical applications, but it shows a large class of nondeterministic problems that cannot be solved using deterministic algorithms in a reasonable time.

Many algorithms and methods have been used to resolve the n-Queen problem [1-8]. Ahrabian et al.[9] used the DNA Sticker algorithm and Khan[10] has successfully used the ant colony algorithm to solve the problem .Farhan et al.[11] used a genetic algorithm to solve this problem and found all 92 possible solutions to the 8-queen problem. In 2003, Božikovic et al.[12] Used global parallel genetic algorithm and a 3-way tournament method to solve this problem. They conducted simulation on a single-processor computer and came to the conclusion that global parallel genetic algorithm was not suitable for large-scale parallel processing; rather it was suitable for a small number of parallel processing units.

The parallel genetic algorithms or PGAs were successfully used in a wide range of optimization problems [13-16]. The good strength of these algorithms in high complexity problems has increased their applications in the fields of Artificial intelligence, numerical and combinatorial optimization, business, engineering, etc. Therefore they were used in this study[17]. In addition to reducing computation time, these algorithms lead to many better explorations and diversities compared to the sequential genetic algorithms [18].

Recent studies have made efforts to resolve the n-Queen problem with different methods. The main purpose of the present study is solving the n-Queen problem using parallel genetic algorithms and finding solutions, as well as reducing the runtime of the program with the parallel genetic algorithms compared to these quential genetic algorithms. The efficiency of these two algorithms compared to the serial version will be presented later in this paper. Parallel is min this study was conducted using MATLAB Parallel Computing Toolbox on a personal computer. This article is organized into the following sections: Section 2 introduces how to display and code the n-Queen problem; Section 3 describes the proposed method and its configurations; Section 4 contains experimental design; and Section 5 presents experimental results. Conclusions are presented in Section 6.

Manuscript received May 5, 2026
Manuscript revised May 20, 2026
https://doi.org/10.22937/IJCSNS.2026.26.5.4

2. Problem Definition

Then-Queen is displayed as an n-tuple. Each queen must be on a different row and column. It can be assumed that the queen i is placed in the i^{th} column. n-tuple solutions are displayed as $(q_1,q_2,\ldots,q_n)$ which are the permutations of an n-tuple (1,2,3,...,n). The place of a number on a tuple defines the column of a queen, while its value shows its row (counting from the bottom). Figure 1 shows the 4-tuple display of the problem.

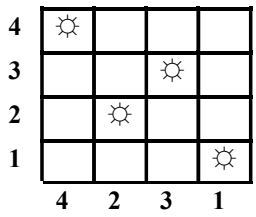

Figure 1 Example of registering a 4-tuple

Fitness function for this problem should count the number conflicts of queens and should calculate the number of rows, columns, and diagonal conflicts. In the correct solution to this problem, this number should be zero. Two fitness functions can be written for this problem, one with $O(n^2)$ time complexity and the other with O(n) time complexity. Writing a fitness function with $O(n^2)$ time complexity is very simple and common. However, the method of calculating the fitness function with O(n) time complexity is as follows:

In this model, the number of diagonal conflicts is obtained to calculate the number of conflicts of the queens. The method is explained in this way: because the n-tuple display removes conflicts in rows and columns, so there are only diagonal threats between the queens; therefore the fitness function only calculates the diagonal conflicts. As shown in Figure 2, there are 2n-1 left (top-bottom, left-to-right) and 2n-1 right (bottom-up, right-to-left) diameters.

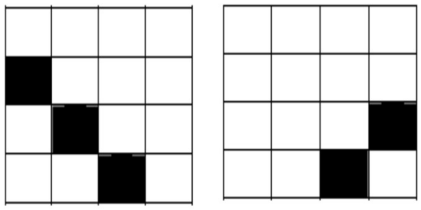

Figure 2 The third left diameter and the second right diameter for 4-queens.

The diagonal conflicts are calculated as: the i^{th} and j^{th} queen are part of a diameter if:

$$|q_i - q_j| = |i - j| \quad (1)$$

Formula 1 results in the O(n) complexity for the fitness function.

3. Suggested Methods

Substantially, the parallel genetic algorithm consists of several genetic algorithms, each of which processes a separate section of a population or populations with or without communication between them. Hence the parallel genetic algorithms can increase population diversity and decrease computational time. These algorithms present a new type of meta heuristics that have a higher efficiency and impact on the population structure and parallel implementation[17]. As mentioned, these algorithms can be divided into two main types, coarse-grained or island genetic algorithms and fine-grained or cellular genetic algorithms. The island genetic algorithm runs several subpopulations in the parallel mode. The subpopulations make up the total population of the island together. Most of the time, the islands work independently in the run model. However, the solutions are periodically exchanged between the islands. This is called the migration process. Using migration, the island model is able to extract the differences in the various sub-populations. The change shows a source of genetic diversity.

