Multi-Population Parallel Imperialist Competitive Algorithm for Solving Systems of Nonlinear Equations

Amin Majd
Department of Information Technology
University of Turku
Turku, Finland
Amin.Majd@utu.fi

Mahdi Abdollahi
Department of Computer Sciences
University of Tabriz
Tabriz, Iran
m.abdolahi89@ms.tabrizu.ac.ir

Golnaz Sahebi
Department of Information Technology
University of Turku
Turku, Finland
Golnaz.Sahebi@utu.fi

Davoud Abdollahi
Department of Mathematic Sciences
University College of Daneshvaran
Tabriz, Iran
abdollahi_d@daneshvaran.ac.ir

Masoud Daneshtalab
Royal Institute of Technology (KTH)
Stockholm, Sweden,
masdan@kth.se

Juha Plosila
Department of Information Technology
University of Turku
Turku, Finland
juplos@utu.fi

Hannu Tenhunen
Royal Institute of Technology
Stockholm,Sweden
University of Turku, Finland
Hannu@kth.se

***Abstract*— the widespread importance of optimization and solving NP-hard problems, like solving systems of nonlinear equations, is indisputable in a diverse range of sciences. Vast uses of non-linear equations are undeniable. Some of their applications are in economics, engineering, chemistry, mechanics, medicine, and robotics. There are different types of methods of solving the systems of nonlinear equations. One of the most popular of them is Evolutionary Computing (EC). This paper presents an evolutionary algorithm that is called Parallel Imperialist Competitive Algorithm (PICA) which is based on a multi-population technique for solving systems of nonlinear equations. In order to demonstrate the efficiency of the proposed approach, some well-known problems are utilized. The results indicate that the PICA has a high success and a quick convergence rate.**

Keywords— parallel imperialist competitive algorithm (PICA); multi-population technique; evolutionary computing (EC); super linear performance; nonlinear equations; multi-objective optimization;

I. Introduction

Systems of nonlinear equations are one of the NP-Hard problems, which resemble the multi-objective optimization problems. Systems of nonlinear equations are utilized in a range of engineering applications, such as weather forecast, petroleum geological prospecting, computational mechanics, and control fields. The quality of answers of the classical methods, like the Newton-type methods, depends on the initial guess of the solution. However, selecting suitable initial solutions for the most systems of nonlinear equations is extremely difficult.

So far, several methods have been proposed for optimization problems. They can be classified into two major classes: mathematical methods and evolutionary computing (EC) methods. There are different types of EC methods, most of them are implemented in a sequential mode and some in a parallel mode.

Sequential EC methods are more popular than other mathematical methods for solving nonlinear equations, but they do not always provide sufficient accuracy. There are different kinds of parallel EC methods that are capable of improving the accuracy of results. In this work, we utilize a multi-population EC method that improves the results of the used benchmarks.

The rest of the paper is organized as follows: In Section II, the imperialist competitive algorithm is reviewed. Section III introduces the parallel implementation of the ICA based on the multi-population technique. In Section IV, the proposed algorithm is compared with the related previous works. Finally, Section V concludes the paper and indicates the future works.

II. Related Work

Let us first look into the sequential algorithms that have been proposed for solving systems of nonlinear equations. El-Emary and El-Kareem employed Gauss-Legendre integration as a technique to solve the system of nonlinear equations and used genetic algorithm (GA) to discover the results without converting the nonlinear equations to linear equations [15]. Mastorakis employed genetic algorithm (GA) to solve a non-linear equation as well as systems of non-linear equations [17]. Li and Zeng [18], used a neural-network algorithm for solving a set of nonlinear equations. The computation is carried out by a simple gradient descent rule with variable step-size levels [18]. Huan-Tong et al. proposed a modified evolution strategy (ES) based on a probability ranking method to solve complicated nonlinear systems of equations (NSE) problems [19]. M. Abdollahi et al. applied the imperialist competitive algorithm for solving nonlinear systems of equations [16] Ouyang et al. employed a hybrid particle swarm optimization (HPSO) algorithm. The particle swarm optimization (PSO) method focuses on ”exploration”, and the Nelder-Mead simplex method (SM) focuses on ”exploitation” [20], while Wu et al. used a new variation of the social emotional optimization algorithm called MSEOA, mainly inspired by the Metropolis Rule [21]. M. Abdollahi et al. proposed a cuckoo optimization algorithm for solving nonlinear systems equations [25], [29]. Luo at al. applied a combination of the chaos search and Newton type methods [1]. Grosan and Abraham employed a new perspective of the evolutionary algorithms (EA) [7], Mo et al. proposed a combination of the conjugate direction method (CD) [2], and M. Jaberipour used the particle swarm algorithm [3]. Henderson at al. [22], and Pourjafari et al. [23] introduced a methodology based on a polarization technique and a novel optimization method based on Invasive Weed Optimization (IWO), respectively, for finding all roots of a system of nonlinear equations.

In past years, researchers have utilized some parallel EC methods for optimization problems such as Parallel Genetic Algorithms [4], and Parallel PSO [9], Parallel ABC (PABC) [6], Parallel Ant Colony Optimization (PACO) [5], and Parallel Memetic [10] algorithms. Wu and Kang used a parallel elite-subspace evolutionary algorithm for solving systems of nonlinear equations [14]. Some parallel EC methods can achieve super-linear performance [12], where each one is implemented with different techniques and hardware platforms. For example, Parallel Genetic Algorithms are implemented in the following four categories [4]: Master-Slave Genetic Algorithms, Corse Grain Genetic Algorithms (Multi-Populations Genetic Algorithms), Fine Grain Genetic Algorithms, and Hybrid Genetic Algorithms [27].

The obtained results of mathematical methods are sensitive to the initial guess of the solution. The population size of the evolutionary algorithms is large and the convergence of the evolutionary methods to the global minimum is slow. The EC methods are impractical for large-scale problems, like systems of nonlinear equations because of their high linear algebra costs and large memory requirements. For this reason, it is necessary to find an efficient algorithm for solving systems of nonlinear equations. Let systems of nonlinear equations be of the form:

$$\begin{cases} f_1(x_1, x_2, \dots, x_n) = 0 \\ f_2(x_1, x_2, \dots, x_n) = 0 \\ \quad\quad . \\ \quad\quad . \\ \quad\quad . \\ f_n(x_1, x_2, \dots, x_n) = 0 \end{cases} \tag{1}$$

In order to transform (1) to an optimization problem, we will use the auxiliary function:

$$\min f(x) = \sum_{i=1}^{n} f_i^2(x), \qquad x = (x_1, x_2, \dots, x_n) \tag{2}$$

Where $f(x)$ is the global minimum that will be minimized.

