Solving the *n*-Queens Problem Using a Tuned Hybrid Imperialist Competitive Algorithm

Ellips Masehian[1], Hossein Akbaripour[1], and Nasrin Mohabbati-Kalejahi[2]
[1]Industrial Engineering Department, Tarbiat Modares University, Iran
[2]Faculty of Industrial Engineering, Amirkabir University of Technology, Iran

Abstract: *The n-queens problem is a classical combinatorial optimization problem which has been proved to be NP-hard. The goal is to place n non-attacking queens on an n×n chessboard. In this paper, the Imperialist Competitive Algorithm (ICA), which is a recent evolutionary metaheuristic method, has been applied for solving the n-queens problem. As another variation, the ICA was combined with a local search method, resulting the Hybrid ICA (HICA). Since, the parameters of heuristic and metaheuristic algorithms have a great influence on the performance of the search, parameter tuning is used for handling the problems in an efficient manner. Hence, a TOPSIS-based parameters tuning is proposed, which not only considers the number of Fitness Function Evaluations (FFE), but also aims to minimize the running time of the presented heuristics. In order to, investigate the performance of the suggested approach, a computational analysis on the n-queens problem was performed. Extensive experimental results showed that the proposed HICA outperformed the basic ICA in terms of average runtimes and average number of FFE. The developed algorithms were also compared to the Cooperative PSO (CPSO) algorithm, which is currently the best algorithm in the literature for finding the first valid solution to the n-queens problem, and the results showed that the HICA dominates the CPSO by evaluating the fitness function fewer times.*

Keywords: *n-queens problem, ICA, local search, parameter tuning, TOPSIS method.*

Received May 20, 2012; accepted May 13, 2013; published online March 13, 2014

1. Introduction

The *n*-queens problem is a classical combinatorial optimization problem in Artificial Intelligence [7]. The objective of the problem is to place *n* non-attacking queens on an *n*×*n* chessboard by considering the chess rules. Although, the problem itself has an uncomplicated structure, it has been broadly utilized to develop new intelligent problem solving approaches.

Despite the fact that the *n*-queens problem is often studied as a 'mathematical recreation', it has found several real-world applications such as practical task scheduling and assignment, computer resource management (deadlock prevention and register allocation), VLSI testing, traffic control, communication system design, robot placement for maximum sensor coverage, permutation problems, parallel memory storage schemes, complete mapping problems, constraint satisfaction, and other physics, computer science and industrial applications [19]. The variety of these applications indicates the reason of the wide interest on this well-known problem.

The earliest paper on the general *n*-queens problem was presented by Lionnet [14], and the first proof of the possibility of placing *n* non-attacking queens on an *n*×*n* chessboard is credited to Pauls [17]. A thorough review on the problem and its applications is presented in [4]. The *n*-queens problem belongs to the class of Constraint Satisfaction Problems (CSP), and is known as an NP-hard problem [10].

There are three variants of the *n*-queens problem [1]: Finding all solutions of a given *n*×*n* chessboard, generating one or more, but not all solutions, and finding only one valid solution. In the first variant, finding all solutions may be possible for small sizes, but the number of feasible solutions increases exponentially with the problem size, such that the largest instance solved to date is for $n = 26$ with a total number of 2.23×10^{16} solutions, calculated within 271 days on parallel supercomputers in [20]. A solution to the 8-queens problem (out of 92 solutions) is illustrated in Figure 1, with the permutation presented as (5, 1, 8, 4, 2, 7, 3, 6).

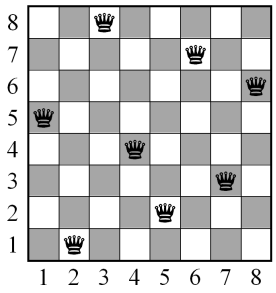


Figure 1. A solution to the 8-queens problem.

According to the extensive bibliography of *n*-queens problems in [13] many mathematical and statistical techniques, heuristic and metaheuristic algorithms, both exact and approximate, have been proposed for solving the problem [7, 15, 18].

The main advantage of metaheuristics compared to exact methods is their ability in handling large-scale instances in a reasonable time [23], but at the expense of losing a guarantee for achieving the optimal solution. Therefore, due to the NP-hardness of the *n*-queens problem, metaheuristic techniques are appropriate choices for solving it.

In designing met heuristics two criteria are important that create two different classes of algorithms: Exploitation (intensification) versus exploration (diversification) [21]. The first class is the algorithms which are able to intensify the search in local regions. They are called Single-solution based metaheuristic (S- metaheuristic), and improve a single solution in solving an optimization problem. Population based metaheuristic (P-metaheuristic) are the second class, which explore the search space and introduce diversity in found solutions. Simulated Annealing (SA) and Tabu Search (TS) are examples of S-metaheuristic, and Genetic Algorithm (GA), Differential Evolution Algorithm (DEA) and Ant Colony Optimization (ACO) are examples of P-met heuristics, which have been used for solving *n*-queens problem in the literature. For instance, Homaifar *et al.* [8] determined how well the operators of GA handled very difficult combinatorial and constraint satisfaction problems such as the *n*-queens problem. Results are presented for $n<200$. Also, three metaheuristic algorithms (SA, TS and GA) are used to solve the *n*-queens problem by Martinjak and Golub [15]. They presented test results and upper bound complexity for the problem. Many problem instances with large dimensions are solved and the efficiencies of algorithms are compared.

Dirakkhunakon and Suansook [6] compare the results of the classical SA algorithm with Iterative Improvement Simulated Annealing (IISA) algorithm for the *n*-queens problem. The numerical results show that the modified scheme provides better results than the classical algorithm. Khan *et al.* [12] proposed a solution for the *n*-queens problem based on ACO. The proposed solution is applied to 8-queens problem and they supposed that it can very easily be extended to the generalized form of the problem for large values of *n*. Their paper contains detailed discussion of the problem background and complexity, ACO and experimental graphs.