The other model is the cellular genetic algorithm, the most prominent feature of which is that each island contains only one person. In this sense of islands, most of the cells are read, and each person is only allowed to be combined with neighboring people. The difference between this model and the island model is that no evolutions happen in the very cells; in fact there are no evolutions within the islands. Improvements can only be achieved by cells that interact with each other[18].

3.1. Selection Guidelines

The main part of the selection process is the random selection of a generation to lay the foundation for the next generation. The most appropriate people are more likely than the poor people. Here the roulette wheel selection method is used to select the chromosomes.

3.2. Crossover operator

Crossover operator is determined with a probability rate (P_c). Here a technique is used that is a combination of single-point crossover, double-point crossover, and uniform crossover methods. The crossover method was placed with various probabilities. The probability for single-point

crossover, double-point crossover, and uniform crossover were assumed as $P_{singlePoint}$=0.1, $P_{doublePoint}$=0.2, and $P_{uniform}$=1-$P_{singlePoint}$-$P_{doublePoint}$, respectively. Each time the crossover function is invoked, one of the above-mentioned three methods is selected using roulette wheel for conducting crossover action.

3.3. Mutation operator

Like crossover operator, a mutation operator with a P_m rate is assumed to determine whether the mutation operator is applied in the chromosome or not. Here the swap mechanism, or changing the places of the elements, is used. Figure 3 shows how it works.

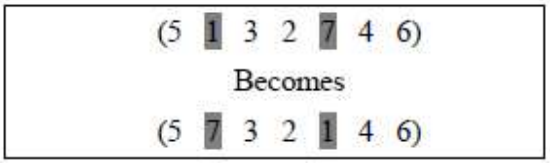

Figure 3 Mutation operator

3.4. Stop criteria

Stop criteria are considered as reaching a certain number of generation iterations. This number will vary for a different number of queens but it is selected in such a way to be sure that the algorithm converges, and the algorithm can find the answers to this problem.

4. Experimental Design

4.1. Strategies related to island genetic algorithms

The island genetic algorithm runs several subpopulations (islands) in the parallel mode. Exchange of information between these subpopulations happens in specified intervals (epoch) in the iteration loops. With the exchange of outstanding chromosomes between subpopulations, the search space of subpopulations subset diversifies to prevent the premature convergence more effectively. n and P_n, are the amounts of subpopulations and scales, respectively. Hence the total population is P_{size}=nxP_n. Here the ring topology is used for migration and it has a single direction. With a single migration frequency, a certain number of the best people in subpopulations are selected for migration and a copy of these people is sent to the neighbors. The immigrants arrived at the destination islands replace the worst individuals.

4.2. Measuring efficiency

As Alba[19] has pointed out, comparing the efficiency of parallel and serial evolutionary algorithms only makes sense when they reach a common accuracy. In a classification by Alba[19] there are two general types of strong speedup and weak speedup to compare the parallel and serial genetic algorithms. Strong speedup is not commonly used. In the weak speedup, the parallel and serial runtime of an algorithm are compared including the following:

- **Single machine/panmixia:** The parallel algorithm is compared with a standard version of its which runs on a single machine. For example, we may compare an island with *m* islands with a genetic algorithm that runs one island. Where by the algorithm that runs on all islands in both cases is the same.
- **Orthodox:** The parallel algorithm that runs on *m* machines is compared with the same algorithm that runs on one machine.
-

4.3. Paralleling with MATLAB Parallel Computing Toolbox

For the purpose of paralleling, Parallel Computing Toolbox (PCT) of MATLAB is selected. The Parallel Computing Toolbox provides the possibility to run massive computational problems using multi core processors, GPUs and computer clusters. The toolbox allows the use of whole processing power of a multi core desktop computer. Full use of a desktop computer multi core processors' power is possible through workers that run locally[20].

The Toolbox provides a local cluster of workers for the local machine and runs applications on local workers (MATLAB computational engines). The same application can run without changing the code on a cluster of computers or a grid computing service using MATLAB distributed computing server. The number of workers in the toolbox depends on the hardware and their connections. In any case, to use the processor of a multi core system, the number of workers is usually considered equal to the number of cores. In the 2012 edition of MATLAB, the toolbox allows users to have 12 workers work on a single machine.

