Neural Comput & Applic (2017) 28:1667–1682
DOI 10.1007/s00521-015-2138-y

ORIGINAL ARTICLE

Improved imperialist competitive algorithm with mutation operator for continuous optimization problems

Shuhui Xu[1,2] · Yong Wang[1,2] · Peichuan Lu[1,2]

Received: 24 October 2014 / Accepted: 21 December 2015 / Published online: 2 January 2016

Abstract Imperialist competitive algorithm is a nascent meta-heuristic algorithm which has good performance. However, it also often suffers premature convergence and falls into local optimal area when employed to solve complex problems. To enhance its performance further, an improved approach which uses mutation operators to change the behavior of the imperialists is proposed in this article. This improved approach is simple in structure and is very easy to be carried out. Three different mutation operators, the Gaussian mutation, the Cauchy mutation and the Lévy mutation, are investigated particularly by experiments. The experimental results suggest that all the three improved algorithms have faster convergence rate, better global search ability and better stability than the original algorithm. Furthermore, the three improved algorithms are also compared with other two excellent algorithms on some benchmark functions and compared with other four existing algorithms on one real-world optimization problem. The comparisons suggest that the proposed algorithms have their own specialties and good applicability. They can obtain better results on some functions than those contrastive approaches.

Keywords Optimization method · Imperialist competitive algorithm · Mutation operators · Numerical experiments

✉ Yong Wang
meywang@126.com; meywang@sdu.edu.cn

[1] School of Mechanical Engineering, Shandong University, Jinan 250061, China

[2] Key Laboratory of High-efficiency and Clean Mechanical Manufacture (Shandong University), Ministry of Education, Jinan, China

1 Introduction

Optimization is the process which aims to find a more appropriate set of parameters for a given problem, in order to obtain a more desirable outcome. It is necessary for many problems which can be found in diverse fields, such as scientific, social, economic and engineering [1]. Among all kinds of optimization problems, continuous optimization problems take up a large proportion [2, 3]. In recent decades, using meta-heuristic algorithms to deal with such problems has attracted more and more attention and has become an important branch of optimization methodology.

Imperialist competitive algorithm (ICA) is a new population-based meta-heuristic algorithm proposed by Atashpaz-Gargari and Lucas in 2007, inspired by the historical phenomenon of imperialism and colonialism [4]. Due to its competitiveness over other meta-heuristics in terms of convergence rate and global search ability, the ICA has received significant interests within the short time period since its advent and has been successfully applied to a wide range of optimization tasks, which come from various fields such as mechanical engineering [5–8], electrical engineering [9–12], industrial engineering [13–18], civil engineering [19, 20], petroleum engineering [21–23] and computer engineering [24–30]. However, similar to other population-based meta-heuristic algorithms, it also often suffers premature convergence and falls into local optimal area, especially when the problem is complicated, high dimensional or multipeak [31–34].

Several improved approaches have been proposed to enhance the algorithm's performance. Bahrami et al. [31] proposed a method which utilizes chaotic maps instead of the original uniform distribution to adjust the angle of colonies' movement toward imperialist, to help the algorithm escaping from local optima; Abdechiri et al. [32]

proposed an adaptive ICA, in which a probability density function is introduced and used to dynamically adapt the angle, to balance the exploration ability and exploitation ability of the algorithm. Arish et al. [35] proposed a fuzzy version, in which, in the absorption policy, colonies are moved toward the resulting vector of all imperialists with the aid of a fuzzy membership function. Kaveh and Talatahari [20, 36] presented an improved approach with two new defined movement steps and used the redefined algorithm to optimize the design of skeletal structures and to solve other engineering optimization problems. As a further work, Talatahari et al. [34] proposed a method which utilizes chaotic variables to replace the random variables in the assimilation equation which guides the colonies' movements. They proposed a total of three replacement approaches and investigated seven different chaotic maps to generate the chaotic variables.

All the improved approaches mentioned above are focused on enhancing the performance of the original algorithm through changing the moving mode of the colonies. Some modifications which are quite complicated are needed, bringing some difficulties to the implementation and application of these improved approaches. In this article, we propose a novel improved method which is simple in structure and is very easy to be carried out. The main idea of the proposed approach is to enhance the algorithm's performance by changing the moving mode of the imperialists through applying mutation operators to them. Three different mutation operators, the Gaussian mutation, the Cauchy mutation and the Lévy mutation, are investigated particularly through numerical experiments.

The rest of this article is structured as follows. In Sect. 2, a brief introduction about the basic ICA is given. In Sect. 3, the proposed improved approach is set out in detail. In Sect. 4, the numerical experiments, results, related discussions and the comparisons with other algorithms are given. And in Sect. 5, a conclusion of this work and the key areas of the future works are provided.

2 Basic imperialist competitive algorithm

The ICA simulates the colonial competition in human society. Its implementation can be divided into the following steps [4].

2.1 Initialize the empires

Similar to other population-based meta-heuristic algorithms, ICA also begins with a randomly generated population which contains N initial solutions. In ICA, each individual is called a 'country.' For an N_{var}-dimensional optimization problem, a country is a $1 \times N_{\text{var}}$ array whose elements are randomly generated in the allowable range of the corresponding parameters, as:

$$\text{country} = [p_1, p_2, p_3, \ldots, p_{N_{\text{var}}}]$$

After, the cost of each country is calculated by the cost function.

$$\text{cost} = f(\text{country}) = f(p_1, p_2, p_3, \ldots, p_{N_{\text{var}}})$$

Then, all the initial countries would be divided into two classes. Some best countries among them would be regarded as 'imperialists,' and the rest countries would be regarded as 'colonies.' After that, the cost of every imperialist is normalized by Eq. (1) [4], where C_n and c_n stand for the normalized cost and the cost of the nth imperialist, respectively. And then, the normalized power of each imperialist is calculated by Eq. (2) [4], where p_n stands for the normalized power of the nth imperialist and N_{imp} stands for the total number of imperialists.

$$C_n = c_n - \max_i \{c_i\} \tag{1}$$

$$p_n = \left| \frac{C_n}{\sum_{i=1}^{N_{\text{imp}}} C_i} \right| \tag{2}$$

ICA uses the concept of 'empire' to proceed with subsequent processes. Here, an empire is made up of one imperialist and some colonies. The initial solutions will form several empires. Every initial colony would be distributed to one and only one imperialist. How many colonies an imperialist can obtain is proportional to its power and calculated by Eq. (3) [4], where $N.C._n$ and N_{col} stand for the number of the colonies distributed to the nth imperialist and the total number of the colonies, respectively. For each imperialist, $N.C._n$ colonies are randomly selected from the initial population and distributed to it.

$$N.C._n = \text{round}\{p_n \cdot N_{\text{col}}\} \tag{3}$$

2.2 Assimilation, revolution and uniting

After empires are initialized, within each empire, colonies would be moved toward the imperialist. This process is called 'assimilation,' as illustrated in Fig. 1. Each colony is

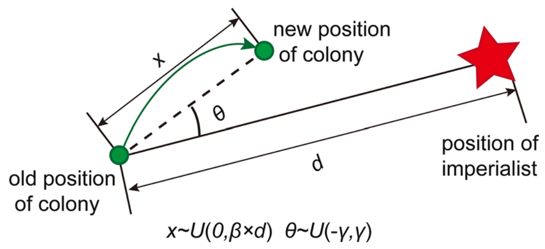

Fig. 1 Assimilation process

moved x units to the relevant imperialist every time. Meanwhile, a random amount of deviation (the θ in Fig. 1) is added to the movement direction, to enhance the exploration ability of the algorithm. x and θ are defined as shown in Fig. 1, where d is the distance between colony and the relevant imperialist; β is a number which is >1 and makes the colony to get closer to the imperialist from both sides, and γ is a parameter whose value determines the size of the deviation added to the original direction. Usually, a value of about 2 for β and a value of about $\pi/4$ (rad) for γ can get a good performance [4].

Meanwhile, some colonies would be randomly selected out according to a preset rate and then be replaced with an equal number of new randomly generated countries. This process is called 'revolution.' The revolution process is similar to the mutation operator in the genetic algorithm, used to enhance the algorithm's ability to escape from local optima and to avoid premature convergence. The rate is called 'revolution rate' [4].

In the processes mentioned above, if a colony becomes better than the relevant imperialist, their roles will be exchanged. Meanwhile, if two imperialists are moved to a similar position (the distance between them is smaller than a preset threshold distance), the two relevant empires would be united to one empire. The new empire would take over all the colonies of the two previous empires and take one of the two previous imperialists as its imperialist [4].

