International Journal of Hybrid Information Technology
Vol.8, No.12 (2015), pp. 433-448
http://dx.doi.org/10.14257/ijhit.2015.8.12.34

A Double Mutation Cuckoo Search Algorithm for Solving Systems of Nonlinear Equations

Chiwen Qu* and Wei He

Department of Mathematics & Computer Information Engineering, Baise University, Baise 533000, China
quchiwen@163.com

Abstract

This paper presents a double mutation cuckoo search algorithm (DMCS) to overcome the disadvantages of traditional cuckoo search algorithms, such as bad accuracy, low convergence rate, and easiness to fall into local optimal value. The algorithm mutates optimal fitness parasitic nests using small probability, which enhances the local search range of the optimal solution and improves the search accuracy. Meanwhile, the algorithm uses large probability to mutate parasitic nests in poor situation, which enlarges the search space and benefits the global convergence. The experimental results show that the algorithms can effectively improve the convergence speed and optimization accuracy when applied to basic test functions and systems of nonlinear equations.

Keywords: *Cuckoo Search Algorithm, Double Mutation, Systems of Nonlinear Equations*

1. Introduction

Solving systems of nonlinear equations is a vital problem that we often encounter in engineering field, such as in computational mechanics, geological prospecting, and engineering optimization. Many researchers have conducted investigations on this issue [1, 2]. Bader [3] and Luo, *et al.,* [4] solved this problem using a Newton method and a combination of chaos search and Newton-type, respectively. However, as the convergence and performance characteristics are sensitive to the initial guess and the object functions are needed to be continuously differentiable, the Newton method will lose effect when the initial guess of the solution is improper or the targeted function is not continuously derivable. Some scholars solved this issue using intelligent swarm algorithms. Jaberipour, *et al.*, [5] and Mo, *et al.,* [6] solved a system of nonlinear equations using a particle swarm optimization method, a combination of the conjugate direction method (CD) and particle swarm optimization, respectively. Literature [7] putted forward a new method using leader glowworm swarm optimization algorithm. Literature [8] proposed an imperialist competitive algorithm for solving systems of nonlinear equations. However, the basic intelligent swarm algorithms are easy to fall into local optimal value and have low accuracy. Therefore, it is necessary to develop an efficient algorithm with high optimization accuracy and the ability to jump out to local optimum. Generally, a system of nonlinear equations can be expressed as follows:

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

Obviously, the solution of the system of nonlinear equations can be transformed into the problem of solving the minimum of the following function:

$$\min \; f(X) = \sum_{i=1}^{n} | f_i(x_1, x_2, ..., x_n) | \qquad (1)$$

where x is the solution of the systems of nonlinear equations.

As a heuristic intelligent swarm algorithm, the cuckoo search algorithm was presented by the British scholar Yang and Deb, which is based on the spawning habits of the cuckoo in the nest and search process in 2009. The algorithm is simple and of few parameters to be set. Furthermore, optimization accuracy and the rate of convergence are better than those of PSO and genetic algorithms [11, 12]. Over the past few years, this algorithm becomes a new research hotspot of computing intelligence. The main applications of the cuckoo search algorithm are to solving constrained optimization problems [13], TSP problems [14, 15], task scheduling [16], and other numerical optimization problems [17, 18]. Because the cuckoo search algorithm is not dependent on the initial guess and derivability of the objective function, and the solving of the nonlinear equations can be transferred into an optimal problem, thus, the cuckoo search algorithm can be used to solve the systems of nonlinear equations.

2. Cuckoo Search Algorithm

The cuckoo search is a metaheuristic algorithm, which imitates the cuckoos' manner of looking for suitable parasitic nests for egg hatching. The basic principle is as follows. (1) The parasitic nests of cuckoo parasitic eggs correspond to a solution in the search space. (2) Each parasitic nest location corresponds to a fitness value of the algorithm. (3) The walk process of cuckoos' manner of looking for parasitic nests maps to the intelligent optimization process of the algorithm.

The new suitable parasitic nest is generated according to the following law

$$nest_i^{t+1} = nest_i^{t} + \alpha \oplus Levy\ (\lambda) \qquad (2)$$

where $nest_i^t$ is the location of the *t* generation of the *i*-th parasitic nest, and α is the step size value depending on the optimization problem. In most cases, α can be set to be the value of 1. The product $\oplus$ means entry-wise multiplication. The random step size is multiplied by the random numbers with *Lévy* distribution, which according to the following probability distribution

$$Levy \sim u = t^{-\lambda} \qquad (3)$$

where *t* is step size drawn from a Levy distribution. Because the integral of the Levy distribution is difficult, the equivalent calculation can be realized by Mantegana algorithm [12], which is given by

$$nest_i^{t+1} = nest_i^{t} + stepsize \oplus randn\ () \quad \mathrm{i} = 1,2,..., \ \mathrm{N} \qquad (4)$$

where *randn*() is the random function which satisfies Gauss distribution, $stepsize = \alpha \cdot step \oplus (\mathrm{nest}_\mathrm{i}^\mathrm{t} - \mathrm{nest}_\mathrm{best}^\mathrm{t})$, $\alpha = 0.01$, $\mathrm{nest}_\mathrm{i}^\mathrm{t}$ is *i*-th parasitic nest of the *t*-th generation, $\mathrm{nest}_\mathrm{best}^\mathrm{t}$ is the optimal parasitic nest of the *t*-th generation, and $step$ is calculated by

$$step = \frac{u}{|\,\mathrm{v}\,|^{1/\beta}}, u \sim N(0, \sigma_u^2),\ v \sim N(0, \sigma_v^2) \qquad (5)$$

where $\sigma_v = 1, \beta = 3/2$, and σ_u can be written as

$$\sigma_u = \left\{ \frac{\Gamma(1+\beta) \cdot \sin(\pi\beta/2)}{\Gamma[(1+\beta)/2] \cdot \beta \cdot 2^{(\beta-1)/2}} \right\}^{1/\beta} \qquad (6)$$

After the location update, the egg laid by a cuckoo can be spotted by the host of the parasitic nest with a probability $p_a = [0,1]$. For such incidents, the host of the parasitic nest abandons the nest and seeks for a new site to rebuild the nest.