In this paper, the Imperialist Competitive Algorithm (ICA) evolutionary method developed in 2007 is applied for the first time to solve the third variant of the *n*-queens problem, that is, to find the first encountered valid solution. Also, the ICA was combined with a local search, resulting in the Hybrid ICA (HICA) method. Because heuristics are parameter sensitive for finding the best solution, a parameter tuning approach based on the TOPSIS method is proposed to obtain the optimal set of parameters. It follows two goals of reducing the number of Fitness Function Evaluation (FFE) and runtime for solving the problem. Using the tuned parameters, HICA outperformed the original ICA in terms of average runtimes and average number of FFE.

The rest of the paper is organized as follows: Section 2 presents the basic ICA and its components for solving *n*-queens problem, section 3 presents the details of the HICA method, and section 4 provides TOPSIS-based parameter tuning. The numerical results of Design of Experiments to find the best setting of alternative parameters, as well as comparison between the performance of basic ICA and the suggested HICA for various sizes of the problem are presented in section 5. Finally, conclusions are in section 6.

2. The Basic Imperialist Competitive Algorithm

The ICA was first introduced by Atashpaz-Gargari and Lucas [3] as an Evolutionary Computation method based on a social-political evolution. The ICA begins with generating an initial population of 'countries' (counterparts of chromosomes in GAs or particles in PSO). Then, according to a fitness function value, some of the best countries are determined as 'imperialists', and remaining ones as the 'colonies' of these imperialists, which altogether form some 'empires'.

Assimilation and Revolution are the two main operators of this algorithm: The colonies of each empire get closer to its imperialist by the assimilation operator (a concept akin to the recombination operator in other evolutionary algorithms), and random changes happen to the colonies according to the Revolution operator (a concept akin to the mutation operator in other evolutionary algorithms) which may modify the position of colonies in the search space. These operators may improve the solutions of the problem and increase the power of the colonies to take the control of the empire. If so, they swap their positions with their imperialists.

Imperialistic competition among these empires is another part of the ICA, which forms the basis of this evolutionary algorithm. During this competition, powerful empires survive and take possession of the colonies of weaker empires. This procedure eliminates all the imperialists except for one, which yields the final solution. The details of the algorithm's steps tailored for the *n*-queens problem are described below.

2.1. Generating Initial Empires

In the *n*-queens problem, each country is represented by a solution encoded in the form of a permutation $[\pi(1), \pi(2),..., \pi(n)]$, in which the value of $\pi(i)$ indicates the row number and i specifies the column number of a queen on the chessboard as shown in Figure 1. Through this scheme, we can easily generate initial

solutions with no two queens on the same row or column, letting the conflicts occur merely along the diagonals of the chessboard.

The algorithm starts by producing a population of countries, which for the sake of improving the quality of initial solutions, a large number of them are created and then sorted in order of their objective function values to form the initial population with a desired size. From this new list, a number (say N) of them with the highest qualities are considered as imperialists, and the remaining solutions are sequentially assigned to the imperialists as their colonies. In our problem the value of a solution is equal to the number of queen attacks (conflicts) and so lower values mean higher quality.

As an example, assuming that the sorted initial population of size 16 with $N = 3$ imperialists is: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16], the resulting three empires with their imperialists shown in bold will be {[**1**, 4, 7, 10, 13, 16]; [**2**, 5, 8, 11, 14]; [**3**, 6, 9, 12, 15]}.

2.2. Assimilation Within an Empire

In the real political world, imperialists try to promote the life standards of their colonies by assimilating and absorbing them. In the ICA, this fact is simulated by moving each colony toward its respective imperialist. For the assimilation phase, we have utilized two types of Crossover operators: The Partially Matched Crossover (PMX) and Order Crossover (OX).

In PMX operator, two genotypes (solution encodings) are selected as parents, and two crossover positions are picked randomly along the solutions. Then, all chromosomes of Parent A lying between these two points are exchanged with the chromosomes of Parent B at the same positions, and vice versa.

For example, for the 8-queens strings in Figure 2, taking the Parents A and B, the two crossover limits are fixed at 4th and 6th positions, and the dark area indicates the pairs which must undergo exchange. As a result, in both parents, the following swaps take place: 7↔4, 3↔1, and 8↔2, which create two new children.

Now, in our method, the first parent is permanently assumed to be the imperialist solution, and the second parent rotates among all colonies. Thus, the generated offspring will somewhat inherit the nature and power of their imperialist parent, which can be interpreted as a kind of assimilation.

Parent A:	2	4	6	7	3	8	5	1
Parent B:	8	5	3	4	1	2	7	6
Child 1:	8	7	6	4	1	2	5	3
Child 2:	2	5	1	7	3	8	4	6

Figure 2. An example of parents and children in the PMX.

In the OX method, one offspring is generated from two parents. First a substring from the Parent A (which is an imperialist) is selected randomly and an offspring is produced by copying the substring into its corresponding position. Then, these selected elements are deleted from the Parent B (a colony). The resulting sequence contains the elements that the offspring needs. The crossover is finished by placing the remaining elements into the vacant positions of the offspring from left to right, according to the order of their appearance in the Parent B. This procedure is demonstrated in Figure 3.

Parent A:	8	7	2	5	1	4	6	3
Parent B:	2	5	8	7	4	1	3	6
Offspring:	2	8	7	5	1	4	3	6

Figure 3. OX operator illustration.

Regardless of the type of applied crossover, the next generation will be selected from the best solutions of the pool, with the size of the population maintained.

2.3. Revolution Within an Empire

The Revolution operator brings about radical changes in a colony in hope for a better fitness value and also diversifying the population. This unary operator is applied to colonies with a constant rate Revolution Rate (RR) and acts like the mutation operator in GAs.