In this paper, a parallel imperialist competitive algorithm (PICA) based on the multi-population technique for solving systems of nonlinear equations problems is presented. The proposed method overcomes the mentioned weaknesses of evolutionary methods for solving the systems of nonlinear equations problems. We have also selected some well-known problems for the evaluation.

III. BACKGROUND

The Imperialist Competitive Algorithm (ICA) was introduced by E. Atashpaz and C. Lucas [26], and is inspired by imperialistic competition. ICA is an evolutionary algorithm and optimizes results of problems. In this algorithm, all countries are divided into two types: imperialist states and colonies. Imperialistic competition is the main part of this algorithm, and the expectation is that the colonies converge to the global minimum of the cost function. In the first step, the algorithm creates some countries and after sorting them the best countries are selected to be imperialists and the rest of the countries form the colonies of these imperialists (Fig 1, step 1). After dividing all colonies among imperialists, these colonies start moving toward their relevant imperialist countries (Fig 1, step 2). In the next step, the ICA computes the power of each imperialist and the imperialistic competition begins. The weakest imperialist loses its weakest colony and the selected imperialist captures this colony (Fig 1, step 5). These steps are then repeated until the termination condition is satisfied. The termination conditions can be different. For example, the ICA algorithm could stop after certain number of iterations or when all colonies have become members of one imperialist (see Fig. 1).

ICA is a suitable method for optimization problems, but there exist some challenges concerning the evolutionary algorithms. For example, when we are considering a far-reaching search area, we need a large initial population to obtain an appropriate result, but with a resource constrained processor we may not be able to satisfy this requirement. Also, when we face a complex problem that needs complex computations, the run time will increase, and therefore we need to utilize an efficient method to improve the speed, stability, and accuracy.

The sequential ICA algorithm inherently has a parallel structure, and therefore, a parallel ICA implementation is a viable solution to improve the ICA. In the ICA, each imperialist and colonies work independently, and after a decade a colony moves to another imperialist- Hence, this algorithm works similarly as a multi-population method that works on a processor. In the next section, we utilize a multi-population ICA to solve some complex problems.

IV. THE PROPOSED METHOD

In this work, we utilize a multi-population model to implement the Parallel Imperialist Competitive Algorithm (PICA), by applying a selective local search strategy, in order to solve systems of nonlinear equations. We intend to utilize the full capacity of evolutionary algorithms (e.g. faster convergence, run time speed, and accuracy) for solving such problems. There are different approaches for parallelizing evolutionary algorithms, such as the master slave, multi-population, fine-grain, and hybrid methods; but multi-population method has better convergences and has more accurate results than other parallel methods. Of these, we use the multi-population method. In the multi-population method, there are some independent populations in different processors, each covering its own independent area of the search space. Each processor runs the ICA on its population with independent parameter values. Due to this independency, different levels of exploration and exploitation can be utilized, and therefore the application can get out from local optimums. The other advantage of the multi-population method is its ability for migration, which can significantly improve performance of solving nonlinear equations. It is the most important parallel operation in the multi-population implementation, as letting processors to share the best results and investigate areas with other processors help discover better results in a smaller number of iterations. There are different kinds of migration techniques, for example, each processor can select the best or worst countries to be sent to the other processors or can randomly select some countries. In our work, we select the best countries to be migrated between processors to share the best results together, and the processors replace their worst countries with the received best countries. Receiver processors are selected based on the connection topology used. Figure 1 illustrates the migration behaviour. For example, for a network with the fully connected topology, each processor can receive countries from any other processor. It can be very useful if all processors receive all migrated countries, but in practice we have to find balance between data communication cost (communication time) and achieved improvement in results.

In this work, each country is a possible solution for the selected nonlinear equation. Hence, the algorithm randomly creates the initial populations, which are the possible solutions. Some of them are then selected, at each iteration step, to be processed by the ICA in order to converge them towards better results.

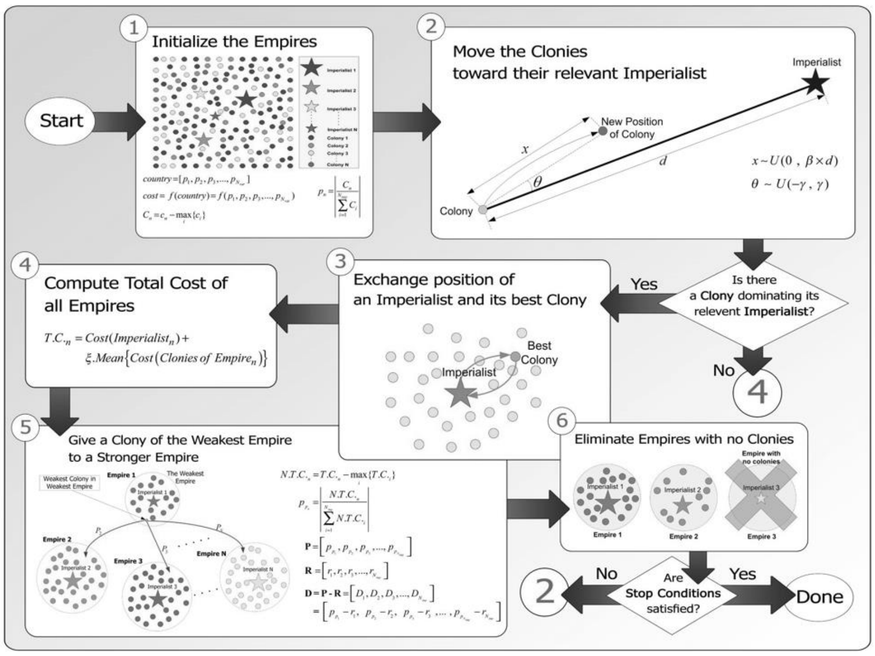
Start
1 Initialize the Empires
2 Move the Clonies toward their relevant Imperialist
Imperialist
New Position of Colony
Colony
$x \sim U(0, \beta \times d)$
$\theta \sim U(-\gamma, \gamma)$
Is there a Clony dominating its relevent Imperialist?
Yes
No
3 Exchange position of an Imperialist and its best Clony
Best Colony
4 Compute Total Cost of all Empires
$T.C._n = Cost(Imperialist_n) + \xi.Mean\{Cost(Clonies\ of\ Empire_n)\}$
5 Give a Clony of the Weakest Empire to a Stronger Empire
6 Eliminate Empires with no Clonies
Are Stop Conditions satisfied?
Done

