Basic and Hybrid Imperialist Competitive Algorithms for Solving the *n*-Queens Problem

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

Keywords: *n*-Queens Problem, Imperialist Competitive Algorithm, Local Search, Effective Swap Operator.

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). Extensive experimental results showed that the proposed HICA outperformed the basic ICA in terms of average runtimes and average number of fitness function evaluations. 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.

1 INTRODUCTION

The *n*-queens problem is a classical combinatorial optimization problem in Artificial Intelligence (Draa et al., 2010). 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 (Erbas et al., 1992); (Sosic and Gu, 1994); (San Segundo, 2011). The variety of these applications indicates the reason of the wide interest on this well-known problem.

Probably the earliest form of the *n*-queens problem was the 8-queens variant, originally proposed in 1848 by the chess player Max Bezzel, published in the German chess newspaper Berliner Schachzeitung (Bezzel, 1848). It was republished in 1850 and attracted the attention of the famous mathematician Carl Friedrich Gauss for finding all possible solution, though he found only 72 of the 92 possible answers. Nauck found all the 92 solutions in the same year (Russel and Norvig, 1995), one of which is shown in Figure 1, with the permutation presented as [5, 1, 8, 4, 2, 7, 3, 6].

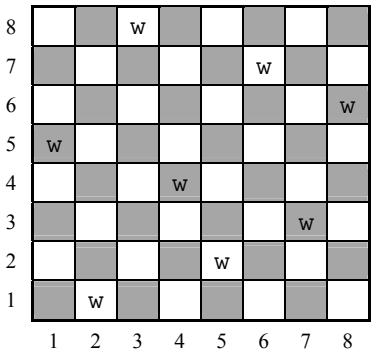

Figure 1: A solution to the 8-queens problem.

The earliest paper on the general *n*-queens problem was presented by Lionnet (1869), and the first proof of the possibility of placing *n* non-attacking queens on an *n*×*n* chessboard is credited to E. Pauls

Masehian E., Mohabbati-Kalejahi N. and Akbaripour H..
Basic and Hybrid Imperialist Competitive Algorithms for Solving the n-Queens Problem.
DOI: 10.5220/0004160900870095
In *Proceedings of the 4th International Joint Conference on Computational Intelligence* (ECTA-2012), pages 87-95
ISBN: 978-989-8565-33-4

(1874). A thorough review on the problem and its applications is presented in (Bell and Stevens, 2009). The *n*-queens problem belongs to the class of Constraint Satisfaction Problems (CSP), and is known as an NP-hard problem (Jagota, 1993). A solution for the 200-queens problem is illustrated in Figure 2.

36	43	170	24	95	140	81	166	146	119
84	107	68	154	111	129	46	106	191	121
133	171	20	88	199	176	50	26	87	178
100	149	49	109	16	159	33	165	136	143
31	44	37	168	45	90	115	74	56	61
114	71	22	3	193	116	162	130	6	98
78	161	96	63	52	192	188	147	183	180
200	174	97	17	131	103	113	62	29	142
157	12	4	155	189	66	60	8	138	190
13	58	41	153	145	7	51	144	34	105
148	196	118	67	91	83	21	169	134	25
195	48	40	179	15	156	197	82	32	172
163	10	19	167	69	11	101	194	89	187
70	35	30	75	141	2	128	151	94	77
186	160	1	112	198	117	123	38	27	14
5	47	39	184	23	139	158	76	110	9
55	42	185	65	135	18	152	126	173	127
181	79	182	57	164	92	53	150	85	125
104	177	28	137	93	124	80	120	99	72
54	132	64	59	86	175	102	122	108	73

Figure 2: A solution to the 200-queens problem.

There are three variants of the *n*-queens problem (Abramson and Yung, 1989): (1) finding all solutions of a given $n \times n$ chessboard, (2) generating one or more, but not all solutions, and (3) 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 2009 (Sloane, 2012). Table 1 shows the size of solution spaces and numbers of valid solutions of various *n*-queens problems.

According to the extensive bibliography of *n*-queens problems in (Kosters, 2012), a wide range of exact, heuristic and metaheuristic optimization methods have been implemented by many researchers (Rivin and Zabih, 1992); (Martinjak and Golub, 2007); (Draa et al., 2005).