5. experimental results

Parallel genetic algorithms were successfully applied on the n-Queen problem. Both algorithms were able to find several solutions for the given number of queens. The results were run on a PC with Intel Pentium(R) Dual-CoreCPUE5700 @ 3.00GHz, 2.00GB of RAM, and Windows 7. The size of the issue was set between 10 and 500 queens. Running parallel genetic algorithms on a dual-core processor reduced the runtime. In order to compare the results, both orthodox and a single–machine method were used for comparing the efficiency of algorithms.

5.1. Orthodox comparing method

In this comparison method the parallel genetic algorithms for the n-Queen problem were once run in parallel on a dual-core processor and again in series on a single-core processor, and the runtime was measured. The speedup and efficiency values of these algorithms were compared to their serial version .Tables 1 and 2 show runtime, speedup and efficiency of cellular and island genetic algorithms compared to their serial version for various numbers of queens.

Table 1. Runtime , efficiency and speed up of the cellular genetic algorithm compared to the serial version

The number of queens	T_S	T_P	speedup	efficiency
10	5.0888	3.1536	1.6136	0.8068
50	10.8353	6.268	1.7287	0.8643
100	29.1715	16.3961	1.7792	0.8895
500	188.4179	104.8886	1.7964	0.8981
T_S=serial runtime of cellular genetic algorithm				
T_p=parallel runtime of cellular genetic algorithm				

Table 2. Runtime, efficiency and speed up of the island genetic algorithm compared to the serial version

The number of queens	T_S	T_P	speedup	efficiency
10	1.4869	1.3456	1.1051	0.5525
50	3.2465	2.1179	1.5329	0.7664
100	14.0042	8.0016	1.7502	0.8756
500	126.33	72.080	1.7528	0.8796
T_S=serial runtime of island genetic algorithm				
T_p=parallel runtime of island genetic algorithm				

As specified in Tables 1 and 2, both island and cellular genetic algorithms are able to run the n-Queen problem in far less time than the serial versions. In addition, the table values show that the efficiency of these algorithms increases with an increase in the number of queens. It can be concluded that parallel genetic algorithms show a better performance for massive and time consuming problems. Figure 1 shows the efficiency of a parallel genetic algorithm for different numbers of queens.

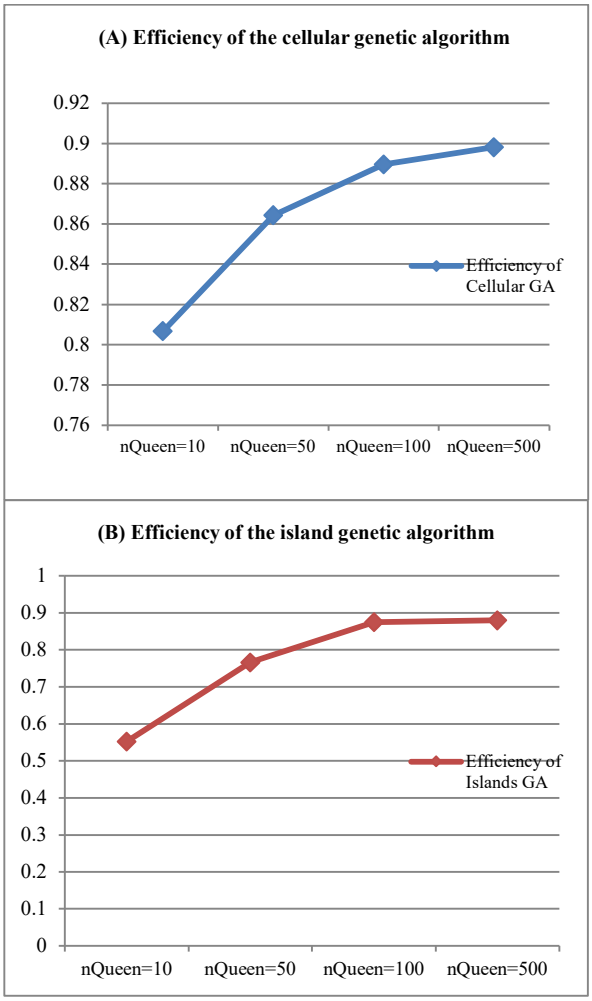

Figure 1. (A) Efficiency of cellular genetic algorithms (B) Efficiency of island genetic algorithms

Several factors are involved in the incomplete speedup (about 1.7 for the dual-core processor), and thus the lack of efficiency including code and data transfer between the client and the workers, and resource competition between the working processes and operating system processes. Also, as Figure 1 shows, the more the number of queens the higher the efficiency of these algorithms. However, there as on for low efficiency in low numbers of queens refers to the fact that opening and closing paralleling tools are time consuming. Hence using this algorithm for a lownumber of queens is not recommended. Rather, as Figure 1 shows, by increasing the number of queens and the complexity of the issue, efficiency of these algorithms increases and makes them affordable.