Fig. 1. Imperialist Competetive Algorithm [28]

In our implementation, several processors are connected together using a ring topology and message passing based communication. The ring topology has been selected because of its low communication cost and simplicity. Each processor is first initialized with a set of independent countries (the number of countries in each processor is the same) and the ICA to be independently run on it. After some decades (the period varies from an execution to another), the best country migrates from each processor *Pi* to the next processor *Pi*+1 in the ring and replaces the worst country in Pi+1. Since we utilize the ring topology to connect processors together, and because the migration takes place in all processors synchronously, the numbers of countries in any two processors are equal at any given time. The migration strategy can affect the result as well. Generally, it is better to establish a balance between the migration rate and data communication. The chosen ring topology is utilized to reduce the migration rate and to decrease the distance of the migrations.

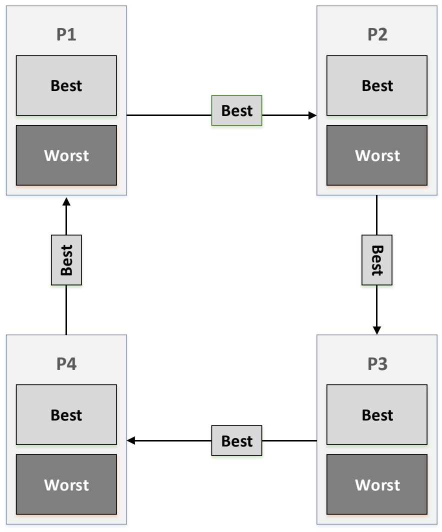

Fig. 2. Multi-population migration operation

Figure 2 shows the architecture of the multi-population structure with a ring topology. Figure 3 presents the Multi-Population ICA pseudo code. In our implementation, all parameters in different processors are equal, and all ICA computations run independently in different processors. On the other hand, the migration operations run synchronously as explained above.

In the multi-population ICA, the pressure of selection increases by growing the number of countries, which helps to obtain more accurate results in a shortest time and converge to the results faster than the sequential ICA. Therefore it is beneficial to increase the number of countries.

```
Processor $P_i$:
1-Create independent initial countries.
2-Run ICA algorithm independently.
3-If now is time of migration do
        a) Wait until all processors arrive to this point.
        b) Send the best country to processor ($P_{i+1}$)mod(#processors).
        c) Receive a country from ($P_{i-1}$)mod(#processors) and
           replace the worse country with the received one.
4-If termination condition is reached, then terminate algorithm.
5-Show the best country.
6-End.
```

Fig. 3. Multi-population ICA pseudo code

V. Experiment and Results

In this section, five commonly explored problems have been utilized to demonstrate the performance of the PICA. The obtained results have been compared with the other well-known methods that have used the same problems. The parallel ICA has been implemented on both share memory and massage passing models. The massage passing interface (MPI) has been utilized to parallelize our algorithm and MPICH2 to run the algorithm.

In the multi-population ICA, processors have been connected in a ring topology with different processors on the different tests. The proposed algorithm has been tested on an Intel core i3-330M, processors 2.13 GHz (64-bit) and memory 4 GB. The best results of benchmarks have been obtained by 30 independent runs. The used parameters for solving the problems have been illustrated in Table 1.

A. Benchmarks

Test 1: 10-dimention Rastrigin Function

$$f(x) = \sum_{i=1}^{10} [x_i^2 - 10\ \cos(2\pi x_i) + 10n] \qquad |x_i| \le 5.2 \tag{3}$$

The answer of this test with f (0, 0, 0,…, 0) is 0. This test has been solved by Mo et al. [2] and ICA [24] with 1000 iterations and 300 population sizes. PICA has been applied to optimize it with the same parameters.

TABLE I. USED PARAMETERS IN PICA FOR TESTS AND CASES

Parameters	Test 1	Test 2	Case 1	Case 2	Case 3
Total population size	300	300	250	250	300
Number of empires	10	10	10	10	10
Number of iteration	1000	1000	45	50	1000
Revolution rate	0.04	0.04	0.04	0.04	0.04
Ξ	0.02	0.02	0.02	0.02	0.02
θ	0.5	0.5	0.5	0.5	0.5
β	2	2	2	2	2

The results of Mo et al. [2], ICA [24] and our proposed algorithm have been presented in Tables 2-4.
Figures 4 and 5 indicate the convergence history of ICA and PICA, respectively. The stability chart of PICA has been indicated in Figure 6. PICA has reached to the optimized answer before 50 iterations.
Test 2: This example has been used as a benchmark in [3] and [24].

$$\min f(x) = \sum_{i=1}^{D} \left[\sin(x_i) + \sin(\frac{2x_i}{3})\right] \tag{4}$$

The answer of the test 2 is 1.21598D and the variables of the function are in (3, 13). The comparison results of ICA [24] and PICA with 1000 iterations and 300 population sizes have been given in Table 5. PICA has solved Test 2 quicker than prior methods before 200 iterations (see Figures 7-9).

B. Case study

In this section, three commonly explored systems of nonlinear equations have been used to demonstrate the performance of the proposed method, and the obtained results have been compared with the other known methods.
Case 1: This example has been given in [3], [11], [24], and [25]:

$$\begin{cases} x_1 - 3x_1x_2^2 - 1 = 0 \\ 3x_1^2x_2 - x_2^3 + 1 = 0 \end{cases} \tag{5}$$

The solutions in [3] and [11] have been obtained with 120 iterations with an unknown number of population sizes. The parameters of the ICA method [24] have been set to 50 iterations with 250 countries. The obtained solutions by PICA are better and more accurate than the previous works (see Table 6). Figures 10-12 indicate the convergence history of the Case 1. Figure 13 shows the stability chart of this case.

Case 2: (Problem 2 in [8], Test Problem 14.1.4 in [13], and Case study in [24] and [25])

$$f_1(x_1, x_2) = 0.5\ \sin(x_1x_1) - 0.25\, x_2/\pi - 0.5x_1 = 0 \tag{6}$$
$$f_2(x_1, x_2) = (1 - 0.25/\pi)\big((\exp(2x_1) - e\big) + e\, x_2/\pi - 2ex_1 = 0$$

The results of Case 2 in [8], [13], [24], and [25] with the 50 iterations and the 250 population sizes were compared with PICA in Table 8. The obtained solutions of PICA have outperformed the mentioned methods with 250 countries and 35 iterations. The speed up of the proposed algorithm gets better than the other literatures (see figures 14-16).