In our method the Revolution operator is implemented by randomly swapping the values of chromosomes at one or two positions. The colony is updated if a better fitness value is obtained. Figure 4 shows an example of this operator for the 8-queens problem.

Colony (state 0):	8	7	2	5	1	4	6	3
Colony (state 1):	8	7	3	5	1	4	6	2

Figure 4: An example of the revolution operator.

2.4. Power Struggle

While moving toward the imperialist, a colony may achieve a position with lower cost (or equivalently, higher power) than its imperialist. In such a case, the imperialist will be toppled and superseded by that colony. The colony becomes the new imperialist starting from the next iteration. This act is similar to shifting the best global experience (gbest) in the swarm from a particle to another particle in the PSO method.

2.5. Imperialistic Competition

Through the imperialistic competition step, weaker empires lose their power further by losing their colonies, and powerful empires become more powerful by owning new colonies.

The total power of an empire is calculated by adding the power (i.e., fitness function value) of the imperialist country to a percentage of the mean power of its colonies. Mathematically:

$$P(E_i) = P(I_i) + \frac{\xi}{n_i} \sum_{j=1}^{n_i} P(C_i^j) \quad (1)$$

in which $P(E_i)$ is the power of Empire i, $P(I_i)$ is the power of the Imperialist country of Empire i, $P(C_i^j)$ is the power of the j-th colony of Empire i, n_i is the number of colonies in Empire i, and $0 < \xi < 1$ is a constant determining the importance and impact of the colonies in each empire. We found $\xi = 0.1$ a proper value as suggested by Nazari-Shirkouhi *et al*. [16].

For a minimization problem, the normalized total power of Empire i is obtained by subtracting the lowest power among all empires from its power, as in Equation 2. Note that a high power corresponds to a low cost:

$$NP(E_i) = P(E_i) - \min_i \{P(E_i)\} \quad (2)$$

Thus, the normalized total power of the weakest empire will be zero, and for others, a positive value.

The Possession Probability (PP) of each Empire is based on its total power and should be calculated at the start of the imperialistic competition step, according to Equation 3, in which N is the total number of empires:

$$PP_i = \frac{NP(E_i)}{\sum_{j=1}^{N} NP(E_j)} \quad (3)$$

The PP is used to update the distribution of the colonies among the empires. For each empire i, by subtracting a uniform random number $rand_i \in U(0, 1)$ from its PP_i, a new vector is formed, defined as:

$$D = [PP_1 - rand_1, PP_2 - rand_2, ..., PP_N - rand_N] \quad (4)$$

In the vector D, the empire that has the least value among others loses its weakest colony, which is reassigned to the most powerful empire.

The Assimilation, Revolution, and Imperialistic Competition steps are repeated until the weakest empire loses all of its colonies, in which case it is discarded and its imperialist becomes a colony of the most powerful empire.

In our n-queens problem, the stopping criterion is satisfied when there are no conflicts (attacks) among the queens.

3. The Hybrid ICA

As described earlier, the ICA utilizes random numbers in almost all of its steps: Initial population creation, assimilation, revolution, and imperialistic competition. This randomness can be quite effective in diversifying the solutions and adequately exploring the search space. However, we noticed that this fact weakens the algorithm's ability to intensify its search around a good solution, which leads to a slow convergence to a suboptimal solution.

As a result, we decided to add a local search component to the ICA and reinforce its intensification ability. This local search is applied on a solution to improve it as much as possible (i.e., until reaching a local optimum) through a neighborhood generation and selection procedure.

A common method for generating neighbors of a given solution is Random Swap, which exchanges the places of two randomly-selected queens. This action may or may not decrease the number of conflicts among queens. So, to make the neighborhood generation more goal-directed, we propose a new variant of the swap operator, called Effective Swap, which acts more intelligently than the Random Swap since, it selects the exchange rows by also considering the number of attacks rather than just choosing them randomly. The following details illustrate the function of this new operator.

The Effective Swap operator starts with counting the number of conflicts on the main diagonal of the chessboard. If this number is nonzero, it marks that diagonal for further operations. Otherwise, it proceeds with the sub diagonals immediately above and below the main diagonal. Conflict counting is repeated for these diagonals too, and if no conflicts are found, it proceeds with farther sub diagonals parallel to the main diagonal. In case that still no conflicts are identified, the above procedure is repeated for the secondary diagonal and its parallel sub diagonals until a conflicting diagonal is found and marked for further operations.

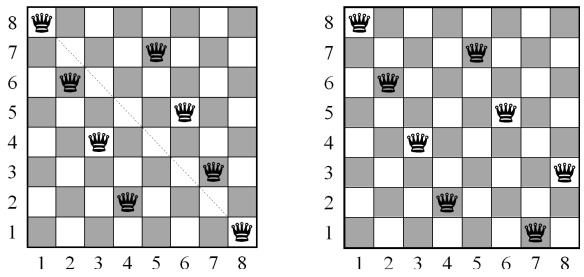
Next, suppose that the marked diagonal has m conflicts. Then, the operator performs m-1 Random Swaps, such that in each swap, one of the queens is selected from the conflicting queens, and the other is a randomly-selected queen not causing any conflict in the marked diagonal. It is worthy to note that performing an Effective Swap does not guarantee an improvement in the fitness function; however, as indicated by our extensive experiments it reduces the number of conflicts far better than the Random Swap operator.

As an example of Effective Swap, consider a configuration of 8-queens displayed in Figure 5-a, where there are m=2 conflicting queens on the marked main diagonal, namely $\pi(1)$ and $\pi(8)$, of which one queen is selected randomly, e.g., $\pi(8)$. Now, another queen which does not cause conflicts in this diagonal is randomly selected, e.g., $\pi(7)$, and the selected rows are swapped by $\pi(7) \leftrightarrow \pi(8)$, as shown in Figure 5-b.