2.3 Competition between empires

Competition between empires is the core of the ICA. In this stage, firstly, the total cost of every empire is calculated by Eq. (4) [4], on the basis of the cost of its imperialist and colonies. Then they would be normalized by Eq. (5) [4]. Here, $T.C._n$ and $N.T.C._n$ stand for the total cost and the normalized total cost of the nth empire, respectively, and ξ is a decimal fraction, whose value determines the weight of the colonies' cost in the total cost.

$$T.C._n = \mathrm{Cost}(\mathrm{imperialist}_n) + \xi \cdot \mathrm{mean}\{\mathrm{Cost}(\mathrm{colonies\ of\ empire}_n)\} \quad (4)$$

$$N.T.C._n = T.C._n - \max_i\{T.C._i\} \quad (5)$$

Then, the weakest colony of the weakest empire would be picked out and become a temporary independent country. Other empires would contend for control of this independent country. Every imperialist may but only one can succeed finally. The success probability of each empire is proportional to its power, as that given by Eq. (6) [4].

$$P_{P_n} = \left| \frac{N.T.C._n}{\sum_{i=1}^{N_{\mathrm{imp}}} N.T.C._i} \right| \quad (6)$$

The ICA realizes the competition process described above in the following method. Firstly, the success probability of each empire would be used to constitute a vector P, as Eq. (7) [4]. And then, a vector R with the same size as P and whose elements are randomly generated in the interval [0,1] is created as Eq. (8) [4]. Then, a vector D is obtained by subtracting R from P as Eq. (9) [4]. Finally, the empire whose relevant index in D is largest will obtain the mentioned colony. This handling method is similar to but quicker than the conventional roulette wheel method in the genetic algorithm.

$$P = \left[p_{P_1}, p_{P_2}, p_{P_3}, \ldots, p_{P_{N_{\mathrm{imp}}}}\right] \quad (7)$$

$$R = \left[r_1, r_2, r_3, \ldots, r_{N_{\mathrm{imp}}}\right] \quad \text{where} \quad r_i \sim U(0,1) \quad \text{and} \quad 1 \le i \le N_{\mathrm{imp}} \quad (8)$$

$$D = \left[D_1, D_2, D_3, \ldots, D_{N_{\mathrm{imp}}}\right] = P - R = \left[p_{P_1} - r_1, p_{P_2} - r_2, p_{P_3} - r_3, \ldots, p_{P_{N_{\mathrm{imp}}}} - r_{N_{\mathrm{imp}}}\right] \quad (9)$$

In each generation of the ICA, the processes of assimilation, revolution and competition are carried out in sequence. As iteration proceeds, the weak empire steadily loses its colonies and the powerful empire obtains more and more colonies. In this process, the empire which loses all its colonies will be collapsed. The ultimate result is that there is only one empire left in the solution population. The imperialist of the empire is the solution obtained by the algorithm. Usually, a preset iterative number can be used as a termination condition. Figure 2 illustrates the entire process of the original ICA.

3 The improved ICA combined with mutation operators

3.1 Analysis of the basic ICA

At the early stage of iterations, the colonies of every empire are dispersed in the whole search space and can be moved in a wide range in the assimilation process. Meanwhile, the competition between multiple empires gives the colonies a chance to be transferred from one empire to another. These two aspects can ensure the diversity of the population. Moreover, though the imperialists are the best individuals in the current population, their qualities are still poor at this stage. So a colony has a big possibility to become better than the relevant imperialist and then replace it. Once an imperialist is replaced, the movement directions of all the colonies it controls would be changed. Thus, the diversity of the population can be enhanced further. However, as the algorithm proceeds, the

Fig. 2 Process of the original ICA

1. Randomly generate some candidate solutions for the problem and initialize empires.
2. Move the colonies toward their relevant imperialist(Assimilation).
3. Randomly pick out a part of colonies from every empire, replace them with new generated(Revolution).
4. If there is a colony in an empire which has lower cost than that of imperialist, then exchange the positions of that colony and the imperialist.
5. Compute the total cost of every empire.
6. Pick the weakest colony from the weakest empire and give it to the empire that has the most likelihood to possess it.(Imperialistic competition).
7. Uniting similar imperialists.
8. If an empire losses all colonies, collapse it.
9. If the stop condition is satisfied, stop, if not, go to step 2.

colonies gradually move closer and closer to the relevant imperialists. Accordingly, their movement ranges also become smaller and smaller. Meanwhile, the competition process would decline the number of empires. When there is only one empire left, all the colonies can only move toward the same imperialist. Obviously, at this time, the diversity of the population would decline greatly. If the remained imperialist is located on a local optimal area, the algorithm would not have enough ability to jump out.

From what have been discussed above, it can be possible to conclude that maintaining the diversity of the population is essential for enhancing the global search ability of the algorithm and avoiding premature convergence. Applying mutation operators to the individuals in the iterative process is a good choice. However, how to implement the mutation operators is the key question. From the description about the mechanism of the ICA, we can find that the imperialists play key roles during the search process. They are the best found solutions, and they would share their location information to other solutions. Therefore, we prefer to apply mutation operators on them (Fig. 3). In addition to directly increasing the diversity of the population, the mutation operators applied to the imperialists can also give them self-exploration ability. In the original algorithm, the imperialists are stationary, unless they are replaced by other better colonies. After the application of mutation operators, the imperialists can take the initiative to explore new better positions. Therefore, the algorithm would get more opportunities to keep away from stagnation.

3.2 Various mutation operators

Many different mutation operators have been proposed and applied to various meta-heuristic algorithms, such as Gaussian mutation [37–40], Cauchy mutation [37, 38, 41], Lévy mutation [42–44], exponential mutation [45], *t* mutation [46], chaotic mutation [47] and mixed or hybrid mutation [48, 49]. Among them, Gaussian mutation, Cauchy mutation and Lévy mutation are the most widely used approaches. Therefore, in our work, these three mutation operators are investigated particularly.

Though mutation operators have various forms, they share such a same basic idea: forcing the algorithm to search in new regions by the means of using random generated numbers to change the positions of the current candidate solutions. How to generate the random numbers is the main difference between different mutation operators. On the other hand, even a same kind of mutation operation can be implemented by various specific forms. In our study, mutation operators are applied to the basic ICA according to Eq. (10).

$$X_i^{j\prime} = X_i^j \cdot (1 + k \cdot N_{\text{random}}) \quad \text{where} \quad j = 1, 2, 3, \ldots, N_{\text{var}}, \tag{10}$$

where X_i^j is the *j*th variable of the *i*th imperialist, k is an additional scale parameter, and N_{random} indicates that a random number which is generated based on the mutation operators and generated anew for each variable. As already mentioned previously, three different mutation operators are investigated in our work. In them, N_{random} is generated according to different kind of random distributions.

3.2.1 Gaussian mutation

In the Gaussian mutation, the random numbers are generated based on the Gaussian distribution, whose one-dimensional probability density function can be given as Eq. (11) [50]

$$f_{\mu,\sigma^2}(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \tag{11}$$

where μ is the mean value and σ^2 is the variance. For convenience, Gaussian distribution can be described as $N(\mu, \sigma^2)$. In our work, the standard Gaussian distribution with $\mu = 0$ and $\sigma = 1$, described as $N(0,1)$, is used to generate random numbers.

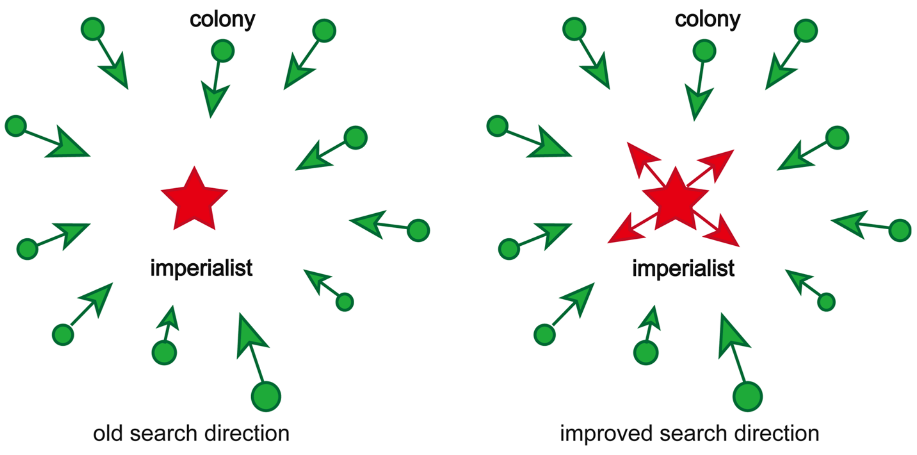

Fig. 3 Mutation operator brings the imperialist self-exploration ability

3.2.2 *Cauchy mutation*

In the Cauchy mutation, the random numbers are generated based on the Cauchy distribution, whose one-dimensional probability density function can be given as Eq. (12) [50]

$$f_t(x) = \frac{1}{\pi} \frac{t}{t^2 + (x - x_0)^2}, \tag{12}$$

where x_0 is a location parameter which specifies the location of the peak of the distribution and t is a scale parameter which specifies the half width at half maximum. For convenience, Cauchy distribution can be described as $C(t, x_0)$. In our work, the standard Cauchy distribution with $t = 1$ and $x_0 = 0$, described as $C(1, 0)$, is used to generate random numbers.