Case 3: (Problem 6 in [8] and Test Problem 14.1.6 in [13]) Case 3 has been solved by the filled function method in [8] and has been proposed as a problem in [13] and [24].

$$4.731 \times 10^{-3}x_1x_3 - 0.357x_2x_3 - 0.1238x_1 + x_7 - 1.637 \times 10^{-3}x_2 - 0.9338x_4 - 0.3 = 0 \tag{7}$$

$$0.2338x_1x_3 + 0.7623x_2x_3 + 0.2638x_1 - x_7 - 0.07745x_2 - 0.6734x_4 - 0.6022 = 0$$

$$x_6x_8 + 0.3578x_1 + 4.731 \times 10^{-3}x_2 = 0$$

$$-0.7623x_1 + 0.2238x_{2_{0.3461}} = 0$$

$$x_1^2 + x_2^2 - 1 = 0$$

$$x_3^2 + x_4^2 - 1 = 0$$

$$x_5^2 + x_6^2 - 1 = 0$$

$$x_7^2 + x_8^2 - 1 = 0$$

$$-1 \le x_i \le 1, \quad i = 1,2,\dots,8.$$

The number of iterations for this problem in [8], [13] and [24] is 1000 and the population size is 300. Our results with the same iterations and countries have been compared in Table 9. The convergence history of ICA [24] and PICA have been shown in Figures 17 and 18, respectively. Figure 19 shows the stability chart of PICA for the Case 3. The statistical results of tests and cases have been illustrated in Table 7. The comparison statistical results of the serial ICA and the parallel ICA have been given in Table 10.

VI. DISCUSSION

In this paper a parallel implementation of ICA based on the multi-population method has been utilized to solve the systems of nonlinear equations. There are different kinds of the PICA implementation such as the master-slave, the multi-population, and the hybrid methods that each one has different advantages. For example, the Master-Slave method can be utilized when we simply intend to increase the speed of our algorithm, but Multi-Population method must be used when we intend to increase both speed and accuracy. Multi-Population method increases the number of the initial population and therefore, the pressure of selection grows that it causes to find more accurate results.

In our implementation, the ring connection topology has been considered to connect the processors. In our algorithm, the migration operation causes that each processor has the ability to send its countries to next processors and receives some countries from previous processors. With the aforementioned ability each processor shares the best results with other processors which reduces the number of iterations considerably.

In this paper, PICA has been compared with other methods through some well-known benchmarks and case studies. PICA has obtained more accurate results with the lower number of iterations.

The most important result of PICA is about super linear performance (where the efficiency value of the algorithm is more than one). Our implementation achieved the super linear performance that means it is an outstanding method and is the best way to solve non-linear problems.

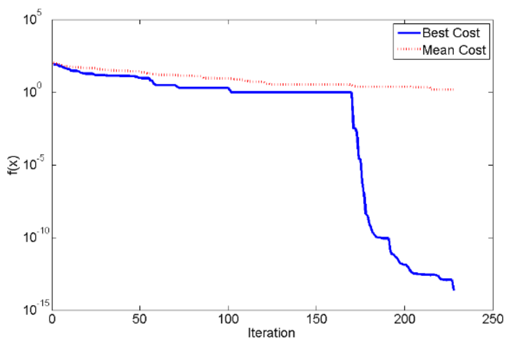

Fig. 4. The convergence history of Rastrigin Function (from [24])

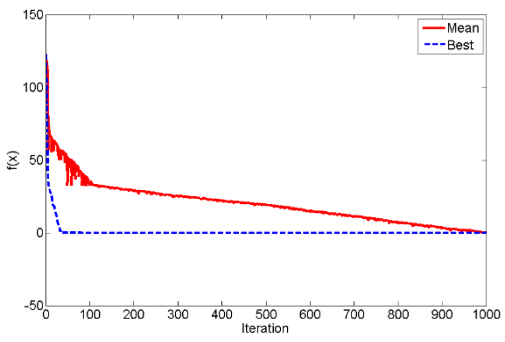

Fig. 5. The convergence history of Rastrigin with PICA (test 1)

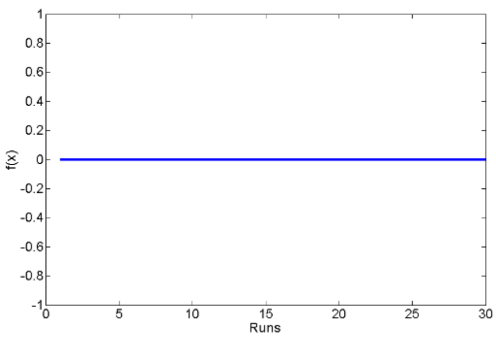

Fig. 6. The stability chart of Rastrigin with PICA (test 1)

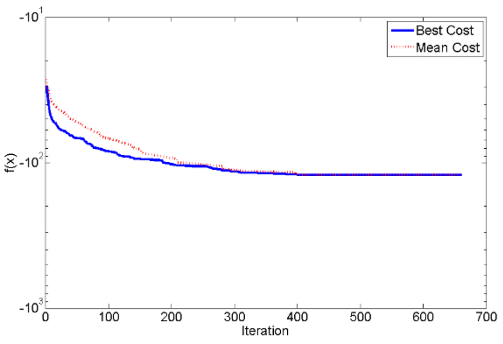

Fig. 7. The convegence history of test 2 with D=100 (from [24])