After applying an Effective Swap, a neighbor solution is generated, and we check whether any improvement has occurred in the fitness function or not. If yes, then this neighbor solution is kept; otherwise, a new one is generated. This procedure iterates until a stopping criterion is satisfied.

The stopping criterion contains a parameter T to control the depth of the local search, set by:

$$T = k \cdot n \quad (5)$$



a) Before applying the Effective Swap on the chessboard.

b) After applying the Effective Swap on the chessboard.

Figure 5. Applying an effective swap as a neighborhood generation method.

Where k is a constant and n is the size of the problem. After each iteration of the local search, the value of T is updated by:

$$T = 0.99 \cdot T \tag{6}$$

The local search procedure iterates until T reaches a lower bound like T_{min}. On the other hand, the n-queens problem has multiple optimal solutions (with a fitness function value of zero, meaning no conflicts), and the number of these solutions increases exponentially as n grows. Therefore, if the local search is given more time to transform an initial solution, it can converge to an optimal solution much faster. For this purpose, whenever the newly generated neighbor causes an improvement in the fitness function value, a rewarding mechanism is enforced to update the T by:

$$T = 1.01 \cdot T \tag{7}$$

Note that, the 1.01 coefficient delays the convergence and causes the search to deeply exploit seemingly good solutions. As a result, such a dynamic definition of T causes an effective search of the space, as the algorithm spends more time on exploring an appropriate solution, and less time on non-promising ones.

We name the ICA with the abovementioned local search procedure as "HICA". Figure 6 shows the flowchart of the HICA.

The HICA has another advantage over the basic ICA: As noticed in Equation 4, the empire having the largest value in the vector D will possess the weakest colony of the weakest empire. On the other hand, we know that the most powerful empire (e.g., E^*) has the largest PP index calculated in Equation 3. But, since the vector D is obtained by subtracting random numbers from the PP_i indices, there is no guarantee that the E^* will still be selected for accommodating the weakest colony.

Although, we used the Equation 4 for our basic ICA to keep the authenticity of the algorithm presented by Atashpaz-Gargari and Lucas [3] we discarded the random number subtraction in Equation 4 in the HICA and used the following vector D instead:

$$D = [PP_1, PP_2, ..., PP_N] \tag{8}$$

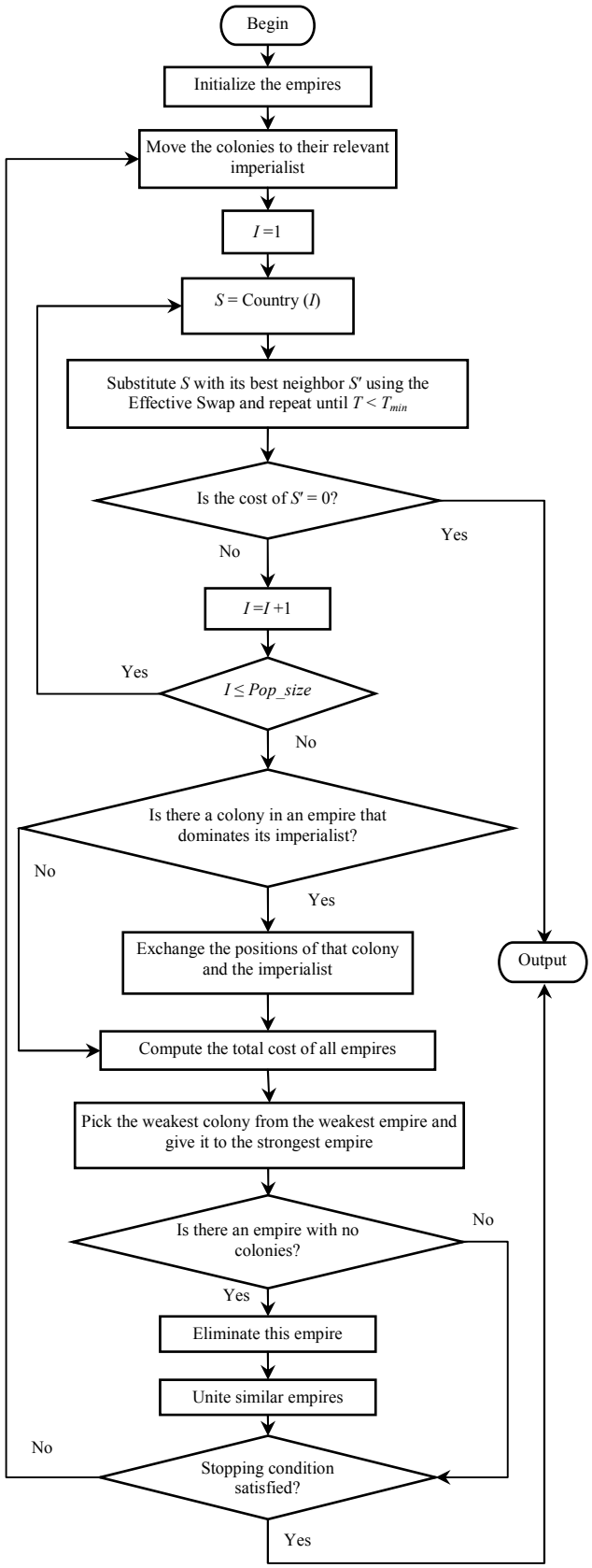


Figure 6. Flowchart of the HICA.

4. Parameter Tuning

As stated before, the parameters of metaheuristic algorithms have a significant effect on the efficiency and effectiveness of the search for a particular problem. There may be many options for these factors

for a given problem. Therefore, using an appropriate approach like parameter tuning in order to find the best choice from many alternatives can produce better solutions for the given problem. In this section, some levels of the parameters for the ICA and HICA are introduced. There are two goals in solving the *n*-queens problem by the proposed algorithms: Reducing the number of FFE and reducing the runtime. The TOPSIS method is used to cope with both objectives at the same time. Finally, the results of the TOPSIS-based parameter tuning are displayed for both algorithms.