The main advantage of metaheuristics compared to exact methods is their ability in handling large-scale instances in a reasonable time (Yang, 2010), 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 fact, a number of papers have implemented metaheuristics for this problem, including Simulated Annealing (SA) (Tambouratzis, 1997); (Dirakkhunakon and Suansook, 2009), Tabu Search (TS) (Martinjak and Golub, 2007), Genetic Algorithms (GA) (Homaifar et al., 1992), Differential Evolution Algorithm (DEA) (Draa et al., 2010), and Ant Colony Optimization (ACO) (Khan et al., 2009).

Table 1: Size of solution space and the number of solutions for the *n*-queens problem solved to date.

n	Size of solution space (*n*!)	Number of solutions
1	1	1
2	2	0
3	6	0
4	24	2
5	120	10
6	720	4
7	5040	40
8	40320	92
9	362880	352
10	3628800	724
11	39916800	2680
12	479001600	14200
13	6227020800	73712
14	87178291200	365596
15	1307674368000	2279184
16	20922789888000	14772512
17	355687428096000	95815104
18	6402373705728000	666090624
19	121645100408832000	4968057848
20	2432902008176640000	39029188884
21	51090942171709440000	314666222712
22	1124000727777607680000	2691008701644
23	25852016738884976640000	24233937684440
24	620448401733239439360000	227514171973736
25	15511210043330985984000000	2207893435808352
26	403291461126605635584000000	22317699616364044

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, which outperformed the original ICA in terms of average runtimes and average number of fitness function evaluations.

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 experimental results on the performance of the basic and Hybrid ICA methods, and provides comparisons with the Cooperative PSO method for various sizes of the problem. Finally, conclusions are in section 5.

2 THE BASIC IMPERIALIST COMPETITIVE ALGORITHM

The Imperialist Competitive Algorithm (ICA) was first introduced by Atashpaz-Gargari and Lucas (2007) 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 entire empire. If so, they swap their positions with their imperialists.

Imperialistic competition among these empires is another part of the ICA algorithm, 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 flowchart of the ICA is illustrated in Figure 3, and 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 (see Figures 1 and 2). 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.

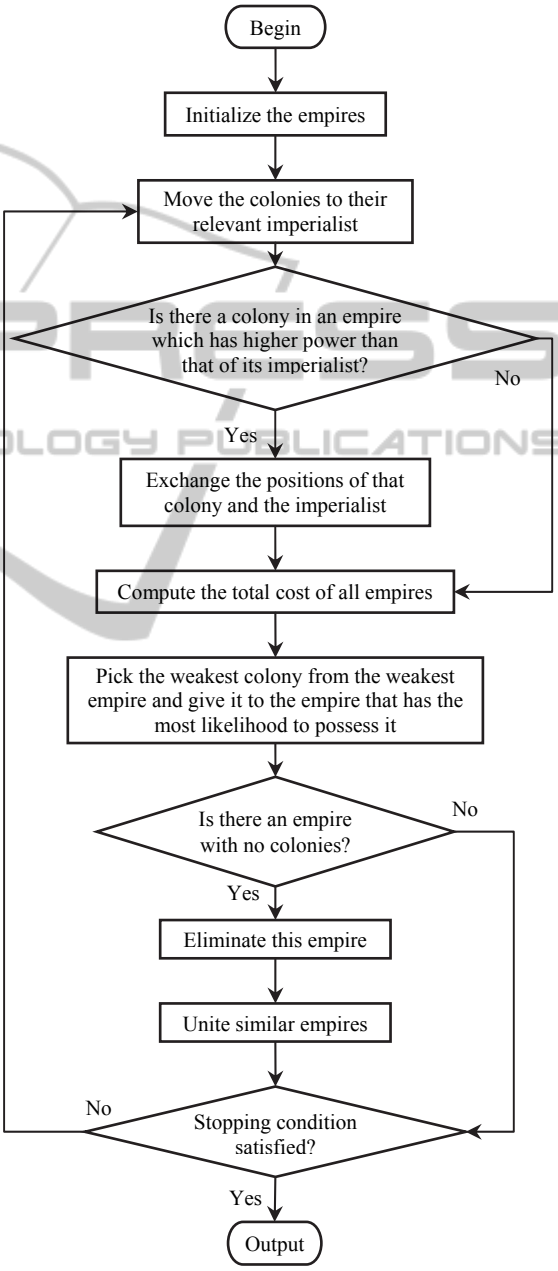

Figure 3: Flowchart of the Imperialist Competition Algorithm.