Based on these rules which described above, the steps involved in the computation of the standard cuckoo search algorithm are presented in Algorithm 1.

Algorithm 1: The standard cuckoo search algorithm

```
1. Objective function $f(nest), nest = (nest_1, nest_{2,...,} nest_n)$
2. Generate initial population of n host of the parasitic nests $nest_i (i = 1,2,..., n)$
3. Generation iter=1, define probability $p_a$, set walk step length $\alpha$
4. Evaluate the fitness function $F_i = f(nest_i)$
5. while (iter < MaxGeneration) or (stop criterion)
6. Get a cuckoo egg $nest_j$ from random host of the parasitic nest by Levy flight
7. Evaluate the fitness function $F_j = f(nest_j)$
8. Choose a parasitic nest i among n host parasitic nests, and Evaluate the fitness function $F_i = f(nest_i)$
9. If( $F_j > F_i$ ) then
10. Replace $nest_i$ with $nest_j$
11. End if
12. A fraction ( $p_a$ ) of the worse parasitic nests are evicted
13. Built new parasitic nests randomly to replace lost nests
14. Evaluate the new parasitic nests' fitness
15. Keep the best solutions
16. Rank the solutions
17. Update the generation number iter= iter+1
18. End while
19. Output the best solutions
```

3. Double Mutation Cuckoo Search Algorithm

The basic cuckoo algorithm uses Levy flight to update the location of parasitic nests. The update mode of Levy flight is essentially based on Markov chain methods. The destination of parasitic nest location update is determined by the current parasitic nest location and transition probabilities. The search process leads to slow convergence speed and low accuracy. In this paper, the cuckoo algorithm is modified using double mutation operators. The mutation operations with large probability are carried out for the parasitic nests in poor position and the mutation operations with small probability are carried out for the ones in good position. A school of population is initialized by chaotic search technology optimization algorithm, making the algorithm high-quality and uniformly-distributed. To overcome the disadvantage that the cuckoo algorithm is easy to fall into local boundary optimal value, the parasitic nests are generated randomly when exceeding the borders.

3.1. Double Mutation Operator

In order to balance the performance of exploration and exploitation in cuckoo search algorithm, double mutation operators are used for the basic cuckoo search algorithm, which can jump out of the local optimal value with global search ability. Meanwhile, the proposed algorithm can get the high accuracy in the global scope of the optimal solution.

3.1.1. Global Search Mutation Operator: Global search mutation operator mutates parasitic nests with the worst fitness value. The proposed algorithm mutates by dimension according to Eq. (7) to enhance the diversity of the population and the global convergence of the algorithm. In Eq. (7), the algorithm mutates the parasitic nests in poor position with large probability p_1 after each iteration:

$$nest'_{worst,j} = nest_{worst,j} \otimes (1 + \beta \cdot C(0,1)) \tag{7}$$

where $nest_{worst,j}$ is the *j*-th dimension value of the worst parasitic nest, $nest'_{worst,j}$ is the *j*-th dimension value after variation, β is the coefficient of the mutation step (*e.g.,* 0.618 in this paper), and $C(0,1)$ is a random variable which obeys Cauchy distribution.

3.1.2. Local Search Mutation Operator: Local search mutation operator mutates parasitic nests with optimal fitness values. The mutation opportunity is carried out by evaluating the change rate of the fitness values of the optimal individuals, which is given by

$$\frac{|\, fit(\text{nest}^{t}_{\text{best}}) - fit(\text{nest}^{t-10}_{\text{best}})\,|}{fit(\text{nest}^{t}_{\text{best}})} \le \Delta, t > 10 \tag{8}$$

where $fit(\text{nest}^{t}_{\text{best}})$ is the optimal fitness value of the *t*-th iteration, and Δ is a threshold value.

If the change rate is less than a certain threshold value (*e.g.,* 0.005 in this paper), the algorithm mutates the parasitic nests with small probability p_2 to improve the local search ability.

$$nest'_{best,j} = nest_{best,j} \otimes (1 + \gamma \cdot randn()) \tag{9}$$

where $nest_{best,j}$ is the *j*-th dimension value of the best parasitic nest, $nest'_{best,j}$ is the *j*-th dimension value after variation, γ is the coefficient of the mutation step (e.g., 0.5 in this paper), and $randn()$ is a random variable which obeys Gaussian distribution.

3.2. Initial Population Generated by a Chaotic Array

In evolutionary algorithms based on population iterations, the diversity of the initial population can produce active effect on the search performance of the algorithms [19]. Due to the uncertainty of the area of the optimal values, the optimization problems need to be solved by multiple searches or to increase the size of the population. The calculated amount of the algorithms is increased, and the stability is reduced. Chaos phenomenon is an inherent characteristic in deterministic nonlinear dynamic systems, which is random and ergodic. It satisfies the diversity of the initial population of the swarm intelligent algorithms. In this paper, Eq. (10) is chosen to initialize the population.

$$x_n = \begin{cases} x_n / a, 0 < x_n \le a \\ (1 - x_n)/(1 - a), a < x_n \le 1 \end{cases} \tag{10}$$

According to Eq. (10), the basic steps of the initializing the cuckoo algorithms is as follows.

Step 1:Generate randomly a *d*-dimensional initial sequence value, $x(0) = (x_1(0), x_2(0),..., x_d(0))$. Each dimension value of $x(0)$ is a random number between 0 and 1.

Step 2:We use Eq. (10) to iterate *T* times. The *i*-th order is $x(i) = (x_1(i), x_2(i),..., x_d(i))$.