4.1. Parameters Levels in ICA

Four factors are considered as the most important parameters in the two algorithms. The first is the initial population which is randomly generated and then 10 solutions are considered as initial solutions, divided into empires and colonies. It's possible levels are {100, 300}. The second is the crossover type in two levels {PMX, OX}. Third is the RR presented in four levels {0.3, 0.4, 0.5, 0.6}, and the fourth parameter (k) is used only in the HICA with the levels {1, 5, 10, 20}.

4.2. TOPSIS

Hwang and Yoon [9] developed the TOPSIS to assess alternatives prior to multiple-attribute decision making. In the TOPSIS, the distance to the ideal solution and negative-ideal solution according to each alternative is considered, and then the best alternative is selected which is the nearest one to the ideal solution and the farthest one from the negative-ideal solution. The TOPSIS structure for aggregating the more important objectives in solving the *n*-queens problem can be explained as follows [22]:

a. *Alternative Performance Matrix Creation*: The structure of the alternative performance matrix can be expressed as follows:

$$D = \begin{array}{c} \\ R_1 \\ R_2 \\ \dots \\ R_m \end{array} \begin{array}{c} \begin{array}{cc} \text{FFE} & \text{Runtime} \end{array} \\ \begin{bmatrix} x_{11} & x_{21} \\ x_{12} & x_{22} \\ \dots & \dots \\ x_{1m} & x_{2m} \end{bmatrix} \end{array} \tag{9}$$

In the proposed problem, the number of FFE and runtime are the objectives (X_j, j = 1, 2) which are related to alternative performances. Possible alternatives (run experiments) are denoted as R_i, i = 1, ..., m; and x_{ij} is the performance of R_i with respect to the objective X_j.

b. *Normalization of the Performance Matrix*: For this purpose, the transformation Equation 10 is used, in which p_{ij} represents the normalized performance of R_i with respect to the objective X_j. The matrix form of p_{ij} is represented as P, with i = 1, 2, ..., m and j = 1, 2.

$$P = \left[p_{ij} \right], \quad p_{ij} = \frac{x_{ij}}{\sqrt{\sum_{i=1}^{m} x_{ij}^2}} \tag{10}$$

c. *Multiplying the Performance Matrix by its Related Weights*: Each column of the matrix P is multiplied by weights associated with each objective FFE (w_{FFE}) and runtime (w_T). The weighted performance matrix V is obtained as follows:

$$V = \begin{bmatrix} w_{FFE}\, p_{11} & w_T\, p_{12} \\ w_{FFE}\, p_{21} & w_T\, p_{22} \\ \dots & \dots \\ w_{FFE}\, p_{m1} & w_T\, p_{m2} \end{bmatrix} = \begin{bmatrix} v_{11} & v_{12} \\ v_{21} & v_{22} \\ \dots & \dots \\ v_{m1} & v_{m2} \end{bmatrix} \tag{11}$$

in which v_{ij} represents the weighted normalized performance of R_i with respect to X_j for i = 1, 2, …, m and j = 1, 2.

d. *Determination of Ideal and Negative-Ideal Solutions*: The ideal value set V^+ and the negative-ideal value set V^- are determined in Equations 12 and 13 for minimizing both objectives simultaneously, in which $J' = \{j = 1, 2 \mid v_{ij}$, a smaller response is desired$\}$:

$$V^+ = \left\{ \left(\min v_{ij} \mid j \in J' \right), i = 1, 2, \dots, m \right\} = \left\{ v_1^+, v_2^+ \right\} \tag{12}$$

$$V^- = \left\{ \left(\max v_{ij} \mid j \in J' \right), i = 1, 2, \dots, m \right\} = \left\{ v_1^-, v_2^- \right\} \tag{13}$$

e. *Calculation of Separation Measures*: The separation of each alternative from the ideal solution (S_i^+), as well as the separation of each alternative from the negative-ideal solution (S_i^-) is given as follows:

$$S_i^+ = \sqrt{\sum_{j=1}^{2} \left(v_{ij} - v_j^+ \right)^2}, \quad S_i^- = \sqrt{\sum_{j=1}^{2} \left(v_{ij} - v_j^- \right)^2} \tag{14}$$

f. *Calculation of Relative Closeness to the Ideal Solution and Ranking the Preference Order*: The relative closeness C_i to the ideal solution can be expressed as follows:

$$C_i = \frac{S_i^-}{S_i^+ + S_i^-} \tag{15}$$

Where C_i lies between 0 and 1. The closer C_i is to 1, the higher is the priority of the i-th run experiment. Because of the multiple levels of each parameter, is $\overline{C_i}$ calculated as the mean of relative closeness to the ideal solution for each parameter per level. Numerical results of using the TOPSIS for the *n*-queens problem with tuned parameter are presented in the next section.

5. Experimental Results

It is clear from the Table 1 that the first two parameters have 2 levels and last two factors have 4 levels. Thus, it is required 2 × 2 × 4 × 4 = 64 experiments for the full factorial design. But, considering the computational cost and time and based on statistical theories, there is no need to test all the combinations of factors. Therefore, the Taguchi method is used to design the experiments. The number of degrees of freedom should be calculated in order to, select an appropriate Taguchi orthogonal array. By considering 1 degree of freedom for the first

two parameters, 3 degrees of freedom for the last two factors and 1 degree of freedom for the total mean, there should be at least $(2 \times 1) + (2 \times 3) + 1 = 9$ experimental runs. As a result, $L_{16}(2^2 \times 4^2)$ is chosen as the orthogonal array. The relative closeness to the ideal solution (C_i) are calculated for each experiment as Table 1.