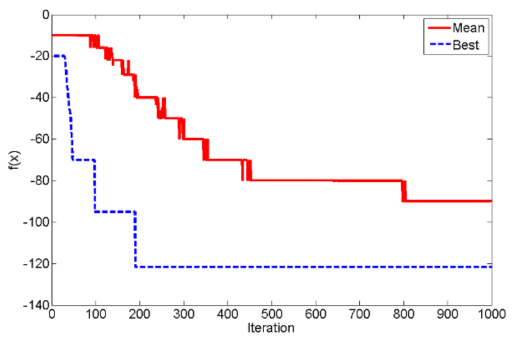

Fig. 8. The convergence history of PICA for test 2 with D=100

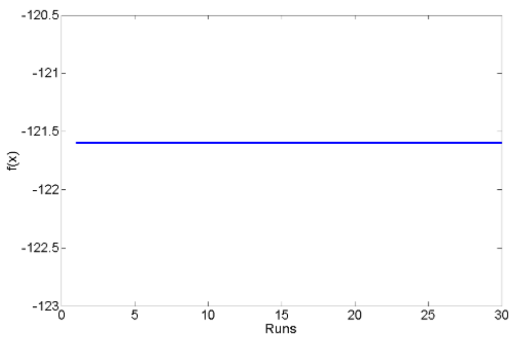

Fig. 9. The Stability chart of test 2 with D=100

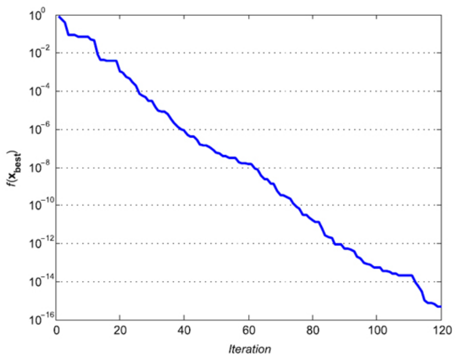

Fig. 10. The convergence history of case 1 (from [3])

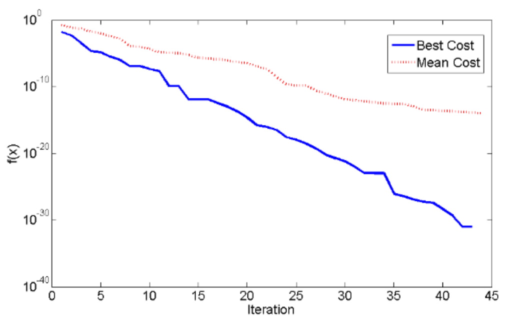

Fig. 11. The convergence history of case 1 (form [24])

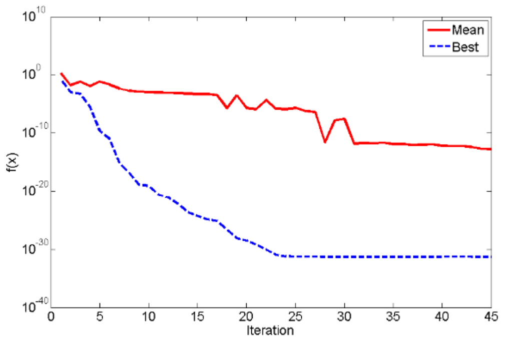

Fig. 12. The convergence history of case 1 with PICA

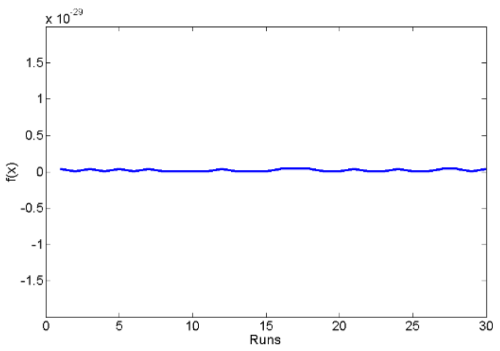

Fig. 13. The stability chart of case 1 with PICA

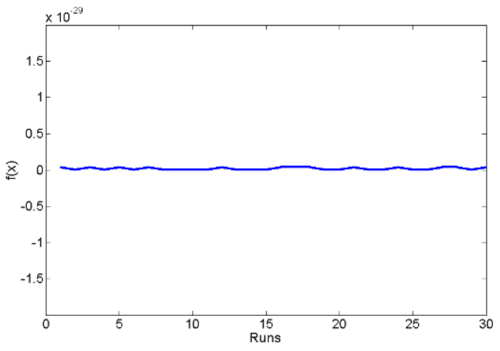

Fig. 14. The stability chart of case 1 with PICA

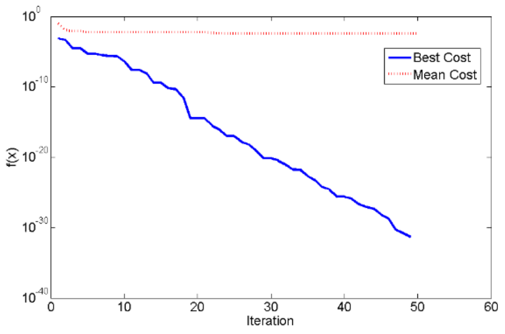

Fig. 15. The convergence history of case 2 (from [24])

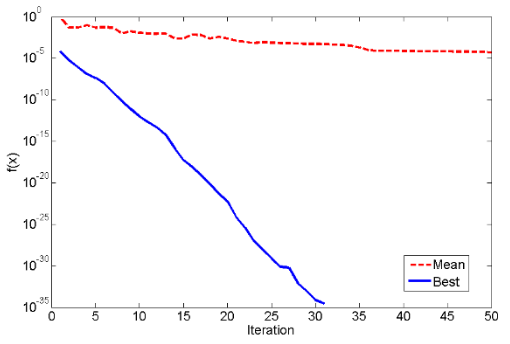

Fig. 16. The convergence history of case 2 with PICA

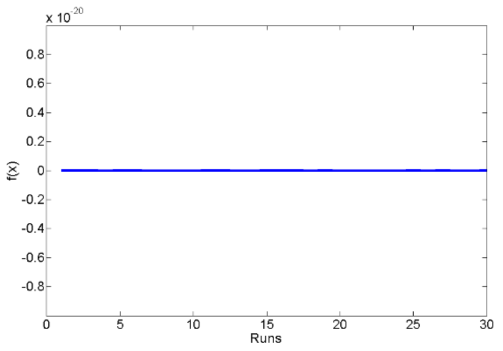

Fig. 17. The stability chart of case 2 with PICA

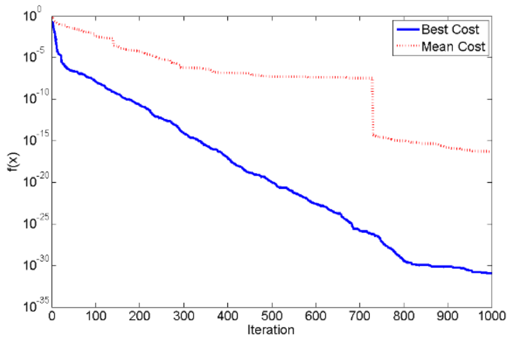

Fig. 18. The convergence history of case 3 (from [24])