Step 3:According to Eq. (11), we can get an initial individual in solution space.

$$nest_j(i) = \frac{l_j + u_j}{2} + \frac{u_j - l_j}{2} \cdot x_j(i) \tag{11}$$

where $nest_j(i)$ is *j*-th dimensional value of *i*-th initial, $x_j(i)$ is *j*-th dimensional value of *i*-th chaotic sequence value, l_j and u_j are the corresponding lower and upper boundary values of the *j*-dimensional solution space.

3.3. Handling Strategy of Boundary Values

In basic cuckoo algorithms, the positions of the parasitic nests are set at the boundary of the population when the Levy flight exceeds the border. The algorithms are apt to fall into local optimal values of the boundary, which results in a search stagnation and multi-directional boundary gathering. After iterations, it is inevitable for the parasitic nests to form similar behaviours, and the diversity of the population is dropped. In the proposed algorithm, Eq. (12) is adopted to handle the boundary values, which is given by

$$nest_j(i) = l_j + rand() \cdot (u_j - l_j) \qquad (12)$$

where l_j and u_j are the corresponding lower bound and upper bound values of the *j*-dimensional solution space, respectively. The handling strategy of boundary values ensures the search scope of the algorithm, and overcomes the disadvantage that the basic cuckoo algorithms are easy to fall into local optimal values on the boundary.

3.4. The Procedure of the Proposed Algorithm

The procedure of the proposed algorithm for solving the systems of nonlinear equation is summarized in Algorithm 2.

```
Algorithm 2: dynamic double mutation cuckoo algorithm
1. Objective function $f(nest)$, $nest = (nest_1, nest_{2,...,} nest_n)$
2. Generate initial population of n parasitic nests $nest_i (i = 1,2,..., n)$ according to 3.2.
3. Generation iter=1, initialization parameters of $p_a, \alpha, p_{m1}, p_{m2}, l_j, u_j$.
4. for all $nest_i$ do
5. Calculate the fitness function $F_i = f(nest_i)$ according to Eq.(1).
6. end for
7. while (iter < MaxGeneration) or (stop criterion)
8.     Get a cuckoo egg $nest_j$ from random host of the parasitic nest by Levy flight.
9.     Evaluate the fitness function $F_j = f(nest_j)$ ,select the parasitic nest of the worst
         fitness values $nest_{worst}$ and $nest_{best}$ , respectively.
10.    Update the $nest_{worst}$ according to 3.1.1, and evaluate the fitness
       function $f(nest_{worst})$ .
         Update the $nest_{best}$ according to 3.1.2,and evaluate the fitness
function $f(nest_{best})$ .
11.    Choose a parasitic nest i among n host parasitic nests, $F_i = f(nest_i)$
12.    If( $F_j > F_i$ ) then
           Replace $nest_i$ with $nest_j$
13.    End if
14.    A fraction ( $p_a$ ) of the worse parasitic nests are evicted
15.    Built new parasitic nests randomly to replace lost nests
16.    Evaluate the new parasitic nests' fitness
17.    Keep the best solutions
18.    If $nest_{i,j} > u_j$ or $nest_{i,j} < l_j$
19.        Generate new parasitic nests which are out of bounds according to
             Eq.(12),
        and evaluate the fitness values accordingly.
20.     End if
21.     Rank the solutions
22.     Update the generation number iter= iter+1
23.End while
24.Output the best solutions
```

4. Experiment and Results

4.1. Algorithm Simulation and Analysis

To test the performances of the proposed algorithm, four standard functions (Sphere, Rastrigrin, Griewangk, Six-Hump Camelback) are selected. To further study the performances of ICS algorithm, comparisons are carried out with several typical methods from the literatures, including the standard cuckoo search (CS) algorithm, opposition-based differential evolution (OBDE) [20] method, comprehensive learning particle swarm optimizer (CLPSO) [21], artificial bee colony(ABC) [23], and particle swarm optimization with an aging leader and challengers (ALC-PSO) [22] algorithm. The search results are from the four kinds of the algorithms in the corresponding literatures. Their statistical results are shown in Table 1 and Table 2, respectively. In this paper, the scale of the population $size = 100$, $p_a = 0.25$, $p_{m1} = 0.2$, $p_{m2} = 0.03$, $\lambda = 2$, the iteration number of the Six-Hump Camelback function *t*=500. The iteration number *t* of the other 3 functions is 3000. For each test function, 20 independent runs are performed in Matlab R2009b.

Test 1: Six-Hump Camelback function

$$F_1 \ \min \ f(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$$

The Six-Hump Camelback function has six local optimal values and two variables. The global solutions are located at either $x = (-0.08984, 0.71266)$ or $x = (0.08984, -0.71266)$, and each solution has a corresponding function value -1.0316285. The bound variables are set between -100 and 100.

Test 2: Sphere function

$$F_2 \ \min \ f(x) = \sum_{i=1}^{n} x_i^2$$

The Sphere function has a global optimal value. The minimum solution is located at $x = (0,0,\dots,0)$, and the corresponding function value is 0. Set n=30, and the bound variables are set to be -10 and 10.

Test 3: Rastrigrin fuction

$$F_3 \ \min \ f(x) = \sum_{i=1}^{D} (x_i^2 - 10\cos(2\pi x_i) + 10)$$

The solution is $f(0,0,\dots,0) = 0$. Set D=30, and the bound variables are set to be -5.2 and 5.2.

Test 4: Griewangk function

$$F_4 \ \min \ f(x) = \frac{1}{4000}\sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$$

The Griewangk function is a multimodal function, and it is difficult to find the global optimal value. When $x_i = 0 (i = 1,2,\dots,D)$, the function can reach the global minimum value 0. Set D=30, and the bound variables are set to be -100 and 100.