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 the Partially Matched Crossover (PMX) operator.

In this binary operator, in general, 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 4, 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. The next generation will be selected from the best solutions of the pool, with the size of the population maintained.

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 4: An example of parents and children in the Partially Matched Crossover (PMX).

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 5 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 5: 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, $P(C^j{}_i)$ 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 valueas suggested by Nazari-Shirkouhi et al. (2010).

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 (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 (3), in which N is the total number of empires:

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

The Possession Probability 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:

$$D = [PP_1 - rand_1, PP_2 - rand_2, ..., PP_N - rand_N] \tag{4}$$

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. See Figure 1 for a review of the algorithm. 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 subdiagonals 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 subdiagonals 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 subdiagonals 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 6(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 $\pi(7) \leftrightarrow \pi(8)$, shown in Figure 6(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:

$$T = k \cdot n \tag{5}$$

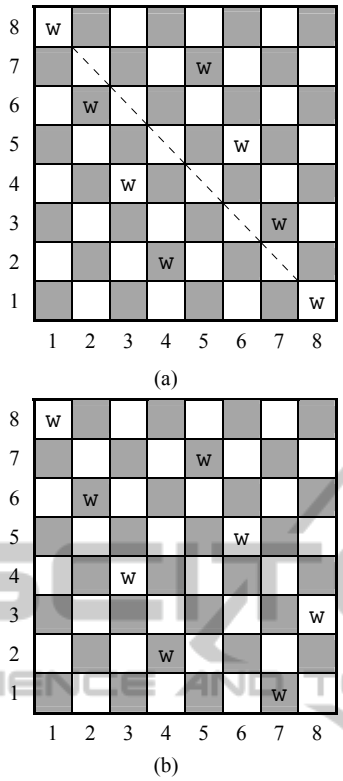

Figure 6: (a) Before, and (b) after applying the Effective Swap on the chessboard.

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:

$$T = 0.99 \cdot T \quad (6)$$

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 (Table 1). 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:

$$T = 1.01 \cdot T \quad (7)$$

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 "Hybrid Imperialist Competitive Algorithm (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 (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 (2007), we discarded the random number subtraction in (4) in the HICA and used the following vector D instead:

$$D = [PP_1, PP_2, ..., PP_N] \quad (8)$$

4 EXPERIMENTAL RESULTS

We conducted a number of experiments to assess the efficiency and effectiveness of the developed algorithms. The parameters of the algorithms were set as follows: Initial population size = 100, k = 1 (in (5)) and Revolution Rate (RR) = 0.4. The algorithms 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 (S.D.) of runtimes and two other performance criteria, the FFE and NCCA, are reported.

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.	S.D.
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

The FFE criterion measures the total number of Fitness Function Evaluations during the whole

search, and NCCA stands for Normalized Convergence Curve Area.

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.	S.D.
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

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.

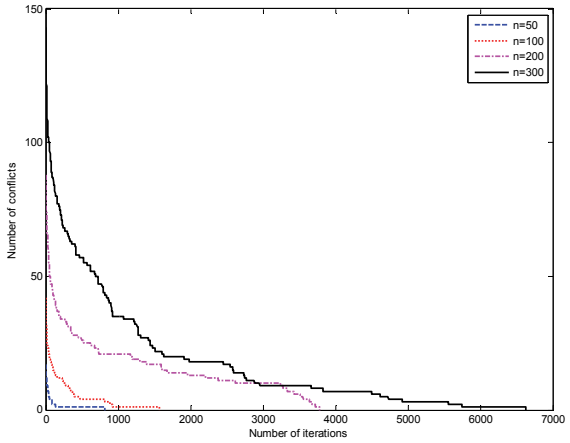

Figure 7: Convergence curves for the ICA run on $n = 50$, 100, 200, and 300 queens.