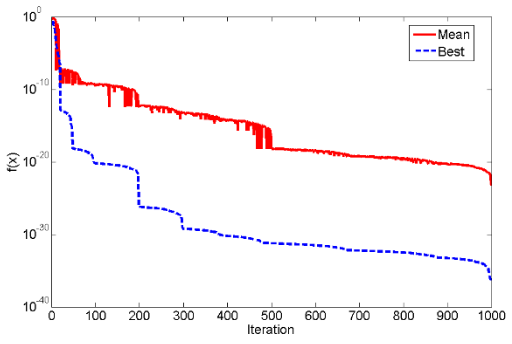

Fig. 19. The convergence history of case 3 with PICA

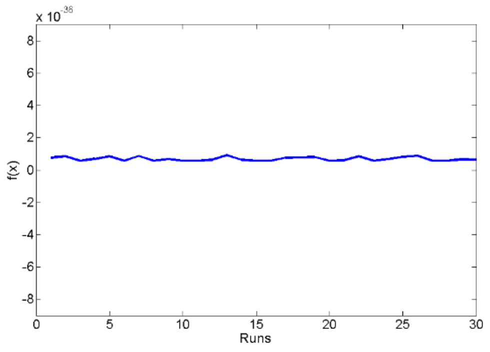

Fig. 20. The stability chart of case 3 with PICA

TABLE II. RESULTS OF TEST 1 WITH MO ET AL. (FROM [2])

Vari ables	Initial iteration	After 200 iterations	After 400 iterations	After 600 iterations	After 800 iterations	After 1000 iteration
x_1	0.1431	-0.0001	-0.0007	0.0001	-0.0000	-0.0000
x_2	2.1983	-0.0001	0.0000	0.0001	0.0001	0.0001
x_3	1.9401	0.0000	0.0000	0.0001	-0.0000	0.0001
x_4	-1.7080	-0.0002	-0.0001	0.0000	-0.0000	-0.0000
x_5	0.2261	-0.9950	-0.9962	-0.9948	0.0001	0.0001
x_6	0.9392	0.9950	0.9941	0.9949	0.9950	0.9949
x_7	-0.1129	0.9949	0.9949	0.0001	-0.0001	0.0000
x_8	-0.1516	0.9950	0.9949	0.9949	0.9949	-0.0000
x_9	-2.1893	-0.0001	0.0000	0.0001	-0.0000	-0.0000
x_{10}	4.9798	0.9950	0.0000	0.0001	-0.0000	-0.0000

TABLE III. RESULTS OF TEST 1 WITH ICA (FROM [24])

Variables	Initial iteration	After 200 iterations	After 400 iterations	After 600 iterations	After 800 iterations	After 1000 iteration
x_1	-2.883527	-0.1084e-007	-0.1404e-008	-0.1404e-008	-0.1404e-008	-0.1404e-008
x_2	-2.111072	0.1710e-007	0.0275e-008	0.0275e-008	0.0275e-008	0.0275e-008
x_3	-0.869045	-0.0048e-007	-0.0656e-008	-0.0656e-008	-0.0656e-008	-0.0656e-008
x_4	1.985114	0.6947e-007	0.0855e-008	0.0855e-008	0.0855e-008	0.0855e-008
x_5	-1.156667	0.0328e-007	-0.1015e-008	-0.1015e-008	-0.1015e-008	-0.1015e-008
x_6	3.083374	0.0356e-007	0.0899e-008	0.0899e-008	0.0899e-008	0.0899e-008
x_7	3.093877	-0.0948e-007	0.0349e-008	0.0349e-008	0.0349e-008	0.0349e-008
x_8	-2.020172	0.1528e-007	0.1610e-008	0.1610e-008	0.1610e-008	0.1610e-008
x_9	2.832951	-0.2304e-007	-0.0180e-008	-0.0180e-008	-0.0180e-008	-0.0180e-008
x_{10}	-2.208695	-0.2454e-007	0.0147e-008	0.0147e-008	0.0147e-008	0.0147e-008
$f(x)$	83.041615	1.3287e-012	0	0	0	0

TABLE IV. RESULTS OF TEST 1 WITH PICA (PRESENT STUDY)

Variables	Initial iteration	After 200 iterations	After 400 iterations	After 600 iterations	After 800 iterations	After 1000 iteration
x_1	1.928441	-4.716237e-011	-4.716237e-011	-4.716237e-011	-4.716237e-011	-4.716237e-011
x_2	-2.248101	2.157231e-011	2.157231e-011	2.157231e-011	2.157231e-011	2.157231e-011
x_3	1.341671	-1.001342e-011	-1.001342e-011	-1.001342e-011	-1.001342e-011	-1.001342e-011
x_4	0.728811	1.713127e-011	1.713127e-011	1.713127e-011	1.713127e-011	1.713127e-011
x_5	1.728128	-7.887191e-011	-7.887191e-011	-7.887191e-011	-7.887191e-011	-7.887191e-011
x_6	-2.839121	-2.837190e-012	-2.837190e-012	-2.837190e-012	-2.837190e-012	-2.837190e-012
x_7	1.871831	1.238291e-011	1.238291e-011	1.238291e-011	1.238291e-011	1.238291e-011
x_8	1.934281	-6.348271e-011	-6.348271e-011	-6.348271e-011	-6.348271e-011	-6.348271e-011
x_9	1.409124	8.119381e-011	8.119381e-011	8.119381e-011	8.119381e-011	8.119381e-011
x_{10}	0.365281	1.981381e-011	1.981381e-011	1.981381e-011	1.981381e-011	1.981381e-011
$f(x)$	1.241803e+002	0	0	0	0	0

TABLE V. COMPARISON RESULTS OF TEST 2 WITH D=100

$f(x)$	Initial iteration	After 100 iteration	After 200 iteration	After 300 iteration	After 400 iteration	After 500 iteration	After 600 iteration	After 700 iteration	After 800 iteration	After 900 iteration
ICA [24]	29.786871	-78.6467	-103.3897	-113.0125	-120.5298	-121.5923	-121.5979	-121.5982	-121.5982	-121.5982
PICA	-20.0000	-95.0017	**-121.5982**	-121.5982	-121.5982	-121.5982	-121.5982	-121.5982	-121.5982	-121.5982

TABLE VI. COMPARISON RESULTS OF PICA FOR CASE 1 WITH [3], [11], [24] AND [25]