On these functions, we focus on comparing the performance of the algorithms in terms of the solution accuracy and convergence speed. For F_1, it is a 2-dimensional function. As can be seen from Table 1, the DMCS and CS algorithms for F_1 have the same precisions, and they both attain theoretical values. The mean number of the iterations of the DMCS algorithm is 345.5, which is less than that of the CS algorithm with 451.25 iterations. With regard to F_2 - F_3 , the optimal values, mean values, and the deviations of the DMCS algorithm are much better than those of the other methods. For F_4 , the mean values and the deviations of the DMCS algorithm are superior to those of the other algorithms. The reliability of search is reflected by the "suc%" in Table 2, which stands for the percentage of the successful runs that acceptable solutions are found. It is found from Table 2 that, the "suc%", the minimum number of iterations, and the mean number of the iterations of the DMCS algorithm are better than those of the CS algorithm. The convergence history of the DMCS algorithm and CS algorithm are shown in Figure 1-Figure 4.

Table 1. Experimental Comparison between OBDE, CLPSO, ALC-PSO, ABC, CS, ICS and DMCS

function	Algorithm	Best	Mean	Dev
F_1	CS	-1.03162845348988	-1.03162845348988	0
	DMCS	-1.03162845348988	-1.03162845348988	0
F_2	OBDE[20]	3.05991167e-86	1.42671462e-6	7.810816693e-6
	CLPSO[21]	2.567e−29	1.390e−27	2.052e−27
	ALC-PSO [22]	1.135e−172	1.677e−161	8.206e−161
	ABC[23]	N/A	4.69e-16	1.07e-16
	CS	6.71006692182639e-15	1.26130238689835e-14	3.697855e-15
	ICS [24]	2.9807e-22	9.5438e-21	1.1279e-20
	DMCS	0	0	0
F_3	OBDE[20]	7.91516238	1.4083459e+1	4.46978243
	CLPSO[21]	0	2.440e−14	5.979e−14
	ALC-PSO [22]	7.105e−15	2.528e−14	1.376e−14
	ABC[23]	N/A	4.80e-5	0.000243
	CS	7.6481865285416e-10	5.59051123172338e-08	1.25653487e-9
	ICS [24]	1.1939e+01	2.2296e+01	4.1242e+00
	DMCS	2.6683564222 e-25	8.5590435844 e-22	5.2290854372 e-28
F_4	OBDE[20]	5.55111512e-16	5.16170306e-3	1.637476538e-2
	CLPSO[21]	0	2.007e−14	8.669e−14
	ALC-PSO [22]	0	1.221e−2	1.577e−2
	ABC[23]	N/A	5.82e-6	3.13e-5
	CS	5.99674524890748	6.27549115481021e+1	20.45980263522
	ICS [24]	1.1102e-16	3.1173e-09	1.1340e-08
	DMCS	5.8933557546 e-32	7.893900369 e-28	3.768812901 e-31

Table 2. Search Speed and Reliability Comparisons between CS and DMCS

Function	Number of Generation	CS				DMCS			
		suc%	best iterations	mean iterations	Running time(S)	suc%	best iterations	mean iterations	Running time(S)
F_1	500	100	402	451.25	3.160804	100	335	345.5	4.866809
F_2	3000	100	2576	2683.75	18.022565	100	489	496.25	20.95774
F_3	3000	80	2897	2962.5	16.835218	100	478	490.75	18.82253
F_4	3000	60	2606	2776.25	17.751657	100	443	457.05	20.01353