Table 1. Taguchi orthogonal array $L_{16}(2^2 \times 4^2)$.

No.	Initial Population Size	Crossover Type	Mutation Rate	k	C_i
1	1	1	1	1	0.89
2	1	1	1	2	0.61
3	2	2	1	3	0.31
4	2	2	1	4	0.00
5	1	2	2	1	0.88
6	1	2	2	2	0.45
7	2	1	2	3	0.31
8	2	1	2	4	0.40
9	2	1	3	1	1.00
10	2	1	3	2	0.69
11	1	2	3	3	0.41
12	1	2	3	4	0.24
13	2	2	4	1	0.77
14	2	2	4	2	0.56
15	1	1	4	3	0.38
16	1	1	4	4	0.40
$\overline{C}$	$\overline{C}_1$ **=0.53** $\overline{C}_2$ =0.51	$\overline{C}_1$ **=0.58** $\overline{C}_2$ =0.48	$\overline{C}_1$ =0.60 $\overline{C}_2$ **=0.62** $\overline{C}_3$ =0.59 $\overline{C}_4$ =0.53	$\overline{C}_1$ **=0.89** $\overline{C}_2$ =0.58 $\overline{C}_3$ =0.35 $\overline{C}_4$ =0.26	

For choosing the best level of each parameter, we should find the C_i that is closer to 1. As can be inferred from the Table 1, the parameters of the algorithms were set as follows: Initial population size = 100, Crossover type = PMX, RR = 0.4 and k = 1 (in (5)). The algorithms with tuned parameters were coded in Matlab® and run on an Intel™ Core i7 2.00 GHz CPU with 4.00 GB of RAM.

Tables 2 and 3 show the experimental results of solving the n-queens problem at different sizes. Considering the randomness of the methods, each instance was run 10 times, and the mean and the Standard Deviation (SD) of runtimes and two other performance criteria, the FFE and Normalized Convergence Curve Area (NCCA), are reported.

The FFE criterion measures the total number of FFE during the whole search, and NCCA. The convergence curve plots the best-found fitness function value at each iteration, until the final solution is reached. In the n-queens problem, this curve shows how the algorithm reduces the number of conflicts during its execution till it becomes zero. Figure 7 shows convergence curves of the ICA for various sizes of the problem: n = 50, 100, 200 and 300. The number of conflicts and iterations are displayed along the vertical and horizontal axes, respectively. As can be seen, initial numbers of conflicts were about half the sizes of the problems, and larger problems took much more iterations to converge than smaller instances.

Table 2. Average results of 10 runs of the ICA for various sizes of the n-queens problem.

n	FFE			NCCA	Runtime (s)	
	Min	Max	Avg.		Avg.	SD
8	17	330	159	0.36	0.05	0.06
10	150	2315	785	2.17	0.14	0.13
25	1550	10880	6500	5.40	2.15	1.06
50	12215	116150	4402	10.95	26.48	17.43
100	105870	542720	280014	22.28	348.51	162.39
200	1022990	1882564	1558751	50.15	3284.22	303.54
300	2754111	4258966	3859979	143.51	21650.58	573.81

Table 3. Average results of 10 runs of the HICA for various sizes of the n-queens problem.

n	FFE			NCCA	Runtime (s)	
	Min	Max	Avg.		Avg.	SD
8	0	445	96.3	2.20	0.05	0.05
10	21	940	408.3	11.33	0.14	0.11
30	184	5038	1657.6	13.74	0.67	0.62
50	323	5882	2327.6	11.61	1.20	1.03
75	525	5708	2265.2	11.21	1.28	0.88
100	1374	7006	2932.7	8.81	1.98	1.29
200	6060	9405	8893.6	13.70	9.38	1.10
300	10805	14624	12302.6	12.79	19.60	2.74
500	13717	24906	20962.4	16.47	148.74	29.82
750	23279	42164	33767.5	13.65	616.17	254.26
1000	31701	74877	43272.4	15.80	984.13	301.12
2000	79984	101571	89827.1	21.93	7023.87	545.54

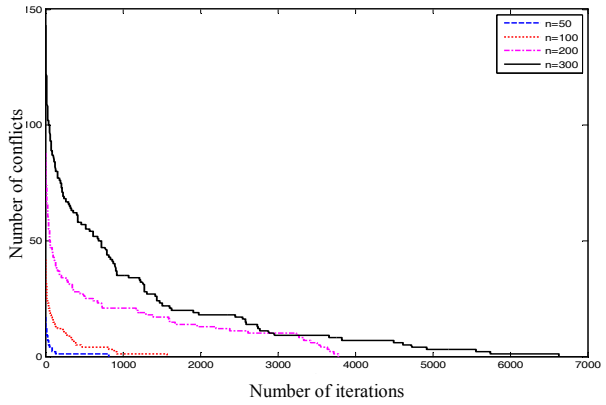


Figure 7. Convergence curves for the ICA run on n = 50, 100, 200, and 300 queens.

Inspired by the behavior of the convergence curve, we designed a new performance criterion to compare the basic and HICA methods: The NCCA. In fact, by calculating the area under a convergence cure we can infer how fast a method reduces the number of conflicts. A relatively small area implies that the algorithm succeeded in reducing the number of conflicts at its early iterations. The NCCA measures the area under the convergence curve with the number of conflicts plotted along the vertical axis and the number of FFE along the horizontal axis; but since for large problem sizes the area becomes too large, we divided it to a factor of n^2 and eliminated the impact of problem size, obtaining a normalized value.