When MATLAB parallel computing toolbox runs a program locally on a computer, sets the number of workers

to the number of CPU cores by default, however this toolkit allows the number of workers to be set more than the number of CPU cores. In another experiment, the number of workers was increased and the run time and the speedup of parallel algorithms were measured for 100 queens. Figure 2 shows the efficiency of parallel genetic algorithms compared to the serial version for a different number of workers.

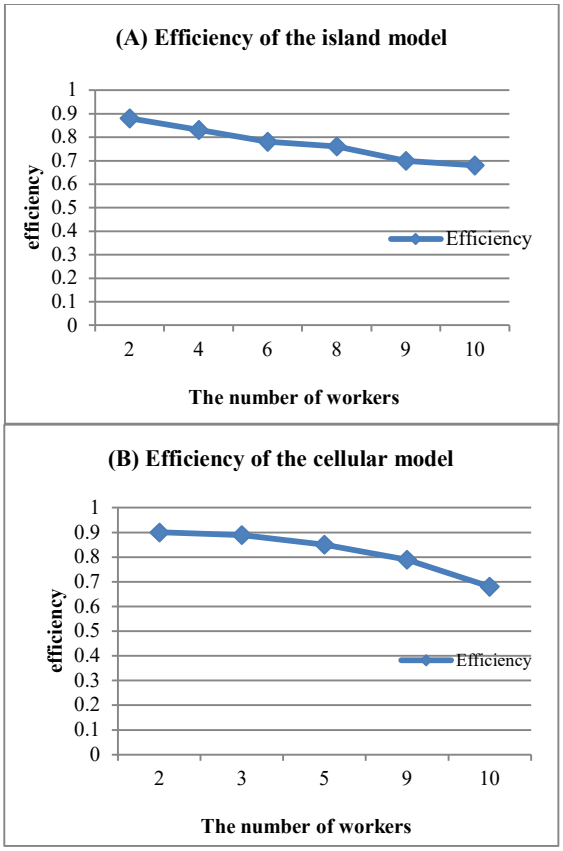

Figure 2. (A) Efficiency of the island genetic algorithm increasing the number of workers; (B) Efficiency of cellular genetic algorithm by increasing the number of workers

According to Figure 2, the number of workers in MATLAB without increasing the number of CPUs or CPU cores will have no effect on the runtime and will make no improvement in the speed, which is due to the overhead associated with concurrence. As long as the number of workers is considered equal to or less than the CPU cores, the program virtually runs in parallel mode, but when the number of workers is considered to be more than CPU cores, then it will be concurrent. The remarkable thing when working with parallel MATLAB toolbox is, for example, when the program is set to run on four cores, the achieved speed is close to 1-3 cores, because the last core has to run system processes.

Running the program on a cluster of computers appears to have better results and shall lead to super linear speedup. The physical resources will be considered as a possible reason to realize this. When running on a cluster, more resources in terms of memory or cache might be available for the program, and when moving codes from a single machine to a cluster of computers, algorithms - possibly – use these additional resources. Also, each machine may work only with a small packet of data, while smaller data may fit in the cache. Whereas that is not the case in a single machine and will create a significant difference in efficiency.

5.2. Single-machine comparing method

In this method, island and cellular genetic algorithms with parallel computing toolbox were run in parallel on a dual-core processor for comparison of the n-Queen problem. The results were compared with standard genetic algorithm - having a single population and doing evolutionary operations until a particular condition arises – to measure its efficiency compared to the standard genetic algorithm. In general, the standard genetic algorithm needs more iterations than parallel genetic algorithm to reach the answer which increases its runtime. Table 3 shows runtime, speedup and efficiency of parallel genetic algorithms compared to standard genetic algorithms.