Inspired by the behavior of the convergence curve, we designed a new performance criterion to compare the basic and hybrid ICA methods: the Normalized Convergence Curve Area, which is calculated as per (9), in which N_c^i is the number of conflicts during *i*-th evaluation of the fitness function:

$$NCCA = \sum_{i=1}^{FFE} N_c^i \tag{9}$$

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 on 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.

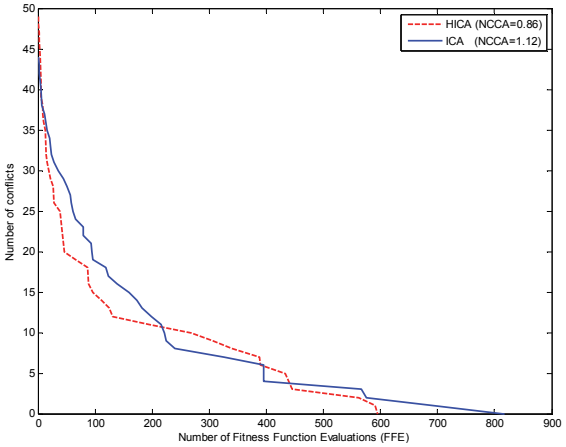

Figure 8: A comparison of convergence curves for basic and hybrid ICAs on $n = 100$ queens.

The curves in Figure 8 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.

4.1 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 (Amooshahi et al., 2011) for solving permutation problems, including the *n*-queens problem. Compared to the standard PSO method (Kennedy and Eberhart, 1995), 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.

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

Through a number of experiments, Amooshahi et al. (2011) compared the CPSO with implementations of standard PSO, SA, TS and GA algorithms (reported in Martinjak and Golub, (2007)) and outperformed all those metaheuristics in terms of the number of fitness function evaluations.

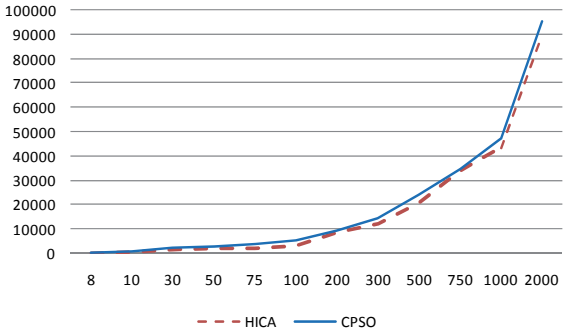

Figure 9: Comparison of the number of fitness function evaluations (FFE) versus the problem size for the HICA and CPSO methods.

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.

We should note here that Minton *et al*. (1990) presented a two-phase method for producing a solution to the *n*-queens problem in a very short time. In the first phase an initial assignment is created via a greedy approach which iterates through the rows and places a queen on a column with minimal conflicts with previously placed queens (ties are broken randomly). In the second phase the assignment is repaired by moving a conflicting queen to a different column in the same row where it conflicts with the least number of queens (ties are broken randomly) until all conflicts are resolved. The reason for not comparing our algorithm with Minton's method was that in their method the number of conflicts after running the assignment phase is dramatically low (averagely about 12.8 conflicts in one million queens!) so that the repairing phase must resolve very few conflicts, while the initial number of conflicts in the ICA or HICA is about half the size of the problem (e.g. 500000 in the one million queen problem). As a result, there was no fair basis for comparing the strength of both methods in reducing the number of conflicts.

5 CONCLUSIONS

In this paper the Imperialist Competitive Algorithm (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 Hybrid ICA (HICA). 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 fitness function evaluations. The HICA was also compared to the best algorithm in the literature for solving this specific problem (i.e., Cooperative PSO), and outperformed it in terms of the number of fitness function evaluations.

As a future work, the Revolution Rate 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

Abramson, B., and Yung, M., (1989). Divide and conquer under global constraints: A solution to the *n*-queens problem. *Journal of Parallel and Distributed Computing*, 6(3), 649-662.

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

Atashpaz-Gargari, E. and Lucas, C., (2007). Imperialist competitive algorithm: An algorithm for optimization inspired by imperialistic competition. *IEEE congress on evolutionary computation*, 4661-4667.

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

Bezzel, M., (1848).Proposal of 8-queens problem, *Berliner Schachzeitung*, 3, 363.