3.2.3 *Lévy mutation*

In the Lévy mutation, the random numbers are generated based on the Lévy distribution. In a sense, Lévy distribution is a generalization of Gaussian distribution and Cauchy distribution. Its probability density function can be given as Eq. (13) [38, 43, 44]

$$L_{\alpha,\gamma} = \frac{1}{\pi} \int_0^{\infty} \mathrm{e}^{-\gamma q^{\alpha}} \cos(qy) \mathrm{d}q, \tag{13}$$

where γ is the scaling factor satisfying $\gamma > 0$ and α satisfies $0 < \alpha < 2$ and controls the shape of the distribution. In our work, we use the Lévy distribution with $\gamma = 1$ and $\alpha = 1.3$, according to that recommended in the literature [51], and adopt the algorithm proposed by Mantegna [52] to obtain Lévy random numbers.

Figure 4 diagrams the difference between the probability density functions of the standard Gaussian distribution, the standard Cauchy distribution and the Lévy distribution with $\gamma = 1$ and $\alpha = 1.3$. From Fig. 4, it can be observed that these three distributions have different characteristics.

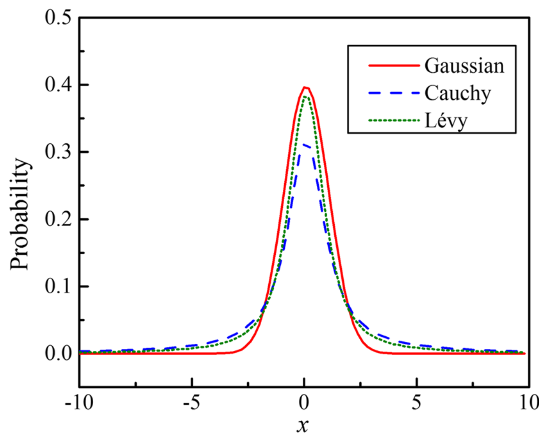

Fig. 4 Comparisons between the probability density functions of the standard Gaussian distribution, the standard Cauchy distribution and the Lévy distribution ($\gamma = 1$, $\alpha = 1.3$)

The standard Gaussian distribution tends to generate random numbers which are closer to 0, the standard Cauchy distribution tends to generate random numbers which are farther from 0, and the Lévy distribution lies between the standard Gaussian distribution and the standard Cauchy distribution. Therefore, it can be expected that the three different mutation operators could bring different effects.

As mentioned previously, in our work, the mutation operators are applied on the imperialists. Specifically, in every iteration, the selected mutation operator is applied to every imperialist dimension by dimension. The elements of the imperialist are updated one by one, according to Eq. (10). If the new element exceeds its domain, it will be restricted on the search boundaries. After finishing the update, the newly obtained solution would be evaluated immediately. Then, a greedy strategy is applied to decide whether to accept this update or not. If the new solution is better than the previous one, the update would be accepted, or otherwise the update would be abandoned.

Fig. 5 Pseudo-code of the applying process of the mutation operators

```
for (i=1; i <= the number of  imperialists in current population; i++)
  TempImperialist = the ith imperialist;
  for (j=1; j <= the number of elements in the imperialist, j++)
     update the jth element of the TempImperialist using mutation operator;
     if (the updated TempImperialist is better than the before ) then
        replace the before TempImperialist by the updated TempImperialist;
     else
        give up the updated TempImperialist;
     endif
  endfor
  the ith imperialist=theTempImperialist;
endfor
```

Figure 5 shows the pseudo-code of the mutation operators. In our method, this procedure is inserted between step 8 and step 9 in Fig. 2.

4 Numerical experiments, results and discussion

The three different mutation operators produce three different improved algorithms. For convenience, in the following paragraphs, the basic ICA, the improved ICA with Gaussian mutation, the improved ICA with Cauchy mutation and the improved ICA with Lévy mutation will be abbreviated as BICA, IICA-G, IICA-C and IICA-L, respectively.

4.1 Benchmark functions

In order to demonstrate the improved effects, all the four algorithms are tested together and then compared on nine widely used minimized benchmark functions. The dimension of every function is set to 30 in our study. The name, formula, variable range and theoretical optimal value of these benchmark functions are listed in Table 1. These functions have different characteristics: F_1, F_2, F_4, F_5 and F_6 are unimodal; F_5 is discontinuous; F_7, F_8 and F9 are multimodal functions; the number of local minima increases exponentially with the problem dimension [37].

4.2 Experiments settings

All the four algorithms are implemented in MATLAB R2012a, under a PC with Pentium 4 at 2.9 GHZ, 4 GB RAM and Windows 7 Ultimate Operating system. On every function, every algorithm is run 30 independent trials to eliminate the influence caused by the randomness of the algorithms. Meanwhile, the maximum number of fitness function evaluations (MAX_NFFEs) is used as a termination condition. Every trial ends when the MAX_NFFEs is reached. The MAX_NFFEs is set to 150,000 for F1, F5, F6 and F8, 200,000 for F2 and F7, 500,000 for F3 and F4 and 300,000 for F9, according to relevant literatures [37, 53].

To be fair, all the common parameters in the four algorithms are set to same, as:

1. The number of initial countries is set to 100.
2. The number of initial empires is set to 3.
3. The revolution rate is initialized to 0.3 and exponentially declines with the number of iterations, as that illustrated by Eq. (14).

$$\text{revolution rate} = 0.3 \times 0.99^{\text{iteration number}} \tag{14}$$

4. The ξ in the formula 4 is set to 0.02.

The number of initial countries and the number of initial empires are set through lots of experiments. These two parameters have great influences on the quality of obtained results. If they are set too small, the algorithms would be easily trapped into local optimum and obtain bad results finally, because the diversity of population cannot be guaranteed. On the other hand, if they are set too large, the fitness evaluations would be exhausted very quickly, and the algorithms also cannot obtain satisfactory results. The revolution rate and the ξ are set according to the suggestions given by the authors. These two parameters would also influence the performance of the algorithms, but the effects brought by them are relatively small.

The scale factor k in the mutation operators is set as:

1. For the Gaussian mutation, the standard Gaussian distribution is used and the scale factor k is set to 0.5.
2. For the Cauchy mutation, the standard Cauchy distribution is used and the scale factor k is set to 0.1.
3. For the Lévy mutation, α is set to 1.3 and the scale factor k is set to 0.2.

The values of the scale factor k are also determined through experiments. In a whole, a larger value of k means that the positions of the imperialist would be changed greatly after the mutation operator is applied. We tested the improve algorithms with different values of k on these

Table 1 Benchmark functions used in the experimental study

Function	Name	Variable range	Optimal value
$F_1(x) = \sum_{i=1}^{D} x_i^2$	Sphere	$[-100, 100]^D$	0
$F_2(x) = \sum_{i=1}^{D} \lvert x_i \rvert + \prod_{i=1}^{D} \lvert x_i \rvert$	Schweel 2.22	$[-10, 10]^D$	0
$F_3(x) = \sum_{i=1}^{D} \left(\sum_{j=1}^{i} x_j \right)^2$	Schweel 1.2	$[-100, 100]^D$	0
$F_4(x) = \max_{i=1}^{D} \{\lvert x_i \rvert\}$	Schweel 2.21	$[-100, 100]^D$	0
$F_5(x) = \sum_{i=1}^{D} [x_i + 0.5]^2$	Step	$[-100, 100]^D$	0
$F_6(x) = \sum_{i=1}^{D} x_i^2 + \left(\sum_{i=1}^{D} 0.5 i x_i\right)^2 + \left(\sum_{i=1}^{D} 0.5 i \cdot x_i\right)^4$	Zakharov	$[-5, 10]^D$	0
$F_7(x) = \frac{1}{4000} \times \sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	Griewank	$[-600, 600]^D$	0
$F_8(x) = -20 \exp\left(-0.2 \sqrt{\frac{1}{D} \sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{D} \sum_{k=1}^{n} \cos 2\pi x_k\right) + 20 + e$	Ackley	$[-32, 32]^D$	0
$F_9(x) = \sum_{i=1}^{D} [x_i^2 - 10 \cos(2\pi x_i) + 10]$	Rastrigin	$[-5.12, 5.12]^D$	0

functions and finally determined the values mentioned above. In what follows, a further illustration about the influence of the value of k is given.