Methods	x_1	x_2	$f(x)$
PPSO [3] and Gyurhan [11]	-0.29051455550725	1.08421508149135	4.686326815078573e-029
PPSO [3] and Gyurhan [11]	-0.793700525984100	-0.793700525984100	1.577721810442024e-030
COA [25]	1.08421508149135	-0.29051455550725	4.686326815078573e-029
COA [25]	-0.29051455550725	1.08421508149135	4.686326815078573e-029
ICA [24]	1.084215081491351	-0.290514555507251	3.562200025138631e-030
ICA [24]	-0.793700525984100	-0.793700525984100	1.577721810442024e-030
ICA [24]	-0.290514555507251	1.084215081491351	3.562200025138631e-030
PICA (present study)	1.0842150814913511	-0.2905145555072514	4.9303806576313238e-032
PICA (present study)	-0.79370052598409995582	-0.79370052598409995582	3.9443045261050590e-031
PICA (present study)	-0.2905145555072514	1.0842150814913511	4.9303806576313238e-032

TABLE VII. STATISTICAL RESULTS

Problem	N	Mean	Std. Deviation	Std. Error Mean	Worst	Best
Test 1	30	0.0	0.0	0.0	0.0	0.0
Test 2	30	-1.215981999999999e+002	7.226896453227138e-014	1.319444736062194e-014	-1.215982000000000e+002	-1.215982000000000e+002
Case 1	30	1.988586000000001e-031	1.739458967563944e-031	3.175803047964731e-032	3.944300000000000e-031	4.930400000000000e-032
Case 2	30	1.046312443884771e-026	1.511543708264576e-026	2.759688618905053e-027	4.414500000000001e-026	0.0
Case 3	30	6.898049999999997e-037	1.150107106181705e-037	2.099798685342363e-038	9.039099999999999e-037	5.800000000000000e-037

TABLE VIII. COMPARISON RESULTS OF CASE 2

Methods	X	Variable values	f	Functions values	F(x)
The best in [8]	x_1	0.500432850000000	f_1	-0.000238520000000	7.693745216994211e-008
	x_2	3.141863170000000	f_2	0.000141590000000	
The best in [13]	x_1	0.299450000000000	f_1	6.139739265609290e-007	1.014347133848949e-012
	x_2	2.836930000000000	f_2	-7.983627943186633e-007	
	x_1	0.500000000000000	f_1	2.111655261760603e-007	5.316365008296489e-012
	x_2	3.141590000000000	f_2	-2.296034435467220e-006	
The best in COA [25]	x_1	0.299300000000000	f_1	-7.128922385554737e-005	5.792081721117691e-009
	x_2	2.836600000000000	f_2	2.664447941302939e-005	
The best in ICA [24]	x_1	0.299448692495720	f_1	1.305289210051797e-012	5.631272867601562e-024
	x_2	2.836927770471037	f_2	2.284838984678572e-013	
	x_1	0.500000000000000	f_1	0	0
	x_2	3.141592653589794	f_2	0	
The best of PICA	x_1	0.29944869249092598	f_1	-1.387778780781446e-016	6.856310602018560e-032
	x_2	2.8369277704589400	f_2	2.220446049250313e-016	
	x_1	0.500000000000000	f_1	0	0
	x_2	3.141592653589794	f_2	0	

TABLE IX. COMPARISON RESULTS OF CASE 3

Methods	x	Variables values	f	Functions values
The best in [8]	x_1	0.67154465	f_1	-0.00000375
	x_2	0.74097111	f_2	0.00001537
	x_3	0.95189459	f_3	0.00000899
	x_4	-0.30643725	f_4	0.00001084
	x_5	0.96381470	f_5	0.00001039
	x_6	-0.26657405	f_6	0.00000709
	x_7	0.40463693	f_7	0.00000049
	x_8	0.91447470	f_8	-0.00000498
The best in [13]	x_1	0.1644	f_1	-8.8531e-005
	x_2	-0.9864	f_2	3.5894e-005
	x_3	-0.9471	f_3	6.6216e-006
	x_4	-0.3210	f_4	2.1560e-005
	x_5	-0.9982	f_5	1.2320e-005
	x_6	-0.0594	f_6	3.9410e-005
	x_7	0.4110	f_7	-6.8400e-005
	x_8	0.9116	f_8	-6.4440e-005
The best of ICA [24]	x_1	0.164431665854327	f_1	2.775557561562891e-016
	x_2	-0.986388476850967	f_2	-1.110223024625157e-016
	x_3	0.718452601027603	f_3	-1.110223024625157e-016
	x_4	0.718452601027603	f_4	1.734723475976807e-018
	x_5	0.997964383970433	f_5	0
	x_6	0.063773727557003	f_6	0
	x_7	-0.527809105283546	f_7	0
	x_8	-0.849363025083964	f_8	0
The best of PICA	x_1	0.164431665854327405	f_1	5.368529659036217811e-019
	x_2	-0.986388476850967110	f_2	2.548307417523678423e-019
	x_3	0.718452601027603350	f_3	-3.378192205891815512e-019
	x_4	-0.695575919707310931	f_4	3.389211820587187123e-019
	x_5	0.997964383970432520	f_5	0
	x_6	0.063773727557002571	f_6	0
	x_7	-0.527809105283546241	f_7	0
	x_8	-0.849363025083964123	f_8	0

TABLE X. THE COMPARISON STATISTICAL RESULTS OF SERIAL ICA [24] AND PICA

Problem	Speed Up	Efficiency	Serial ICA time	PICA time	#processors	Super linear performance
Test 1	4.02	2.01	3.38	0.84	2	Yes
Test 2	7.22	3.61	11.82	1.63	2	Yes
Case 1	2.82	1.41	0.0341	0.012	2	Yes
Case 2	5.1	2.55	2.1	0.411	2	Yes
Case 3	6.24	3.12	6.78	1.08	2	Yes

VII. CONCLUSION AND FUTURE WORKS

In this paper, the parallel imperialist competitive algorithm based on the MPI instructions (Multi-Population) was utilized to solve the systems of nonlinear equations. The PICA was compared with the serial ICA and some of the other proposed methods. According to the obtained results, the PICA is suitable for solving different kinds of complex problems, and it is faster and more efficient than the other methods. The figures indicated that the answers of our algorithm are stable and the convergence of the PICA to the best solution is faster than the other methods, with the lower number of iterations and better run time. As a result, we claim that the proposed PICA is a faster and more accurate method, which can be employed to solve and improve the complex problems. At the end, our future works will consist of using the proposed parallel algorithm to solve some of the more practical optimization problems, like constrained engineering optimizatio

REFERENCES

[1] Y.Z. Luo, G.J. Tang, L.N. Zhou, Hybrid approach for solving systems of nonlinear equations using chaos optimization and quasi-Newton method, Appl. Soft. Comput. 8 (2008) 1068-1073.J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[2] Y. Mo, H. Liu, Q. Wang, Conjugate direction particle swarm optimization solving systems of nonlinear equations, Comput. Math. Appl. 57 (2009) 1877-1882.