Campos, V., Laguna, M. and Mart, R., (2005). Context-independent scatter search and tabu search for permutation problems. *INFORMS J. Computing*, 17, 111-122.

Dirakkhunakon, S. and Suansook, Y., (2009). Simulated Annealing with iterative improvement. *International Conference on Signal Processing Systems*, 302-306.

Draa, A., Meshoul, S., Talbi, H., and Batouche, M., (2010).A Quantum-Inspired Differential Evolution Algorithm for Solving the *n*-Queens Problem. *The International Arab Journal of Information Technology*, 7(1), 21-27.

Draa, A., Talbi H., and Batouche, M., (2005). A Quantum Inspired Genetic Algorithm for Solving the N-Queens Problem, *Proceedings of the 7th International Symposium on Programming and Systems*, 145-152.

Erbas, C., Sarkeshik, S. and Tanik, M. M., (1992). Different perspectives of the n-queens problem. *Proceedings of the 1992 ACM Annual Conference on Communications, ACM Press*, 99-108.

Homaifar, A., Turner, J., and Ali, S., (1992). The *n*-Queens Problem and Genetic Algorithms. *Proceedings IEEE Southeast Conference*, 1, 262-267.

Jagota, A., (1993). Optimization by reduction to maximum clique. *IEEE International Conference on Neural Networks*, 3, 1526-1531.

Kennedy, J. and Eberhart, R. C., (1995). Particle swarm optimization. *Proceedings of IEEE Int'l. Conf. on Neural Networks*, IV, 1942–1948.

Khan, S., Bilal, M., Sharif, M., Sajid, M. and Baig, R., (2009). Solution of *n*-Queen Problem Using ACO.*IEEE 13th International Multi-Topic Conference,* 1-5.

Kilani, Y., (2010). Comparing the performance of the genetic and local search algorithms for solving the satisfiability problems. *Applied Soft Computing*, 10, 198-207.

Kosters, W., (2012). *n-Queens Bibliography*. Retrieved May 4, 2012, from http://www.liacs.nl/~kosters/nqueens/.

Lionnet, F. J. E., (1869). Question 963, *Nouvelles Annales de Mathématiques*, 28, 560.

Martinjak, I. and Golub, M., (2007). Comparison of Heuristic Algorithms for the N-Queen Problem. *Proceedings of the ITI 2007 29th International. Conference on Information Technology Interfaces*, 25-28.

Minton, S., M. D. Johnston, A. Philips and P. Laird, (1990). Solving Large-Scale Constraint-Satisfaction and Scheduling Problems Using a Heuristic Repair Method. *Proceedings of 8th National Conference on Artificial Intelligence*, Boston, Massachusetts, AIAA Press.

Nazari-Shirkouhi, S., Eivazy, H., Ghodsi, R., Rezaie, K. and Atashpaz-Gargari, E., (2010). Solving the integrated product mix-outsourcing problem using the Imperialist Competitive Algorithm. *Expert Systems with Applications,* 37, 7615-7626.

Pauls, E., (1874). Das Maximalproblem der Damen auf dem Schachbrete, II, Deutsche Schachzeitung.*Organ fur das Gesammte Schachleben*, 29(9), 257-267.

Rivin, I. and Zabih, R., (1992). A Dynamic Programming Solution to the *n*-Queens Problem. *Information Processing Letters*, 41, 253-256.

Russell, S. J. and Norvig, P., (1995).*Artificial Intelligence A Modern Approach.* Prentice-Hall Inc., NJ.

San Segundo, P., (2011). New decision rules for exact search in *n*-Queens. *Journal of Global Optimization*, 51, 497-514.

Sloane, N. J. A., (2012). *The online encyclopedia of integer sequences*.retrieved from http://oeis.org/A000170.

Sosic, R. and Gu, J., (1994). Efficient local search with conflict minimization.*IEEE Transactions on Knowledge and Data Engineering,* (6E), 661-668.

Tambouratzis, T., (1997). A Simulated Annealing Artificial Neural Network Implementation of the *n*-Queens Problem. *Int. J. of Intelligent Systems*, 12, 739-752.

Yang, X-S., (2010). *Nature-inspired metaheuristic algorithms*: Luniver Press.