It should be pointed out that the parameters we used here are effective, but they are not the best for specific functions. During our previous experiments, we found that for some specific functions, using other parameters could obtain better results. Without loss of generality, in our tests, fixed parameters are used for the tests on different functions. We have not made more effort in finding the best parameter settings for every specific function, because this work has exceeded the scope of this article.

4.3 Influence of the scale factor

For meta-heuristic algorithms, the values of the parameters would influence the performance greatly. The proposed approach introduces a new parameter: the scale factor k. To investigate the effect of this parameter to the performance of the proposed approach, we tested the proposed algorithms with different values of k on these functions and the results obtained on two benchmark functions: The Sphere function and the Griewank function are given here. Due to the different characteristics of the three mutation operators, the IICA-G is tested on the situations of $k = 0.25$, $k = 0.5$, and $k = 0.75$, and the IICA-C and the IICA-L are tested on the situations of $k = 0.1$, $k = 0.2$, and $k = 0.3$. For each value of k, every algorithm is tested for 20 times on each function and the results are arranged in Tables 2, 3 and 4, respectively.

From Tables 2, 3 and 4, it can be seen that the value of the scale factor k would influence the obtained results obviously. On the Sphere function, the obtained results become better and better along with the increase in the scale factor. On the Griewank function, the situation is different. The largest values of k have not produced best results. On other functions which have not been listed here, the values of k also affected the quality of the results. Based on an overall consideration of the quality of results

Table 2 Comparisons of the results of the IICA-G with different values of k on the Sphere function and the Griewank function

Function	Stats.	$k = 0.25$	$k = 0.5$	$k = 0.75$
Sphere	Min	4.1420e−112	5.0790e−296	0.0000e+000
	Mean	5.9443e−106	6.3494e−274	0.0000e+000
	Median	2.1288e−107	5.4620e−279	0.0000e+000
	Std	1.5247e−105	0.0000e+000	0.0000e+000
Griewank	Min	0.0000e+000	0.0000e+000	0.0000e+000
	Mean	5.1093e−002	5.4299e−002	7.7021e−002
	Median	5.6407e−002	5.9864e−002	5.9924e−002
	Std	9.2840e−002	4.1284e−002	6.3694e−002

Table 3 Comparisons of the results of the IICA-C with different values of k on the Sphere function and the Griewank function

Function	Stats.	$k = 0.1$	$k = 0.2$	$k = 0.3$
Sphere	Min	1.7771e−106	4.7143e−175	4.4994e−211
	Mean	6.6147e−088	1.1638e−149	1.7939e−200
	Median	5.1793e−097	1.1760e−165	1.3926e−202
	Std	2.0896e−087	3.6803e−149	0.0000e+000
Griewank	Min	0.0000e+000	0.0000e+000	3.4335e−002
	Mean	5.8997e−002	3.3568e−002	8.7684e−002
	Median	6.5948e−002	2.7064e−002	8.8157e−002
	Std	4.0419e−002	3.5881e−002	4.3512e−002

Table 4 Comparisons of the results of the IICA-L with different values of k on the Sphere function and the Griewank function

Function	Stats.	$k = 0.1$	$k = 0.2$	$k = 0.3$
Sphere	Min	8.6590e−072	4.4194e−140	1.1545e−193
	Mean	9.7231e−063	1.2557e−135	1.1556e−179
	Median	1.3851e−064	2.6002e−136	9.7218e−186
	Std	2.3844e−062	1.9439e−135	0.0000e+000
Griewank	Min	0.0000e+000	0.0000e+000	2.9459e−002
	Mean	5.6822e−002	5.4070e−002	6.6354e−002
	Median	5.0315e−002	4.2991e−002	6.3717e−002
	Std	4.7137e−002	4.7780e−002	2.4397e−002

obtained on these functions, we determined to set these parameters to 0.5 for the IICA-G, 0.1 for the IICA-C and 0.2 for the IICA-L, to guarantee the overall performance of the proposed algorithms.

4.4 Results and discussion

The statistic results of the four algorithms are arranged in Table 5, where 'Min(best),' 'Mean,' 'Median,' 'Max(-worst)' and 'Std' represent the best obtained result, the average value of all the obtained results, the median value of all the obtained results, the worst obtained result and the standard deviation of all the obtained results of the corresponding algorithm, respectively; 'Run time' denotes the average computation time of the 30 trials for every function, reflecting the computational complexity of the algorithms. The best results have been detached in bold.

From Table 5, it can be observed that all the three mutation operators bring the basic algorithm effective improvements. On every tested function, compared with the results obtained by the BICA, the results obtained by the IICA-G, the IICA-C and the IICA-L are closer to the theoretical optimal value. Meanwhile, the three improved algorithms show better stability and robustness. The results obtained by them are more stable than that obtained by the BICA. For instance, on F1, the results obtained by the BICA vary from 2.2725e−002 (the best result) to 2.0000e+004 (the worst result), which is a relatively large range, while the results obtained by the three improved algorithms are more concentrated. A similar situation can also be observed on the F2, F3, F5, F7 and F9.

There are some differences between the results obtained by the three improved algorithms. On F1, F2 and F4, the results obtained by the IICA-G are significantly better than that obtained by the IICA-C and the IICA-L; on F3, F5 and F6, the results obtained by the IICA-C are much better; on F7, all the three improved algorithms obtained the theoretical optimal value, and the difference between their results is very slightly because they are located on the same order of magnitude; on F8, the results obtained by the three improved algorithms also have the same order of magnitude, and the results obtained by the IICA-C are slightly better. Only on F9, there is no difference between the finally results obtained by the three algorithms. All the three algorithms obtained the theoretical optimal value in every run.

Meanwhile, it can also be observed that though the three improved algorithms show superiority than the BICA on every function, the improvements shown on different function are different. On F1, F2, F3, F4, F5, F8 and F9, the improvements are significant, while on F6 and F7, they are not so significant.

Furthermore, from Table 5, it can also be observed that the mutation operators increase the cost time, while at the same time enhancing the performance of the BICA. However, the increases are slight and acceptable.

In order to further study the difference among the behaviors of the four algorithms, the convergence curves of the four algorithms on every function are given in Fig. 6. The convergence curve is obtained by averaging the variations of the best obtained result with the NFFEs over the 30 runs.

From Fig. 6, it can be observed that there are significant differences between the convergence curves of the four algorithms. Compared with the convergence curves of the BICA, the convergence curves of the three improved algorithms are steeper, indicating that the improved algorithms have faster convergence rate than the BICA. On F1–F6, the curves of the BICA become flat before the Max_NFFEs is reached, indicating that the BICA has got trapped into local optimum, while the curves of the three improved approaches decline steadily throughout the whole solving process. It can be predicted that if the Max_NFFEs is increased, the BICA cannot improve the obtained solutions further, while the three improved approaches with mutation operators can do that. On F7 and F8, all the curves of the four algorithms become flat before the Max_NFFEs is reached. However, the curves of the three improved algorithms converge faster and converge to more accurate solutions finally. On F9, the curve of the BICA becomes flat soon after the trials begin, while the curves of the three improved algorithms decline steadily. Finally, the curve of the IICA-G converges to the theoretical optimal value after about 50,000 NFFEs, and the curves of the IICA-C and the IICA-L also converge to the theoretical optimal value after about 150,000 NFFEs and about 120,000 NFFEs, respectively.