[3] M. Jaberipour, E. Khorram, B. Karimi, Particle swarm algorithm for solving systems of nonlinear equations, Comput. Math. Appl. 62 (2011) 566-576.

[4] E. Cantu-Paz, A Survey of Parallel Genetic Algorithms, Department of Computer Science and Illinois Genetic Algorithms Laboratory University of Illinois at Urbana-Champaign, 1997.

[5] H. Liu, P. Li, and Y. Wen, Parallel Ant Colony Optimization Algorithm, World Congress on Intelligent Control and Automation, China, June, 2006.

[6] R. Parpinelli, C. Benitez, and S. Lopes, Parallel Approaches for the Artificial Bee Colony Algorithm, Handbook of Swarm Intelligence, 8(2010) 329-345.

[7] C. Grosan, A. Abraham, A New Approach for Solving Nonlinear Equations Systems, IEEE Trans. Syst. Man Cybern. A 38 (3) (2008) Senior Member, IEEE.

[8] C. Wang, R. Luo, k. Wu, B. Han, A new filled function method for an unconstrained nonlinear equation, Comput. Appl. Math. 235 (2011) 1689-1699.

[9] L. Vanneschi, D. Codecasa, and G. Mauri, A Comparative Study of Four Parallel and Distributed PSO Methods, New Generat. Comput. 29(2011) 129-161.

[10] J. Digalakis, and K. Margaritis, A Parallel Memetic Algorithm for Solving Optimization Problems, 4th Metaheuristics International Conference, Parallel Distributed Processing Laboratory, Greece, 2001.

[11] Gyurhan H. Nedzhibov, A family of multi-point iterative methods for solving systems of nonlinear equations, J. Comput. Appl. Math. 222(2008) 244250.

[12] E. C. G. Wille, E. Y. H. S. Lopes, Discrete Capacity Assignment in IP networks using Particle Swarm Optimization, Appl. Math. Comput.,217 (2011) 5338-5346.

[13] C. A. Floudas, P. M. Pardalos, C. S. Adjiman, W. R. Esposito, Z. H. Gumus, S. T. Harding, J. L. Klepeis, C. A. Meyer, C. A. Schweiger, Handbook of Test Problems in Local and Global Optimization, Kluwer Academic Publishers, Dordrecht, the Netherlands, 1999.

[14] A. Mousa, W. Wahed, R. Allah, A Hybrid Ant Colony Optimization Approach Based Local Search Scheme for Multi Objective Design Optimizations, Electr. Pow. Syst. Res. 81 (2011) 1014-1023.

[15] Ibrahiem M. M. El-Emary and Mona M. Abd El-Kareem, Toward Using Genetic Algorithm for Solving Nonlinear Eqution Systems, World Appl. Sci. J. 5 (2008) 282-289.

[16] M. Abdollahi, A. Isazadeh, D. Abdollahi, Solving systems of nonlinear equations using imperialist competitive algorithm, The 8th International Industrial Engineering Conference, 8 (2012) 1-6.

[17] Nikos E. Mastorakis, Solving Non-linear Equations via Genetic Algorithms, Proceedings of the 6th WSEAS Int. Conf. on EvolutionaryComputing, Lisbon, Portugal, June 16-18 (2005) 24-28.

[18] G. Li, Zh. Zeng, A neural-network algorithm for solving nonlinear equation systems, IEEE International Conference on Computational Intelligence and Security, CIS08 (2008) 20-23.

[19] G. Huan-Tong, S. Yi-Jie, S. Qing-Xi, W. Ting-Ting, Research of Ranking Method in Evolution Strategy for Solving Nonlinear System of Equations, IEEE International Conference on Information Science and Engineering, ICISE09 (2009) 348-351.

[20] A. Ouyang, Y. Zhou, Q. Luo, Hybrid Particle Swarm Optimization Algorithm for Solving Systems of Nonlinear Equations, IEEE International Conference on Granular Computing, GRC09 (2009) 460-465.

[21] J. Wu, Zh. Cui, J. Liu, Using Hybrid Social Emotional Optimization Algorithm with Metropolis Rule to Solve Nonlinear equations, IEEE International Conference on Cognitive Informatics & Cognitive Computing, ICCI*CC'11 (2011) 405-411.

[22] N. Henderson, W. F. Sacco, G. Mendes Platt, Finding more than one root of nonlinear equations via a polarization technique: An application to double retrograde vaporization, Chem. Eng. Res. Des. 88 (2010) 551-561.

[23] E. Pourjafari, H. Mojallali, Solving nonlinear equations systems with a new approach based on invasive weed optimization algorithm and clustering, Swarm Evol. Comput. 4 (2012) 3343.

[24] M. Abdollahi, A. Isazadeh, D. Abdollahi, Imperialist competitive algorithm for solving systems of nonlinear equations, Comput. Math. Appl.65 (2013) 1894-1908.

[25] M. Abdollahi, Sh. Lotfi, D. Abdollahi, Solving systems of nonlinear equations using cuckoo optimization algorithm, The 3rd International conference on The Contemporary Issues in Computer Sciences and Information Technology (CICIS), 3 (2012) 191-194.

[26] E. Atashpaz-Gargari, C. Lucas, Imperialist competitive algorithm: an algorithm for optimization inspired by imperialistic competition, in:IEEE Congress on Evolutionary Computation, 2007, pp. 46614667.

[27] A. Majd, Sh. Lotfi and G. Sahebi, "Review on Parallel Evolutionary Computing and Introduce Three General Framework to Parallelize All EC Algorithms," 5th Conference on Information and Knowledge Technology (IKT), 2013.

[28] A. Majd, Sh. Lotfi, G. Sahebi, M. Daneshtalab and J. Plosila, "PICA: Multi-Population Implementation of Parallel Imperialist Competitive Algorithms," 24th Euromicro International Conferences on Parallel, Distributed and Network-Based Processing, PDP 2016.

[29] M. Abdollahi, A. Bouyer, D. Abdollahi, Improved cuckoo optimization algorithm fo solving systems of nonlinear equations, J. Supercomput. 72 (2016) 1246-1269.