Table 2 shows that the ICA spent about 6 hours of computation averagely for the 300-queens problem,

and so we stopped solving larger instances. On the other hand, the HICA performed surprisingly well and could find a solution to the 2000-queens problem in less than 2 hours. The number of FFE in the HICA method was also significantly less than that of the basic ICA method. For the NCCA criterion the behaviors are a bit different: For small sizes the ICA converges to a low number of conflicts faster than the powerful HICA method, but then for $n > 100$ the HICA regains its superiority (with smaller NCCA index). This fact is due to the impact of the implemented local search on the algorithm's speed.

Figure 8 illustrates the superimposed convergences of the two algorithms, which are plotted for $n = 100$ by considering the best run in terms of convergence speed out of 10 runs. Note, that here the horizontal axis shows the number of FFE's (and not iterations) since the local search component in the HICA executes some additional iterations which should not be compared to the main iterations of ICA.

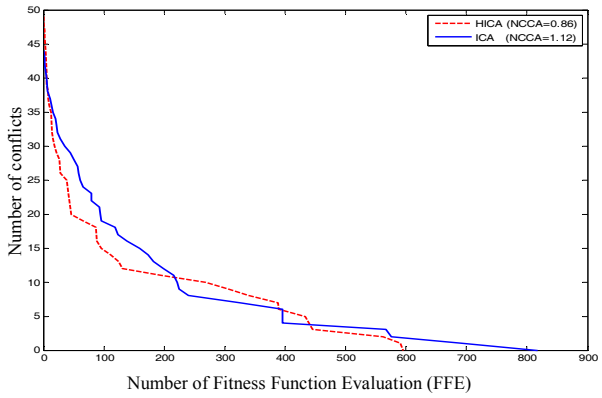


Figure 8. A comparison of convergence curves for basic and hybrid ICAs on $n = 100$ queens.

5.1. Comparisons

In order to evaluate the efficiency of the presented HICA method, we compared it with an algorithm that had produced the best known results in finding the first solution to the n-queens problem. This method is called Cooperative PSO (CPSO) and is introduced in [2] for solving permutation problems, including the n-queens problem. Compared to the PSO method [11], the CPSO uses parallel searching to reduce calculation time.

For solving the n-queens problem by using the CPSO, an initial random population of particles is generated, where each particle has initial information about the locations of n-queens on an $n \times n$ chessboard. Each particle of the population is divided into n equal sub-swarms, and then each sub-swarm is changed into one sub-particle. Sub-particles use the standard PSO to update their velocities and positions according to the best local experience of each sub-particle and the best position for each particle among all particles.

Through a number of experiments, Amooshahi *et al.* [2] compared the CPSO with implementations of standard PSO, SA, TS and GA algorithms reported in [15] and outperformed all those met heuristics in terms of the number of FFE. The results of average FFE values obtained by our proposed HICA and the CPSO algorithms are reported in Table 4 and plotted in Figure 9. It was observed that the HICA always evaluated the fitness function fewer times than the CPSO.

Table 4. Average number of FFEs for HICA and CPSO.

n	HICA	CPSO	Improvement (%)
8	96.3	225.8	57.4
10	408.3	540.5	24.5
30	1657.6	2020.5	18.0
50	2327.6	2764.2	15.8
75	2265.2	3661.6	38.1
100	2932.7	5063.6	42.1
200	8893.6	9184.5	3.2
300	12302.6	14559.6	15.5
500	20962.4	23799.6	11.9
750	33767.5	34765.2	2.9
1000	43272.4	47299.8	8.5
2000	89827.1	95235.9	5.7

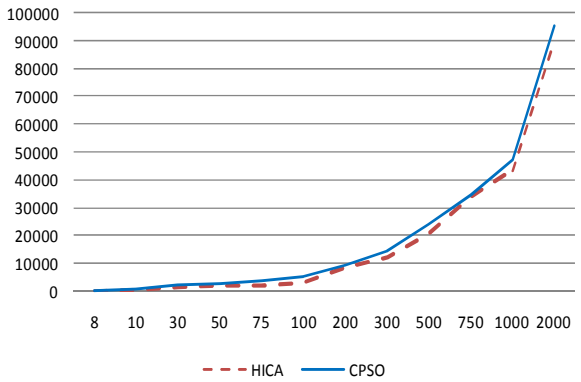


Figure 9. Comparison of the number of FFE versus the problem size for the HICA and CPSO methods.

6. Conclusions

In this paper the ICA, which is a recent evolutionary method, is used for finding the first encountered solution to the n-queens problem. For improving the performance of the algorithm a local search is incorporated into the algorithm, which we call HICA. Due to the effect of the initial factors of metaheuristic on effectiveness of the algorithms, TOPSIS-based parameter tuning is proposed to select the best set of parameters for the algorithm. Experimental result showed that the HICA is able to find the solution for a given number of queens faster than the basic ICA and can solve large instances through smaller numbers of FFE. The HICA was also compared to the best algorithm in the literature for solving this specific problem (i.e., CPSO), and outperformed it in terms of the number of FFE.

As a future work, the RR can be considered as an adaptive parameter such that in initial iterations it takes a relatively large value and decreases as the search proceeds. The decreasing rate would be dynamic and would depend on some information obtained from the

course of the search. As a result, more diversification of solutions in the earlier iterations can be expected, which may lead to faster convergence. Another enhancement could be performing a landscape analysis for the *n*-queens problem, which probably can explain the reason of the significant improvement caused by hybridizing the ICA with a simple local search compared to the basic ICA.

References

[1] Abramson B. and Yung M., "Divide and Conquer under Global Constraints: A Solution to the *n*-Queens Problem," *Journal of Parallel and Distributed Computing*, vol. 6, no. 3, pp. 649-662, 1989.

[2] 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 Proceedings of the 3rd International Conference on Electronics Computer Technology*, Kanyakumari, India, vol. 4, pp. 218-222, 2011.