Table 3. Runtime, speed up and efficiency of the island and cellular genetic algorithm compared to the standard genetic algorithm

The number of queens	T_{GA}	T_{IGA}	T_{CGA}	S_{IGA}	S_{CGA}	E_{IGA}	E_{CGA}
10	1.5522	1.4869	1.3456	1.153537	0.492199	0.576769	0.2461
50	5.0065	3.2465	2.1179	2.363898	0.79874	1.181949	0.39937
100	27.4384	14.0042	8.0016	3.429114	1.68374	1.714557	0.84187
500	264.13	126.33	72.080	3.664401	2.518188	1.8322	1.259094
T_{GA}: the standard genetic algorithm runtime (single population)							
S_{CGA}: Speedup of the cellular genetic algorithm compared to the standard genetic algorithm				T_{IGA}: Runtime of the island genetic algorithm			
E_{IGA}: efficiency of island genetic algorithm compared to the standard genetic algorithm				T_{CGA}: Runtime of the cellular genetic algorithm			
E_{CGA}: efficiency of cellular genetic algorithm compared to the standard genetic algorithm				S_{IGA}: Speedup of the island genetic algorithm compared to the standard genetic algorithm			

As shown in Table 3, by increasing the number of queens, the speedup and efficiency of the algorithms increase. Based on experimental results both algorithms reached super linear speedup for the highest number of queens examined, i.e. 500. Figure 3 shows the efficiency of the island and cellular genetic algorithm compared to the standard genetic algorithm.

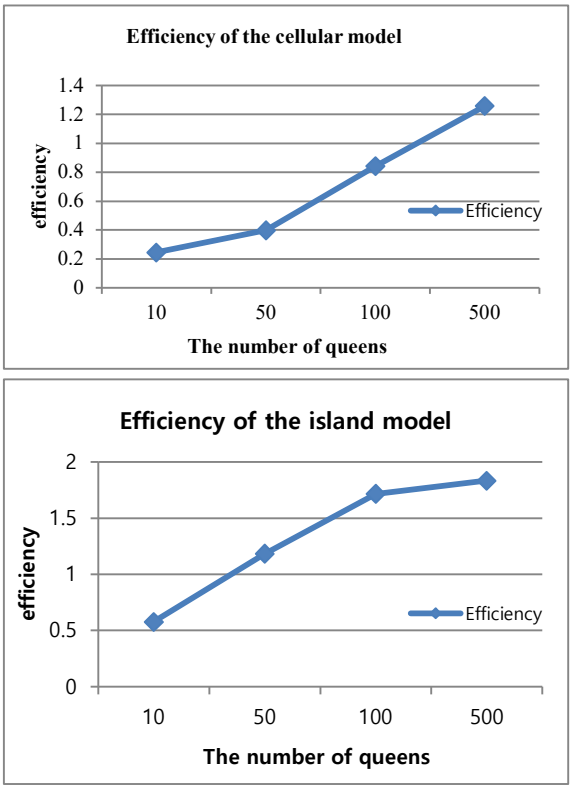

Figure 3. Efficiency of island and cellular genetic algorithm compared to the standard genetic algorithm

Additional tests were conducted, too. For this problem, cellular and island genetic algorithms were run without a parallel hardware and in series on a single core processor. They were compared with standard genetic algorithms to measure their performance compared to the standard genetic algorithm. The experimental results showed that the standard genetic algorithm usually needs more iteration to reach the answer than the parallel genetic algorithm, which increases its runtime.

Both the parallel genetic algorithms that were run without hardware paralleling and in series mode reached the answer in fewer iterations and thus needed less time. However the island genetic algorithm reached the answer in far less time due to its specific better performance. This performance can be due to migration between subpopulations because migration allows the subpopulations to share genetic materials and increases diversity in subpopulations.

The experiments carried out indicated that the use of parallel genetic algorithms, not only leads to faster algorithms, but also yields better numerical performance, even when the algorithm runs on a processor with one core. Interestingly, using structured population in the form of islands or in the form of a network is responsible for such numerical benefits.

Table 4. Speed up and efficiency in different methods compared to the standard genetic algorithm for 500-queen problem

Algorithm	The number of iterations	The number of queens	Population	Time	speedup
standard genetic algorithm	8000	500	100	264.13 seconds	1
The island genetic algorithm without hardware paralleling (two islands)	4000	500	100	126.064 seconds	2.095
The cellular genetic algorithm without hardware paralleling	2000	500	100	188.5514 seconds	1.40

6. Conclusion

N-Queen problem is well-known CSP problem. Although n-Queen problem has little practical application, but it shows a large class of nondeterministic issues that cannot be solved using deterministic algorithms in a reasonable time. The present study showed that the n-Queen problem can be successfully run with parallel genetic algorithms. The experimental results showed that these algorithms are able to find different solutions for a specified number of queens. Parallel genetic algorithms have a better performance than the serial version of genetic algorithms and run in less time to answer even when implemented on a single-core processor. These algorithms lead to calculations speedup and they find better solutions compared to the serial version. In addition to reducing computation time, these algorithms lead to increased exploration and better diversity compared to the serial genetic algorithms. Also results indicated that these algorithms are more efficient with large numbers of queens than parallel genetic algorithms, and in a type of comparison they can achieve super linear speedup. As a result, using these algorithms is suitable for massive parallel processing. Running parallel genetic algorithms on a cluster of computers is suggested for future work.