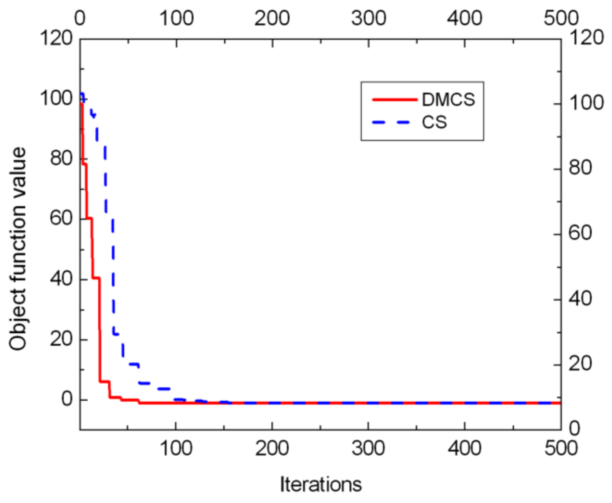

Figure 1. F_1 Curves of the Objective Value

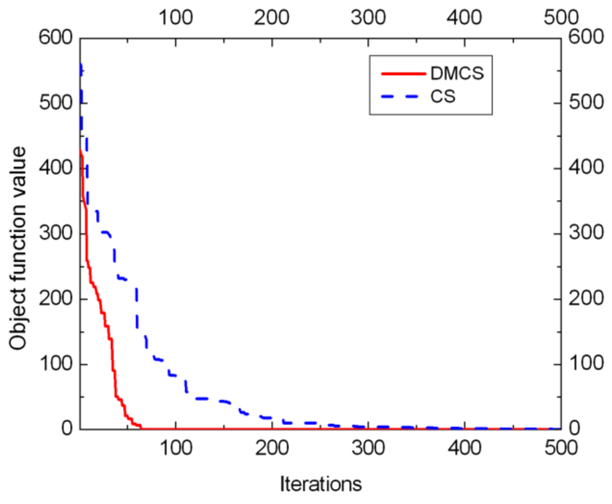

Figure 2. F_2 Curves of the Objective Value

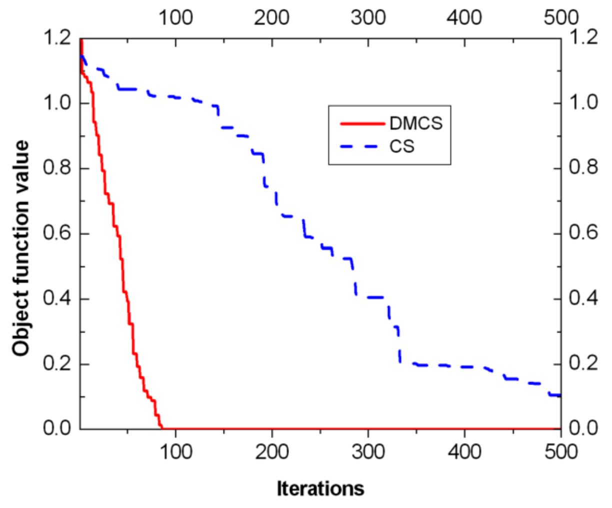

Figure 3. F_3 Curves of the Objective Value

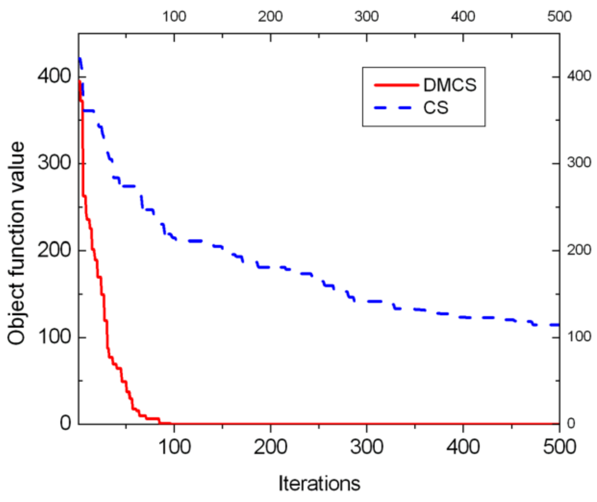

Figure 4. F_4 Curves of the Objective Value

4.2. The Solution of the Nonlinear Equations

Some standard systems are introduced to demonstrate the efficiency of the proposed algorithm for solving systems of nonlinear equations. The scale of the population $size = 100$, $p_a = 0.25$, $p_{m1} = 0.2$, $p_{m2} = 0.03$, $\lambda = 2$. The iteration number t of the other 3 functions is 1000.

Case 1:

$$f_1(x) = 0.5\sin(x_1 x_2) - 0.25 x_2 / \pi - 0.5 x_1 = 0$$

$$f_2(x) = (1 - \frac{1}{4\pi})(e^{2x_1} - e) + e \cdot x_2 / \pi - 2e \cdot x_1 = 0$$

$$0.25 \le x_1 \le 1, 1.5 \le x_2 \le 2\pi$$

This problem has already been solved by many researchers. Wang, *et al.*, [25] presented a new filled function method. Abdollahi, *et al.*, [8] proposed an imperialist competitive algorithm. The best solutions obtained by the mentioned approaches are shown in Table 3. From Table 3, the optimal value of DMCS algorithm for Case 1 is superior to that of other two methods. In Figure 5, it is shown that DMCS algorithm can quickly converge to the optimal value by the convergence curves.

Table 3. Comparing Results of DMCS and CS for Case 1 with those by Wang, *et al.*, [25] and Abdollahi, *et al.*, [8]

Algorithm	Solutions		Functions values	
Wang et al.[25]	x_1	0.50043285	$f_1(x)$	-2.3851e-004
	x_2	3.14186317	$f_2(x)$	4.7741e-005
Abdollahi et al.[8]	x_1	0.299448692495720	$f_1(x)$	2.3267e-012
	x_2	2.836927770471037	$f_2(x)$	4.6696e-013
CS	x_1	0.500000000000745	$f_1(x)$	-3.86524146023248E-13
	x_2	3.14159265358997	$f_2(x)$	-1.70086167372574E-13
DMCS	x_1	0.500000000000006	$f_1(x)$	-1.16573417585641E-15
	x_2	3.14159265358977	$f_2(x)$	-2.26485497023531E-14

Case 2:

$$f_1(x) = x_1^{x_2} + x_2^{x_1} - 5x_1x_2x_3 - 85 = 0$$

$$f_2(x) = x_1^3 - x_2^{x_3} - x_3^{x_2} - 60 = 0$$

$$f_3(x) = x_1^3 + x_3^{x_1} - x_2 - 2 = 0$$

$$3 \le x_1 \le 5, 2 \le x_2 \le 4, 0.5 \le x_3 \le 2$$

This problem involves three variables and three nonlinear equations. Mo, *et al.,* [6] and Zhang, *et al.,* [26] solved the system. The best solutions obtained by the DMCS and other algorithms are listed in Table 4.

Table 4. Comparsion Results of Case 2 by DMCS, CS, Zhang, *et al.,* [26] and Mo, *et al.,* [6]

Algorithm	Solutions		Functions values	
Mo et al.[6]	x_1	4	$f_1(x)$	0
	x_2	3	$f_2(x)$	0
	x_3	1	$f_3(x)$	0
Zhang et al.[26]	x_1	4.00118	$f_1(x)$	-0.0640
	x_2	3.00013	$f_2(x)$	0.0322
	x_3	1.00385	$f_3(x)$	0.0380
CS	x_1	4.00000000000101	$f_1(x)$	-1.49640300151077E-11
	x_2	3.00000000000005	$f_2(x)$	3.28341798194742E-11
	x_3	1.00000000000247	$f_3(x)$	2.45665709996956E-11
DMCS	x_1	4	$f_1(x)$	0
	x_2	3	$f_2(x)$	0
	x_3	1	$f_3(x)$	0

It is shown that the best solutions obtained by DMCS can achieve exact solution, and the solution accuracy is far better than that by CS and Zhang, *et al.,* [26]. So DMCS is efficient for solving the systems of nonlinear equations.

Case 3:

$$f_1(x) = x_1^2 + x_3^2 - 1 = 0$$

$$f_2(x) = x_2^2 + x_4^2 - 1 = 0$$

$$f_3(x) = x_5x_3^3 + x_6x_4^3 = 0$$

$$f_4(x) = x_5x_1^3 + x_6x_2^3 = 0$$

$$f_5(x) = x_5x_1x_3^2 + x_6x_4^2x_2 = 0$$

$$f_6(x) = x_5x_1^2x_3 + x_6x_2^2x_4 = 0$$

$$-10 \le x_i \le 10, 1 \le i \le 6$$

The system of nonlinear equations of Case 3 involves six variables and six nonlinear equations. Table 5 listed the results by Grosan, *et al.,* [9] and the best results of DMCS and CS. From Table 5, it is observed that DMCS algorithm can obtain the approximate roots of the system of nonlinear equations, and the calculation accuracy achieves 10e-05. Compared with the results of Grosan, *et al.,* [9], the optimal solution by DMCS algorithm has small errors.