Meanwhile, it can be observed that on every function except F5 and F6, the IICA-G shows the fastest convergence rate, the IICA-L occupies the second position, and the IICA-C shows slowest convergence rate. On F5 and F6,

Table 5 Comparisons of the results obtained by the BICA, the IICA-G, the IICA-C and the IICA-L on the tested benchmark functions

Function	Stats.	BICA	IICA-G	IICA-C	IICA-L
F1	Min(best)	2.2725e−002	**5.8865e−295**	8.9250e−105	2.0952e−143
	Mean	3.0014e+003	**7.9783e−270**	3.0161e−085	4.2975e−125
	Median	1.4174e+000	**2.4519e−281**	1.1919e−094	4.0116e−135
	Max(worst)	2.0000e+004	**2.0760e−268**	8.9934e−084	1.2864e−123
	Std	5.3492e+003	**0.0000e+000**	1.6416e−084	2.3484e−124
	Run time(s)	**8.8156e+000**	1.0710e+001	9.8775e+000	1.0301e+001
F2	Min(best)	4.0001e+001	**3.9419e−208**	6.9182e−074	3.5339e−100
	Mean	8.4001e+001	**1.6277e−188**	7.4268e−063	4.1246e−086
	Median	8.5007e+001	**8.9486e−199**	9.2948e−068	4.2744e−094
	Max(worst)	1.2000e+002	**4.2967e−187**	1.8805e−061	1.2373e−084
	Std	2.2221e+001	**0.0000e+000**	3.4476e−062	2.2591e−085
	Run time(s)	**1.2293e+001**	1.5327e+001	1.4224e+001	1.4760e+001
F3	Min(best)	5.0001e+003	2.1446e−002	**1.9041e−006**	5.0878e−004
	Mean	3.2697e+004	8.1017e−001	**4.6869e−004**	3.6854e−002
	Median	3.4167e+004	5.3242e−001	**1.8374e−004**	8.9871e−003
	Max(worst)	6.1667e+004	5.6179e+000	**3.5259e−003**	2.2999e−001
	Std	1.4282e+004	1.1145e+000	**7.4389e−004**	5.7622e−002
	Run time(s)	**8.2037e+001**	1.0697e+002	1.0186e+002	1.0442e+002
F4	Min(best)	7.3937e−001	**1.9000e−025**	2.8627e−012	5.3067e−014
	Mean	1.3653e+000	**2.2029e−023**	5.4118e−011	4.2560e−013
	Median	1.3252e+000	**9.4330e−024**	3.8589e−011	1.9079e−013
	Max(worst)	2.3747e+000	**2.2938e−022**	1.7137e−010	1.6640e−012
	Std	4.0130e−001	**4.2625e−023**	4.6253e−011	4.6041e−013
	Run time(s)	**1.4207e+001**	1.6222e+001	1.7371e+001	1.7632e+001
F5	Min(best)	4.4430e−002	3.3098e−008	**2.7614e−009**	3.6218e−009
	Mean	2.3309e+003	1.2741e−007	**8.6985e−009**	2.1788e−008
	Median	4.1674e−001	1.1570e−007	**7.4352e−009**	1.8753e−008
	Max(worst)	2.0201e+004	3.0886e−007	**3.0155e−008**	6.8125e−008
	Std	5.0486e+003	7.5029e−008	**5.9357e−009**	1.5291e−008
	Run time(s)	**1.3179e+001**	1.5015e+001	1.4151e+001	1.4620e+001
F6	Min(best)	3.7284e+002	2.5406e+001	**6.7272e+000**	2.1371e+001
	Mean	5.7281e+002	8.9470e+001	**2.8515e+001**	4.6687e+001
	Median	5.7082e+002	8.3778e+001	**2.6689e+001**	4.6148e+001
	Max(worst)	8.5021e+002	1.6394e+002	**5.9512e+001**	9.1294e+001
	Std	1.2545e+002	3.7148e+001	**1.1570e+001**	1.7290e+001
	Run time(s)	**1.5978e+001**	2.0093e+001	1.8302e+001	1.9177e+001
F7	Min(best)	7.6503e−003	**0.0000e+000**	**0.0000e+000**	**0.0000e+000**
	Mean	2.7179e+001	6.4241e−002	6.4898e−002	**6.2983e−002**
	Median	9.0283e−002	5.6283e−002	**5.4035e−002**	5.6289e−002
	Max(worst)	1.8098e+002	**1.8528e−001**	1.9265e−001	1.8761e−001
	Std	4.8312e+001	**5.0310e−002**	5.0447e−002	5.0348e−002
	Run time(s)	**2.0534e+001**	2.3581e+001	2.2442e+001	2.3043e+001
F8	Min(best)	6.9381e−002	3.2863e−014	**2.5757e−014**	3.2863e−014
	Mean	1.6654e+001	5.3113e−014	**4.7902e−014**	5.3350e−014
	Median	1.7791e+001	**5.0626e−014**	**5.0626e−014**	**5.0626e−014**
	Max(worst)	1.9963e+001	7.9048e−014	**7.5495e−014**	**7.5495e−014**
	Std	4.9684e+000	1.1584e−014	**1.0584e−014**	1.1145e−014
	Run time(s)	**1.4359e+001**	1.5986e+001	1.5265e+001	1.5680e+001
F9	Min(best)	7.8672e+001	**0.0000e+000**	**0.0000e+000**	**0.0000e+000**
	Mean	1.3836e+002	**0.0000e+000**	**0.0000e+000**	**0.0000e+000**

Table 5 continued

Function	Stats.	BICA	IICA-G	IICA-C	IICA-L
	Median	1.3855e+002	**0.0000e+000**	**0.0000e+000**	**0.0000e+000**
	Max(worst)	1.9842e+002	**0.0000e+000**	**0.0000e+000**	**0.0000e+000**
	Std	3.3194e+001	**0.0000e+000**	**0.0000e+000**	**0.0000e+000**
	Run time(s)	**2.2699e+001**	2.3123e+001	2.4582e+001	2.3395e+001

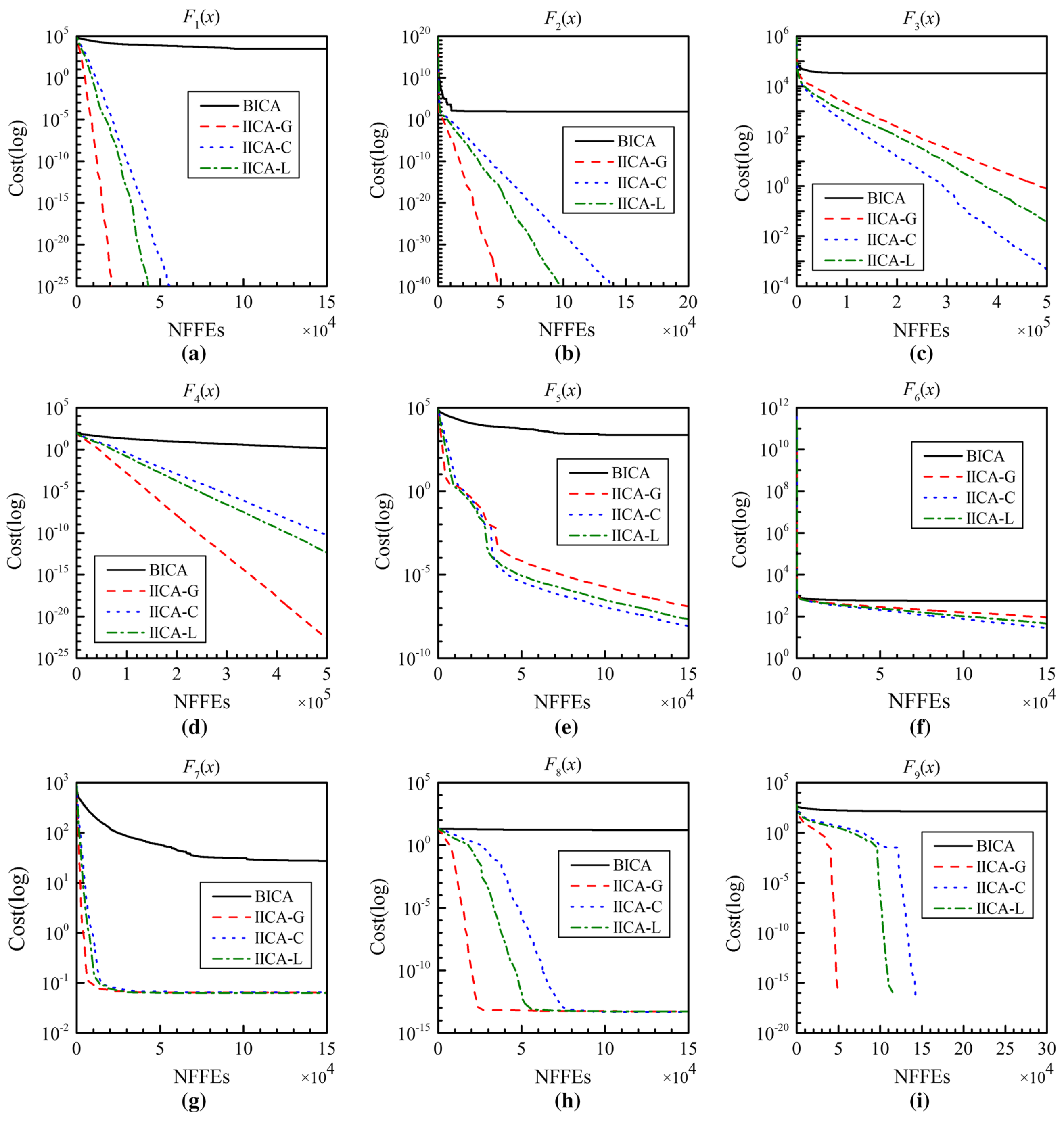

Fig. 6 Convergence curves of the four algorithms on every function

the situation is the opposite. The IICA-C shows the fastest convergence rate, while the IICA-G shows the slowest convergence rate.