References

[1] Bashir, L.Z. and N. Mahdi, *Use Genetic Algorithm in Optimization Function For Solving Queens Problem.* World Scientific News, 2015. **11**: p. 138-150.

[2] Ohta, M., *Chaotic neural networks with reinforced self-feedbacks and its application to N-Queen problem.* Mathematics and computers in simulation, 2002. **59**(4): p. 305-317.

[3] Hu, X., R.C. Eberhart, and Y. Shi. *Swarm intelligence for permutation optimization: a case study of n-queens problem*. in *Swarm intelligence symposium, 2003. SIS'03. Proceedings of the 2003 IEEE*. 2003. IEEE.

[4] Agarwal, K., A. Sinha, and M.H. Bindu, *A novel hybrid approach to N-queen problem*, in *Advances in Computer Science, Engineering & Applications*. 2012, Springer. p. 519-527.

[5] El-Qawasmeh, E. and K. Al-Noubani. *A Polynomial Time Algorithm for the N-Queens Problem*. in *IASTED International Conference on Neural Networks and Computational Intelligence*. 2004.

[6] Hu, N., *An Integer Coding Based Optimization Model for Queen Problems.* American Journal of Computational Mathematics, 2016. **6**(01): p. 32.

[7] Mandziuk, J., *Solving the n-queens problem with a binary Hopfield-type network. Synchronous and asynchronous model.* Biological Cybernetics, 1995. **72**(5): p. 439-446.

[8] Mohabbati-Kalejahi, N., H. Akbaripour, and E. Masehian, *Basic and Hybrid Imperialist Competitive Algorithms for Solving the Non-attacking and Non-dominating n-Queens Problems*, in *Computational Intelligence*. 2015, Springer. p. 79-96.

[9] Ahrabian, H., A. Mirzaei, and A. Nowzari-Dalini, *A DNA Sticker Algorithm for Solving N-Queen Problem.* IJCSA, 2008. **5**(2): p. 12-22.

[10]. Khan, S., et al. *Solution of n-queen problem using aco*. in *Multitopic Conference, 2009. INMIC 2009. IEEE 13th International*. 2009. IEEE.

[11]. Farhan, A.S., W.Z. Tareq, and F.H. Awad, *Solving N Queen Problem using Genetic Algorithm.* International Journal of Computer Applications, 2015. **122**(12).

[12]. Božikovic, M., M. Golub, and L. Budin. *Solving n-Queen problem using global parallel genetic algorithm*. in *International Conference on Computer as a tool EUROCON 2003*. 2003.

[13] Dash, S.R., S. Dehuri, and S. Rayaguru. *Discovering interesting rules from biological data using parallel genetic algorithm*. in *Advance*

Computing Conference (IACC), 2013 IEEE 3rd International. 2013. IEEE.

[14] Yu, B., et al., *Parallel genetic algorithm in bus route headway optimization.* Applied Soft Computing, 2011. **11**(8): p. 5081-5091.

[15] Mihaylova, P. and K. Brandisky. *Parallel genetic algorithm optimization of die press*. in *Proc. of 3rd International PhD Seminar "Computational Electromagnetics And Technical Applications*. 2006.

[16] Soufan, O., et al., *DWFS: a wrapper feature selection tool based on a parallel genetic algorithm.* PloS one, 2015. **10**(2): p. e0117988.

[17] Alba, E. and J.M. Troya, *A survey of parallel distributed genetic algorithms.* Complexity, 1999. **4**(4): p. 31-52.

[18] Kacprzyk, J. and W. Pedrycz, *Springer Handbook of Computational Intelligence*. 2015: Springer.

[19] Alba, E., *Parallel evolutionary algorithms can achieve super-linear performance.* Information Processing Letters, 2002. **82**(1): p. 7-13.

[20] Sharma, G. and J. Martin, *MATLAB®: a language for parallel computing.* International Journal of Parallel Programming, 2009. **37**(1): p. 3-36.