Table 5. Comparison Results of Case 3

The best results of [9]		The best results of CS		The best results of DMCS	
Variables values	Functions values	Variables values	Functions values	Variables values	Functions values
-0.8078668904	-0.0050	0.997077606024159	6.253476E-05	1	6.060615E-06
-0.9560562726	-0.0367	0.997035297339922	2.544754E-05	1	5.971777E-06
0.5850998782	0.0125	0.076803530685376	1.397927E-06	0.002461831574518	-3.385235E-11
-0.2219439027	-0.0276	0.077110721652628	-1.314747E-05	0.002443721874666	2.611694E-05
0.0620152964	-0.0169	-0.254520085156639	1.204824E-05	-0.104737114527382	-9.148682E-09
-0.0057942792	0.0249	0.25453922272324	7.754110E-05	0.104763231468994	-1.832935E-06

Case 4:

$$f_1(x) = x_1 + \frac{x_2^4 x_4 x_6}{4} + 0.75 = 0$$

$$f_2(x) = x_2 + 0.405\, e^{1+x_1x_2} - 1.405 = 0$$

$$f_3(x) = x_3 - \frac{x_4 x_6}{2} + 1.5 = 0$$

$$f_4(x) = x_4 - 0.605\, e^{(1-x_3^2)} - 0.395 = 0$$

$$f_5(x) = x_5 - \frac{x_2 x_6}{2} + 1.5 = 0$$

$$f_6(x) = x_6 - x_1 x_5 = 0$$

This system has six variables and six nonlinear equations. Mo, *et al.,* [6] presented a Conjugate direction particle swarm optimization for solved the system, and Jaberipour, *et al.,* [5] solved it using a particle swarm optimization. Table 6 lists the best solution by DMCS algorithm and other methods. From Table 6, it is observed that the DMCS results are very close to the theoretical solution which obtained by Mo, *et al.,* [6] and the solution accuracy is better than the solution of the CS algorithm.

Table 6. Comparison Results of Case 4

Algorithm	x	Variables values	$f(x)$	Functions values
Mo et al.[6]	x_1	-1	$f_1(x)$	0
	x_2	1	$f_2(x)$	0
	x_3	-1	$f_3(x)$	0
	x_4	1	$f_4(x)$	0
	x_5	-1	$f_5(x)$	0
	x_6	1	$f_6(x)$	0
CS	x_1	-0.999639167	$f_1(x)$	1.588257E-04
	x_2	0.999501109	$f_2(x)$	-1.506263E-04
	x_3	-0.999747837	$f_3(x)$	1.573150E-04
	x_4	1.000276522	$f_4(x)$	-2.863310E-05
	x_5	-1.00028690	$f_5(x)$	5.922799E-06
	x_6	0.999913197	$f_6(x)$	-1.276931E-05
DMCS	x_1	-0.999999985	$f_1(x)$	2.958164E-09
	x_2	0.999999991	$f_2(x)$	8.783643E-10
	x_3	-1.000000018	$f_3(x)$	-2.005251E-09
	x_4	0.999999977	$f_4(x)$	-1.096312E-09
	x_5	-1.000000006	$f_5(x)$	2.527675E-09
	x_6	0.999999991	$f_6(x)$	-7.837286E-11

Case 5:

$$f_1(x) = (3-5x_1)x_1 + 1 - 2x_2 = 0$$

$$f_i(x) = (3-5x_i)x_i + 1 - x_{i-1} - 2x_{2+1} = 0, i = 2,3,\ldots,9$$

$$f_{10}(x) = (3-5x_{10})x_{10} + 1 - 2x_9 = 0$$

Table 7. Comparison Results of DMCS with CS and Mo, *et al.,* [6]

Mo et al.[6]		CS		DMCS	
x	$f(x)$	x	$f(x)$	x	$f(x)$
0.915551	-3.1680e-006	-0.382085114	-1.813716E-07	-0.382085089	1.554312E-15
-0.222256	3.5232e-007	-0.438100165	9.731738E-08	-0.43810017	-2.664535E-15
-0.414654	-1.6986-006	-0.445937127	-1.334805E-07	-0.445937108	1.221245E-15
-0.439254	1.7710e-006	-0.447005346	-4.114950E-08	-0.447005339	2.997602E-15
0.420892	-1.6836	-0.447073882	9.840950E-09	-0.447073887	-1.998401E-15
-0.354588	2.5254	-0.446795796	6.933357E-08	-0.44679581	-1.887379E-15
-0.135767	-0.8418	-0.445722995	1.289920E-07	-0.445723013	3.774758E-15
0.427562	-3.9144e-007	-0.441859131	3.325555E-09	-0.441859125	-2.220446E-16
0.752203	6.8078e-007	-0.428025929	-2.294208E-07	-0.428025896	-1.110223E-15
-0.440697	2.3396e-007	-0.379124703	4.536877E-08	-0.3791247	-1.110223E-15

In Table 7, it is observed that we can obtain the approximate roots of Case 5, and the calculation accuracy achieves 10e-15. Compared with the solutions of the Ref. [6], the DMCS results are the exact solution and outperform those by Mo, *et al.*, [6]. From Figure 5 to Figure 9, in terms of convergence speed, DMCS consumes smaller numbers of iteration than those of CS to reach acceptable results. Overall, DMCS is an efficient method for solving the systems of nonlinear equations in the comparison.