As what have been mentioned, the improvement effects brought by the three mutation operators are significant on some functions but not significant on some others. Meanwhile, the three improved algorithms show different behaviors on different functions. This is caused by the different characteristics of different functions. As we know, though meta-heuristic algorithms have the characteristics of generality, a specific meta-heuristic algorithm cannot be suitable to every benchmark function. In other words, the inherent characteristics would influence the performance of one algorithm greatly.

4.5 Comparison with other works

In this section, we compare the proposed approach with two other works. The first contrastive algorithm is the imperialist competitive algorithm combined with chaos (CICA), which is proposed in the literature [34] and used here as a representative of the previous improved approaches of the basic ICA; the second contrastive algorithm is the real coded genetic algorithm approach with random transfer vectors-based mutation (RCGA-RTVM), which is proposed in the literature [54] and used here as a representative of various meta-heuristic algorithms, due to its outstanding performance which has been proven in the literature [54].

In the literature [34], the CICA is tested on four $10D$ benchmark functions. To compare the IICAs we proposed with the CICA, we also test the IICAs on these four functions. Meanwhile, the results given in the literature [34] are directly taken into the comparisons, and every improved ICA we proposed is run 30 independent trials. To be fair, we use the same iterations number and search ranges given in the literature [34]. And the other parameters we used are as same as that used in Sect. 4.2. The results are arranged in Table 6. The best results have been detached in bold.

From Table 6, it can be clearly observed that on the Griewank function and the Rosenbrock function, the CICA shows better performance than the three IICAs proposed in this work. The results of the CICA are more accurate. While on the Rastrigin function and the Ackley function, the IICAs obtained better results: On Rastrigin function, the three improved algorithms proposed in this work obtained the optimal value in every trial; on Ackley function, the three IICAs also obtained better results and showed better stability.

In the literature [54], the RCGA-RTVM is tested on four $30D$ functions and one $2D$ function. To make the comparisons, we also tested the IICAs on these five functions. Each IICA is also run 30 independent trials on each function, and the MAX_NFFEs of each trial, the search range and the number of initial solutions are as same as those used in the literature [54]. The other parameters in the IICAs are as same as that presented in Sect. 4.2. The results of the comparison are arranged in Table 7, where the results of the RCGA-RTVM are directly taken from the literature [54], and the best results have also been detached in bold.

From Table 7, it can be seen that on the Rosenbrock function and the Step function, the improved ICAs obtained better results. The results are closer to the theoretical optimal value. On the Schweel 1.2 function and the Schweel 2.21 function, the RCGA-RTVM performs better than the three different IICAs. The results it obtained are significantly more accurate. On the Six-hump camel-back, because this function is relatively simple, all the four algorithms show satisfactory performances, and there is no significant difference between their results.

In summary, the proposed IICAs show better performance than the CICA and the RCGA-RTVM on some functions and show worse performance on some other functions. Each algorithm has its own specialties and is more suitable for some problems. In practice, when using meta-heuristic algorithm to deal with problem, usually the problem is regarded as a black box, and its characteristics cannot be obtained. Usually, different algorithms should be tried until a satisfactory solution is obtained. In these cases, the improved algorithms proposed in this work can be used as options.

4.6 An application of the proposed algorithms

In this section, we use a real-world optimization case to demonstrate the applicability of the proposed algorithms. This problem is a classical benchmark problem which aims to minimize the total weight of a simple gear box, which is shown in Fig. 7. It involves seven design variables, the face width b (x_1), module of teeth m (x_2), number of teeth on pinion z (x_3), length of first shaft between bearings l_1 (x_4), length of second shaft between bearings l_2 (x_5), diameter of first shaft d_1 (x_6) and diameter of second shaft d_2 (x_7).

Table 6 Comparisons of the results obtained by the CICA [34], the IICA-G, the IICA-C and the IICA-L on four different functions

Function	Stats.	CICA	IICA-G	IICA-C	IICA-L
Griewank	Min(best)	**1.17e−16**	2.95e−02	1.23e−02	7.39e−03
	Mean	**3.48e−14**	4.40e−02	4.08e−02	5.26e−02
	Max(worst)	**2.58e−12**	5.90e−02	7.38e−02	9.34e−02
	Std	**5.07e−15**	1.13e−02	1.91e−02	2.62e−02
Rosenbrock	Min(best)	**1.82e−04**	2.27e−02	2.22e−01	5.09e−01
	Mean	**2.42e−02**	1.72e+00	9.04e−01	1.03e+00
	Max(worst)	**7.18e−02**	3.84e+00	1.89e+00	2.06e+00
	Std	**2.19e−02**	1.42e+00	6.05e−01	4.97e−01
Rastrigin	Min(best)	**0**	**0**	**0**	**0**
	Mean	9.34e−09	**0**	**0**	**0**
	Max(worst)	1.07e−07	**0**	**0**	**0**
	Std	3.43e−08	**0**	**0**	**0**
Ackley	Min(best)	5.80e−08	**7.99E−15**	**7.99E−15**	**7.99E−15**
	Mean	1.02e−07	**1.23E−14**	1.30E−14	1.37E−14
	Max(worst)	5.14e−06	2.93E−14	**1.51E−14**	2.93E−14
	Std	1.24e−07	6.86E−14	**3.43E−15**	7.34E−15

Table 7 Comparisons of the results obtained by the RCGA-RTVM [54], the IICA-G, the IICA-C and the IICA-L on five different functions

Function	Stats.	RCGA-RTVM	IICA-G	IICA-C	IICA-L
Rosenbrock (30*D*)	Mean	2.8988E+01	2.5075e+01	**2.3115e+01**	2.4363e+01
	Std	6.7394E−01	1.4043e+00	8.9919e−01	**4.0048e−01**
Schweel 1.2 (30*D*)	Mean	**7.5456E−242**	9.3308E−37	4.4802E−02	4.4267E−03
	Std	**0**	2.6685e+00	5.0505E−02	1.2639E−02
Step (30*D*)	Mean	2.0000E−04	1.5568e−08	**2.4265e−10**	6.3296e−10
	Std	1.4142E−02	3.6954e−08	**3.1147e−10**	4.3038e−10
Six-hump camel-back (2*D*)	Mean	**−1.0316E+00**	**−1.0316E+00**	**−1.0316E+00**	**−1.0316E+00**
	Std	**2.8796E−11**	2.8547e−06	5.4013E−07	6.7907E−07
Schweel 2.21 (30*D*)	Mean	**7.4950E−24**	4.8494e−11	7.5750e−05	3.3035e−06
	Std	**1.0434E−23**	4.4586e−11	7.8729e−05	1.5067e−06

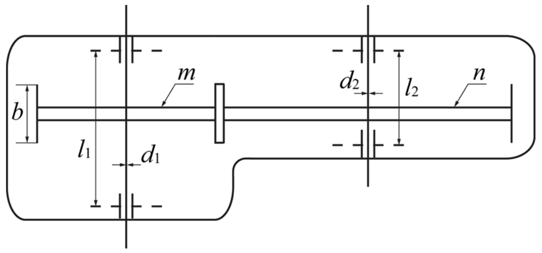

Fig. 7 Structure of the gear box to be optimized

Moreover, this problem involves seven nonlinear and four linear constraints, limitations on the bending stress of gear teeth, surface stress, transverse deflections of shafts 1 and 2 due to transmitted force and stresses in shafts 1 and 2. Due to the fact that the third design variable (the number of teeth on pinion) should only be integral, this problem is a mixed integral-continuous constrained problem. The mathematical formulation of this problem can be summarized as follows [55]:

$$f(x) = 0.7854x_1x_2^2(3.3333x_3^2 + 14.9334x_3 - 43.0934) - 1.508x_1(x_6^2 + x_7^2) + 7.4777(x_6^3 + x_7^3) + 0.7854(x_4x_6^2 + x_5x_7^2)$$