[3] Atashpaz-Gargari E. and Lucas C., "Imperialist Competitive Algorithm: An Algorithm for Optimization Inspired by Imperialistic Competition," *in Proceedings of IEEE Congress on Evolutionary Computation*, Singapore, pp. 4661-4667, 2007.

[4] Bell J. and Stevens B., "A Survey of Known Results and Research Areas for *n*-Queens," *Discrete Mathematics*, vol. 309, no. 1, pp. 1-31, 2009.

[5] Campos V., Laguna M., and Mart R., "Context-Independent Scatter Search and Tabu Search for Permutation Problems," *INFORMS Journal on Computing*, vol. 17, no. 1, pp. 111-122, 2005.

[6] Dirakkhunakon S. and Suansook Y., "Simulated Annealing with Iterative Improvement," *in Proceedings of International Conference on Signal Processing Systems*, Singapore, pp. 302-306, 2009.

[7] Draa A., Meshoul S., Talbi H., and Batouche M., "A Quantum-Inspired Differential Evolution Algorithm for Solving the *n*-Queens Problem," *the International Arab Journal of Information Technology*, vol. 7, no. 1, pp. 21-27, 2010.

[8] Homaifar A., Turner J., and Ali S., "The *n*-Queens Problem and Genetic Algorithms," *in Proceedings of IEEE Southeastcon*, Birmingham, USA, vol. 1, pp. 262-267, 1992.

[9] Hwang C. and Yoon K., *Multiple Attribute Decision Making-Method and Applications, A State-of-the-Art Survey*, Springer-Verlag, New York, USA, 1981.

[10] Jagota A., "Optimization by Reduction to Maximum Clique," *in Proceedings of IEEE International Conference on Neural Networks*, San Francisco, USA, vol. 3, pp. 1526-1531, 1993.

[11] Kennedy J. and Eberhart R., "Particle Swarm Optimization," *in Proceedings of IEEE International Conference on Neural Networks*, Perth, Australia, vol. 4, pp. 1942-1948, 1995.

[12] Khan S., Bilal M., Sharif M., Sajid M., and Baig R., "Solution of *n*-Queen Problem Using ACO," *in Proceedings of the 13th IEEE International Multi-Topic Conference*, Islamabad, Pakistan, pp. 1-5, 2009.

[13] Kosters W., "*n*-Queens Bibliography," available at: http://www.liacs.nl/~kosters/ nqueens/, last visited 2012.

[14] Lionnet F., NouvellesAnnales de Mathématiques, available at : http://archive.numdam.org/ARCHIVE/NAM/NAM_1869_2_8_/NAM_1869_2_8__529_0/NAM_1869_2_8__529_0.pdf, last visited1869.

[15] Martinjak I. and Golub M., "Comparison of Heuristic Algorithms for the *n*-Queen Problem," *in Proceedings of the 29th International. Conference on Information Technology Interfaces*, Cavtat, Croatia, pp. 759-764, 2007.

[16] Nazari-Shirkouhi S., Eivazy H., Ghodsi R., Rezaie K., and Atashpaz-Gargari E., "Solving the Integrated Product Mix-Outsourcing Problem Using the Imperialist Competitive Algorithm," *Expert Systems with Applications: An International Journal*, vol. 37, no. 12, pp. 7615-7626, 2010.

[17] Pauls E., "Das Maximalproblem Der Damen Auf Dem Schachbrete, II, Deutsche Schachzeitung," *Organ fur das Gesammte Schachleben*, vol. 29, no. 9, pp. 257-267, 1874.

[18] Rivin I. and Zabih R., "A Dynamic Programming Solution to the *n*-Queens Problem," *Information Processing Letters*, vol. 41, no. 5, pp. 253-256, 1992.

[19] San-Segundo P., "New Decision Rules for Exact Search in *n*-Queens," *Journal of Global Optimization*, vol. 51, no. 3, pp. 497-514, 2011.

[20] Sloane N., "The Online Encyclopedia of Integer Sequences," available at: http://oeis.org/A000170, last visited 2012.

[21] Talbi E., *Metaheuristics from Design to Implementation*, John Wiley & Sons, USA, 2009.

[22] Tong L., Wang C., and Chen H., "Optimization of Multiple Responses Using Principal Component Analysis and Technique for Order Preference by Similarity to Ideal Solution," *International Journal of Advance Manufacturing and Technology*, vol. 27, no. 3-4, pp. 407-414, 2005.

[23] Yang X., *Nature-Inspired Metaheuristic Algorithms*, Luniver Press, UK, 2010.

Ellips Masehian is an assistant professor at the Faculty of Engineering, Tarbiat Modares University, Iran. He received his BSc and MSc degrees in industrial engineering, both from Iran University of Science and Technology, Tehran, with honors, and a PhD degree from Tarbiat Modares University in 2004. His research is focused on applications of heuristic, metaheuristic and intelligent methods to combinatorial optimization, as well as single and multiple robot motion planning problems. He has served in Editorial Boards of a number of journals as member, reviewer, and guest editor, and has been in steering and program committees of many international conferences.

Hossein Akbaripour received his BSc degree in engineering in 2010 from the University of Tabriz and a MSc degree in industrial engineering in 2012 from Tarbiat Modares University. He has ranked first in both undergraduate and graduate levels. He is currently a PhD. student at Sharif University of Technology, Iran. He has authored a number of journal and conference papers in the fields of combinatorial optimization and heuristic and metaheuristic algorithms.

Nasrin Mohabbati-Kalejahi received her BSc in industrial engineering from University of Tabriz, Iran, in 2010 and her MSc in industrial engineering from Amirkabir University of Technology, Iran, in 2013 with a thesis titled "Transportation scheduling in supply chain environment with considering the maximum supply of demand". Her research interests include applied operation research, mathematical programming, sequencing and scheduling, heuristic and metaheuristic algorithms, and supply chain management.