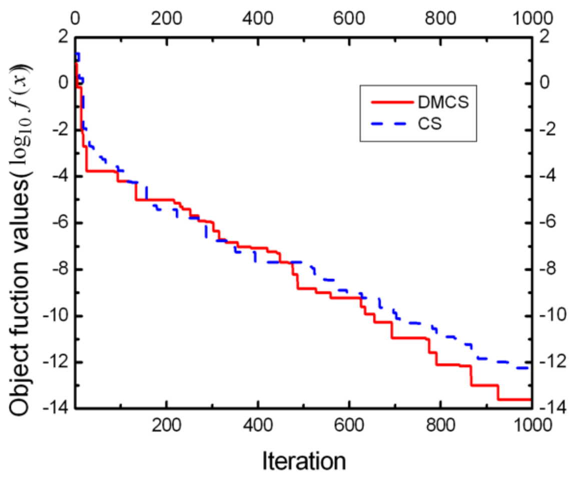

Figure 5. Case 1 Curves of the Objective Value

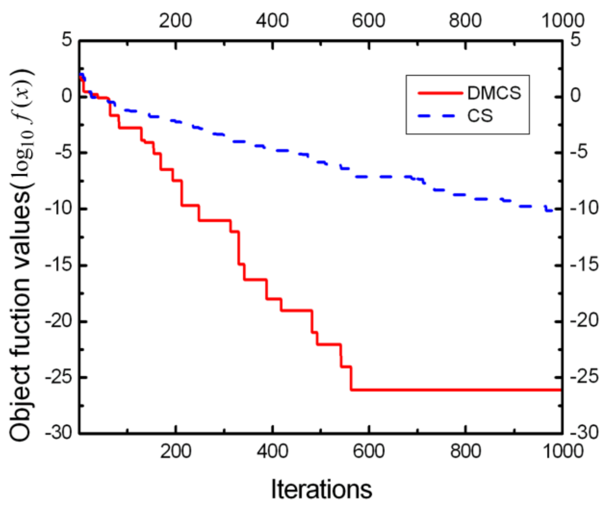

Figure 6. Case 2 Curves of the Objective Value

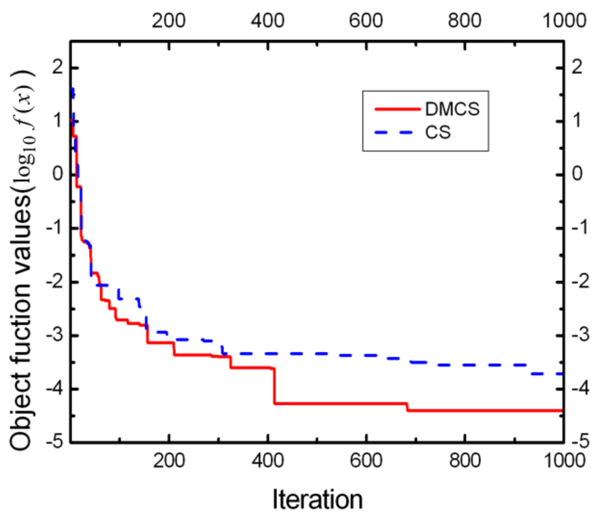

Figure 7. Case 3 Curves of the Objective Value

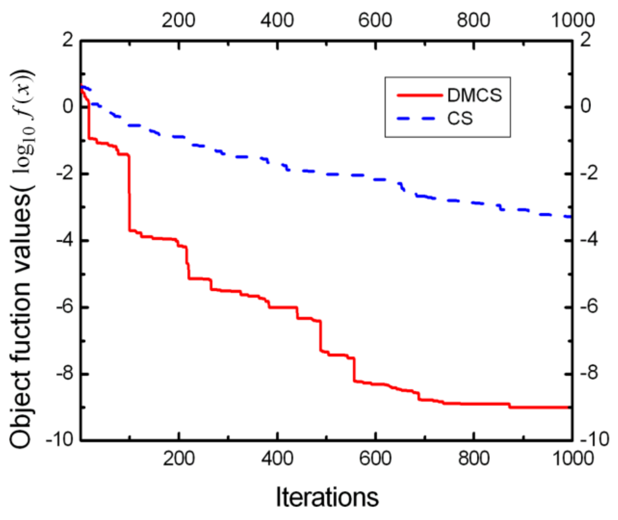

Figure 8. Case 4 Curves of the objective value

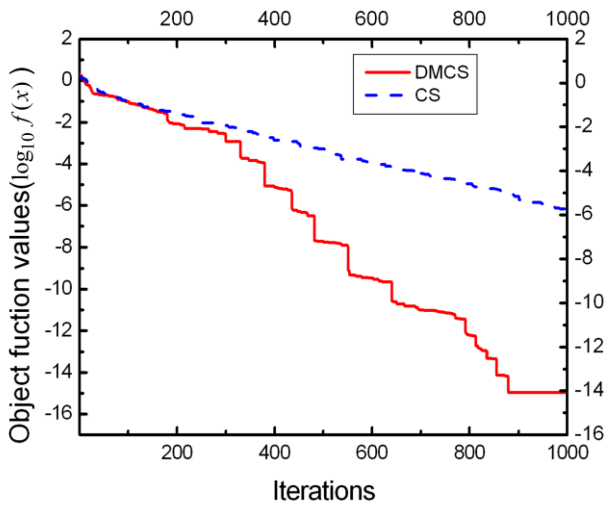

Figure 9. Case 5 Curves of the Objective Value

Conclusion

This paper proposes a double mutation cuckoo search algorithm for systems of nonlinear equations. The systems of nonlinear equations can be converted to minimization problems. To avoid the sensitivity to initial values, the algorithm adopts double mutation method to enhance the abilities of local search and global optimization. Meanwhile, the boundary value handling strategy adds diversity to the population. The proposed algorithm does not rely on the selection of the initial values, and the equations do not need to be continuous and differentiable. Some standard problems were solved by DMCS, and the proposed method is effective and has high accuracy for solving systems of nonlinear equations in comparison with other methods.

Acknowledgments

This work is financially supported by the Natural Science Foundation of Guangxi Province (Grant No. 2014GXNSFBA118283) and the Higher School Scientific Research Project of Guangxi Province (Grant No. 2013YB247).