Subject to:

$$g_1(x) = \frac{27}{x_1x_2^2x_3} - 1 \le 0$$

$$g_2(x) = \frac{397.5}{x_1x_2^2x_3^2} - 1 \le 0$$

$$g_3(x) = \frac{1.93x_4^3}{x_2x_3x_6^4} - 1 \le 0$$

$$g_4(x) = \frac{1.93x_5^3}{x_2x_3x_7^4} - 1 \le 0$$

$$g_5(x) = \frac{\sqrt{\left(\frac{745x_4}{x_2x_3}\right)^2 + 16.9e6}}{110x_6^3} - 1 \le 0$$

$$g_6(x) = \frac{\sqrt{\left(\frac{745x_5}{x_2x_3}\right)^2 + 157.5e6}}{85x_7^3} - 1 \le 0$$

$$g_7(x) = \frac{x_2x_3}{40} - 1 \le 0$$

$$g_8(x) = \frac{5x_2}{x_1} - 1 \le 0$$

$$g_9(x) = \frac{x_1}{12x_2} - 1 \le 0$$

$$g_{10}(x) = \frac{1.56x_6 + 1.9}{x_4} - 1 \le 0$$

$$g_{11}(x) = \frac{1.56x_7 + 1.9}{x_5} - 1 \le 0$$

$$2.6 \le x_1 \le 3.6,\ 0.7 \le x_2 \le 0.8,\ 17 \le x_3 \le 28,\ 7.3 \le x_4 \le 8.3,\ 7.3 \le x_5 \le 8.3,\ 2.9 \le x_6 \le 3.9,\ 5.0 \le x_7 \le 5.5$$

To handle the constraints that exist in this problem, the penalty function method is adopted. Penalty function method is a commonly used constraint-handling method which can transform a constrained problem to an unconstrained problem by the aid of a self-defined penalty function. In this work, the penalty function is defined as Eq. (15).

$$F(x) = f(x) + \Delta \cdot \sum_{i=1}^{N} \max(0, g_i(x))^2, \tag{15}$$

where $F(x)$ is the fitness value of a solution x in the transformed problem; $f(x)$ is the original objective function value; N is the total number of constraints; Δ is the penalty factor, and it is set to 1e20 in our work.

The BICA, the IICA-G, the IICA-C and the IICA-L are tested together on this problem and then compared with other four algorithms. Because the dimension of this problem is relatively low, in our study, the number of countries is set to 25, the number of imperialists is set to 2, the maximum number of fitness function evaluations is set to 20,000, and every algorithm is tested 10 independent times. The results of the four algorithms are arranged in Table 8.

Table 8 Comparisons of the results obtained by the BICA, the IICA-G, the IICA-C and the IICA-L on the gear box design problem

Stats.	BICA	IICA-G	IICA-C	IICA-L
Best	3007.436552	2994.471066	2994.471066	2994.471066
Mean	3085.818289	2994.471066	2994.471066	2994.471066
Std	7.4198e+01	1.8686e−11	2.9240e−11	3.6929e−11

As a classical benchmark case, this problem has been solved by many different algorithms. In our study, we compared the best solutions obtained by the three improved ICAs with that obtained by other four existing algorithms, the society and civilization algorithm (SCA) [56], the Evolution Strategy ($\mu + \lambda$-ES) [57], the artificial bee colony algorithm (ABC) [55] and the cuckoo search algorithm (CS) [58]. The parameter and constraint values of the best solutions obtained by different algorithms are arranged in Table 9.

From Table 9, it can be observed that the results obtained by the three improved ICAs are better than that obtained by the other algorithms. Though all the best solutions obtained by the algorithms listed in the table are feasible because no any constraint is violated, the corresponding objective function value of the solutions obtained by the IICA-G, the IICA-C and the IICA-L is smallest. This case suggests that the proposed algorithms have good applicability.

5 Conclusion and future work

In this work, an improved method for the imperialist competitive algorithm to enhance its performance is proposed by introducing mutation operators into the original algorithm. In the proposed approach, the mutation operators are applied on the imperialist to change their behaviors and then to obtain better performance. The proposed approach is simple in structure and easy to be carried out. Based on the proposed method, three improved algorithms are obtained by using three different mutation operators, the Gaussian mutation, the Cauchy mutation and the Lévy mutation, and then are investigated particularly by experiments. The experimental results suggest that they can effectively improve the performance of the original algorithm. Compared with the original ICA, the three obtained

Table 9 Comparisons of the parameter and constraint values of the best solutions obtained by the SCA [56], the ($\mu + \lambda$)-ES [57], the ABC [55], the CS [58], the IICA-G, the IICA-C and the IICA-L for the gear box design problem

	SCA	($\mu + \lambda$)-ES	ABC	CS	IICA-G	IICA-C	IICA-L
x_1	3.500000	3.499999	3.499999	3.5015	3.500000	3.500000	3.500000
x_2	0.700000	0.699999	0.7	0.7000	0.700000	0.700000	0.700000
x_3	17	17	17	17	17	17	17
x_4	7.327602	7.300000	7.3	7.6050	7.300000	7.300000	7.300000
x_5	7.715321	7.800000	7.8	7.8181	7.715320	7.715320	8.300000
x_6	3.350267	3.350215	3.350215	3.3520	3.350215	3.350215	3.350215
x_7	5.286655	5.286683	5.287800	5.2875	5.286654	5.286654	5.286859
g_1	−0.073915	−0.073915	−0.073915	−0.0743	−0.073915	−0.073915	−0.073915
g_2	−0.197999	−0.197998	−0.197999	−0.1983	−0.197999	−0.197999	−0.197999
g_3	−0.493501	−0.499172	−0.499172	−0.4349	−0.499172	−0.499172	−0.499172
g_4	−0.904644	−0.901472	−0.901555	−0.9008	−0.904644	−0.904644	−0.904644
g_5	0.000000	0.000000	0.000000	−0.0011	−0.000000	−0.000000	−0.000000
g_6	0.000633	0.000000	0.000000	−0.0004	0.000000	0.000000	0.000000
g_7	−0.7025	−0.702500	−0.7025	−0.7025	−0.702500	−0.702500	−0.702500
g_8	0.000000	0.000000	0.000000	−0.0004	0.000000	0.000000	0.000000
g_9	−0.583333	−0.583333	−0.583333	−0.5832	−0.795833	−0.795833	−0.795833
g_{10}	−0.054889	−0.051325	−0.051326	−0.0890	−0.023790	−0.023790	−0.023790
g_{11}	0.000000	−0.010852	−0.010695	−0.0130	0.000000	0.000000	0.000000
$f(x)$	2994.744241	2996.348094	2997.058412	3000.9810	2994.471066	2994.471066	2994.471066

improved algorithms have faster convergence rate, better global search ability and better stability and would not dramatically increase the time consumption.

Meanwhile, the three improved algorithms are also compared with two other excellent algorithms. The comparative results suggest that the proposed algorithms have their own advantages. They can obtain better results than the contrastive algorithm on some functions. Therefore, the proposed algorithms can be used as an option in practical applications.

Finally, the three improved algorithms are tested on a real-world optimization case and compared with other four existing algorithms. The results suggest that the proposed algorithms have good applicability. All the three variants with different mutation operators can obtain better solutions than the four contrastive algorithms.

This work suggests that changing the behavior of the imperialists can enhance the performance of the original algorithm. Therefore, more attention should be paid to the behavior of the imperialists in future work. Additionally, the proposed improved algorithms should be used to solve more real-world optimization problems in future.

Acknowledgments We sincerely appreciate the supports offered by the Specialized Research Fund for the Doctoral Program of Higher Education (Grant No. 20110131110042), and the National High Technology Research and Development Program 863 (No. 2008AA04Z130). Meanwhile, we sincerely appreciate editor and anonymous reviewers for their valuable comments and suggestions.

References

1. Antoniou A, Lu W-S (2007) Practical optimization: algorithms and engineering applications, 2007th edn. Springer, New York
2. Fletcher R (2000) Practical methods of optimization, 2nd edn. Wiley, Great Britain
3. Sun W, Yuan Y-X (2006) Optimization theory and methods: nonlinear programming, vol 1. Springer, New York
4. Atashpaz-Gargari E, Lucas C (2007) Imperialist competitive algorithm: an algorithm for optimization inspired by imperialistic competition. In: 2007 IEEE congress on evolutionary computation, CEC 2007, Singapore 25–28 Sept 2007. IEEE Computer Society, Piscataway, United States, pp 4661–4667
5. Hadidi A, Hadidi M, Nazari A (2012) A new design approach for shell-and-tube heat exchangers using imperialist competitive algorithm (ICA) from economic point of view. Energy Convers Manag 67(2):66–74
6. Karami A, Rezaei E, Shahhosseni M, Aghakhani M (2012) Optimization of heat transfer in an air cooler equipped with classic twisted tape inserts using imperialist competitive algorithm. Exp Thermal Fluid Sci 38(4):195–200
7. Mozafari H, Ayob A, Kamali F (2012) Optimization of functional graded plates for buckling load by using imperialist competitive algorithm. Proc Technol 1:144–152
8. Yousefi M, Darus AN, Mohammadi H (2012) An imperialist competitive algorithm for optimal design of plate-fin heat exchangers. Int J Heat Mass Transf 55(11–12):3178–3185
9. Lucas C, Nasiri-Gheidari Z, Tootoonchian F (2010) Application of an imperialist competitive algorithm to the design of a linear induction motor. Energy Convers Manag 51(7):1407–1411
10. Mohammadi-Ivatloo B, Rabiee A, Soroudi A, Ehsan M (2012) Imperialist competitive algorithm for solving non-convex dynamic economic power dispatch. Energy 44(1):228–240 (As the access to this document is restricted, you may want to look for