References

[1] Pielorz and Amalia, “Nonlinear equations with a retarded argument in discrete-continuous systems”, Mathematical Problems in Engineering, vol. 2007, article no. 28430, **(2007)**.
[2] L. Xiao and R. Lu, “Finite-time solution to nonlinear equation using recurrent neural dynamics with a specially-constructed activation function”, Neurocomputing, vol. 151, no. P1, **(2015)**, pp. 246-251.
[3] Bader and W. Brett, “Tensor-Krylov methods for solving large-scale systems of nonlinear equations”, SIAM journal on numerical analysis, vol. 43, no. 3, **(2005)**, pp. 1321-1347.
[4] Y.-Z. Luo, G.-J. Tang and L.-N. Zhou, “Hybrid approach for solving systems of nonlinear equations using chaos optimization and quasi-Newton method”, Applied Soft Computing, vol. 8, no. 2, **(2008)**, pp. 1068-1073.
[5] M. Jaberipour, E. Khorram and B. Karimi, “Particle swarm algorithm for solving systems of nonlinear equations”, Computers & Mathematics with Applications, vol. 62, no. 2, **(2011)**, pp. 566-576.
[6] Y. Mo, H. Liu and Q. Wang, “Conjugate direction particle swarm optimization solving systems of nonlinear equations”, Computers & Mathematics with Applications, vol. 57, no. 11, **(2009)**, pp. 1877-1882.
[7] Y. Zhou, J. Liu and G. Zhao, “Leader glowworm swarm optimization algorithm for solving nonlinear equations systems”, Przeglad Elektrotchniczny, vol. 88, no. 1b, **(2012)**, pp. 101-106.
[8] M. Abdollahi, A. Isazadeh and D. Abdollahi, “Imperialist competitive algorithm for solving systems of nonlinear equations”, Computers & Mathematics with Applications, vol. 65, no. 12, **(2013)**, pp. 1894-1908.

[9] C. Grosan and A. Abraham, "A new approach for solving nonlinear equations systems", IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans, vol. 38, no. 3, **(2008)**, pp. 698-714.
[10] X.-S. Yang and S. Deb, "Cuckoo search via Lévy flights", IEEE World Congress on Nature & Biologically Inspired Computing (NaBIC), **(2009)**, pp. 210-214.
[11] X.-S. Yang and S. Deb, "Engineering optimisation by cuckoo search", International Journal of Mathematical Modelling and Numerical Optimisation, vol. 1, no. 4, **(2010)**, pp. 330-343.
[12] X. S. Yang, "Nature-inspired metaheuristic algorithms", Luniver press, **(2010)**.
[13] A. H. Gandomi, X.-S. Yang and A. H. Alavi, "Cuckoo search algorithm: a metaheuristic approach to solve structural optimization problemsc", Engineering with computers, vol. 29, no. 1, **(2013)**, pp. 17-35.
[14] X. Ouyang, Y. Zhou, Q. Luo and H. Chen, "A novel discrete cuckoo search algorithm for spherical traveling salesman problem", Appl. Math, vol. 7, no. 2, **(2013)**, pp. 777-784.
[15] X. Ding, Z. Xu, N. J. Cheung and X. Liu, "Parameter estimation of Takagi-Sugeno fuzzy system using heterogeneous cuckoo search algorithm", Neurocomputing , vol. 151, no. P3, **(2015)**, pp. 1332-1342.
[16] S. Burnwal and S. Deb, "Scheduling optimization of flexible manufacturing system using cuckoo search-based approach", The International Journal of Advanced Manufacturing Technology, vol. 64, **(2013)**, pp. 951-959.
[17] X.-S. Yang and S. Deb, "Multiobjective cuckoo search for design optimization", Computers & Operations Research, vol. 40, no. 6, **(2013)**, pp. 1616-1624.
[18] M. Tuba, M. Subotic and N. Stanarevic, "Modified cuckoo search algorithm for unconstrained optimization problems", Proceedings of the 5th European Conference on European Computing Conference, World Scientific and Engineering Academy and Society (WSEAS), **(2011)**, pp. 263-268.
[19] R. L. Haupt and S. E. Haupt, "Practical genetic algorithms", John Wiley & Sons, **(2004)**.
[20] S. Rahnamayan, H. R. Tizhoosh and M. M. A. Salama, "Opposition-based differential evolution", IEEE Transactions on Evolutionary Computation, vol. 12, no. 1, **(2008)**, pp. 64-79.
[21] J. J. Liang, A. K. Qin, P. N. Suganthan and S. Baskar, "Comprehensive learning particle swarm optimizer for global optimization of multimodal functions", IEEE Transactions on Evolutionary Computation, vol. 10, no. 3, **(2006)**, pp. 281-295.
[22] W. N. Chen, J. Zhang, Y. Lin, N. Chen, Z.-H. Zhan, H.-S.-H. Chung, Y. Li and Y.-H. Shi, "Particle swarm optimization with an aging leader and challengers", IEEE Transactions on Evolutionary Computation, vol. 17, no. 2, **(2013)**, pp. 241-258.
[23] D. Karaboga and B. Akay, "Artificial bee colony (ABC), harmony search and bees algorithms on numerical optimization", Proceeding of IPROMS-2009 on Innovative Production Machines and Systems, **(2009)**.
[24] E. Valian, S. Mohanna and S. Tavakoli, "Improved cuckoo search algorithm for global optimization", International Journal of Communications and Information Technology, vol. 1, no. 1, **(2011)**, pp. 31-44.
[25] C. Wang, R. Luo, K. Wu and B. Han, "A new filled function method for an unconstrained nonlinear equation ", Journal of Computational and Applied Mathematics, vol. 235, no. 6, **(2011)**, pp. 1689-1699.
[26] X. Zhang and B. Zhou, "Artificial bee colony algorithm for solving systems of nonlinear equations", International Journal of Advancements in Computing Technology, vol. 5, no. 9, **(2013)**, article no. 757391.