a different version under “Related research” (further below) or for a different version of it)
11. Shabani H, Vahidi B, Ebrahimpour M (2013) A robust PID controller based on imperialist competitive algorithm for load-frequency control of power systems. ISA Trans 52(1):88–95
12. Tamimi A, Sadjadian H, Omranpour H (2010) Mobile robot global localization using imperialist competitive algorithm. In: Advanced computer theory and engineering (ICACTE), 2010 3rd international conference on, 2010, pp V5-524–V525-529
13. Bagher M, Zandieh M, Farsijani H (2011) Balancing of stochastic U-type assembly lines: an imperialist competitive algorithm. Int J Adv Manuf Technol 54(1–4):271–285
14. Banisadr AH, Zandieh M, Mahdavi I (2013) A hybrid imperialist competitive algorithm for single-machine scheduling problem with linear earliness and quadratic tardiness penalties. Int J Adv Manuf Technol 65(5–8):981–989
15. Behnamian J, Zandieh M (2011) A discrete colonial competitive algorithm for hybrid flowshop scheduling to minimize earliness and quadratic tardiness penalties. Expert Syst Appl 38(12): 14490–14498
16. Karimi N, Zandieh M, Najafi AA (2011) Group scheduling in flexible flow shops: a hybridised approach of imperialist competitive algorithm and electromagnetic-like mechanism. Int J Prod Res 49(16):4965–4977
17. Nazari-Shirkouhi S, Eivazy H, Ghodsi R, Rezaie K, Atashpaz-Gargari E (2010) Solving the integrated product mix-outsourcing problem using the imperialist competitive algorithm. Expert Syst Appl 37(12):7615–7626
18. Sadigh AN, Mozafari M, Karimi B (2012) Manufacturer–retailer supply chain coordination: a bi-level programming approach. Adv Eng Softw 45(1):144–152
19. Kaveh A, Talatahari S (2010) Optimum design of skeletal structures using imperialist competitive algorithm. Comput Struct 88(21):1220–1229
20. Talatahari S, Kaveh A, Sheikholeslami R (2012) Chaotic imperialist competitive algorithm for optimum design of truss structures. Struct Multidiscip Optim 46(3):355–367
21. Ahmadi MA, Ebadi M, Shokrollahi A, Majidi SMJ (2013) Evolving artificial neural network and imperialist competitive algorithm for prediction oil flow rate of the reservoir. Appl Soft Comput 13(2):1085–1098
22. Berneti SM (2013) A hybrid approach based on the combination of adaptive neuro-fuzzy inference system and imperialist competitive algorithm: oil flow rate of the wells prediction case study. Int J Comput Intell Syst 6(2):198–208
23. Berneti SM, Shahbazian M (2011) An imperialist competitive algorithm artificial neural network method to predict oil flow rate of the wells. Int J Comput Appl 26(10):47–50
24. Jula A, Othman Z, Sundararajan E (2015) Imperialist competitive algorithm with PROCLUS classifier for service time optimization in cloud computing service composition. Expert Syst Appl 42(1):135–145
25. Duan H, Xu C, Liu S, Shao S (2010) Template matching using chaotic imperialist competitive algorithm. Pattern Recogn Lett 31(13):1868–1875
26. Ebrahimzadeh A, Addeh J, Rahmani Z (2012) Control chart pattern recognition using K-MICA clustering and neural networks. ISA Trans 51(1):111–119
27. Mahmoudi MT, Taghiyareh F, Forouzideh N, Lucas C (2013) Evolving artificial neural network structure using grammar encoding and colonial competitive algorithm. Neural Comput Appl 22(1 Supplement):1–16
28. Mousavi SM, Tavakkoli-Moghaddam R, Vahdani B, Hashemi H, Sanjari MJ (2013) A new support vector model-based imperialist competitive algorithm for time estimation in new product development projects. Robot Comput Integr Manuf 29(1): 157–168
29. Niknam T, Taherian Fard E, Pourjafarian N, Rousta A (2011) An efficient hybrid algorithm based on modified imperialist competitive algorithm and K-means for data clustering. Eng Appl Artif Intell 24(2):306–317. doi:10.1016/j.engappai.2010.10.001
30. Razmjooy N, Mousavi BS, Soleymani F (2013) A hybrid neural network imperialist competitive algorithm for skin color segmentation. Math Comput Model 57(3–4):848–856
31. Bahrami H, Faez K, Abdechiri M Imperialist competitive algorithm using chaos theory for optimization (CICA). In: Computer modelling and simulation (UKSim), 2010 12th international conference on, 2010, pp 98–103
32. Abdechiri M, Faez K, Bahrami H. Adaptive imperialist competitive algorithm (AICA). In: Cognitive informatics (ICCI), 2010 9th IEEE international conference on, 2010. IEEE, pp 940–945
33. Jun-Lin L, Hung-Chjh C, Yu-Hsiang T, Chun-Wei C. Improving imperialist competitive algorithm with local search for global optimization. In: Modelling symposium (AMS), 2013 7th Asia, 23–25 July 2013, pp 61–64. doi:10.1109/AMS.2013.14
34. Talatahari S, Farahmand Azar B, Sheikholeslami R, Gandomi A (2012) Imperialist competitive algorithm combined with chaos for global optimization. Commun Nonlinear Sci Numer Simul 17(3):1312–1319
35. Arish S, Amiri A, Noori K (2014) FICA: fuzzy imperialist competitive algorithm. J Zhejiang Univ Sci C 15(5):363–371
36. Kaveh A, Talatahari S (2010) Imperialist competitive algorithm for engineering design problems. Asian J Civil Eng 11(6): 675–697
37. Yao X, Liu Y, Lin G (1999) Evolutionary programming made faster. Evolut Comput IEEE Trans 3(2):82–102
38. Gong W, Cai Z, Ling CX, Li H (2010) A real-coded biogeography-based optimization with mutation. Appl Math Comput 216(9):2749–2758
39. Tinós R, Yang S (2011) Use of the q-Gaussian mutation in evolutionary algorithms. Soft Comput 15(8):1523–1549
40. Xinchao Z (2011) Simulated annealing algorithm with adaptive neighborhood. Appl Soft Comput 11(2):1827–1836
41. Ali M, Pant M (2011) Improving the performance of differential evolution algorithm using Cauchy mutation. Soft Comput 15(5):991–1007
42. Chen M-R, Lu Y-Z, Yang G (2007) Population-based extremal optimization with adaptive Lévy mutation for constrained optimization. In: Computational intelligence and security. Springer, pp 144–155
43. He R-J, Yang Z-Y (2012) Differential evolution with adaptive mutation and parameter control using Lévy probability distribution. J Comput Sci Technol 27(5):1035–1055
44. Lee C-Y, Yao X (2004) Evolutionary programming using mutations based on the Lévy probability distribution. Evolut Comput IEEE Trans 8(1):1–13
45. Narihisa H, Taniguchi T, Ohta M, Katayama K (2005) Evolutionary programming with exponential mutation. In: Proceedings of the IASTED artificial intelligence and soft computing, Benidorn, Spain, pp 55–50
46. Gong W, Cai Z, Lu X, Jiang S A new mutation operator based on the T probability distribution in evolutionary programming. In: Cognitive informatics, 2006. ICCI 2006. 5th IEEE international conference on, 2006. IEEE, pp 675–679
47. Coelho LdS (2008) A quantum particle swarm optimizer with chaotic mutation operator. Chaos Solitons Fractals 37(5): 1409–1418
48. Sindhya K, Ruuska S, Haanpää T, Miettinen K (2011) A new hybrid mutation operator for multiobjective optimization with differential evolution. Soft Comput 15(10):2041–2055

49. Dong H, He J, Huang H, Hou W (2007) Evolutionary programming using a mixed mutation strategy. Inf Sci 177(1):312–327
50. Wu Q (2011) Cauchy mutation for decision-making variable of Gaussian particle swarm optimization applied to parameters selection of SVM. Expert Syst Appl 38(5):4929–4934
51. Lee C-Y, Yao X Evolutionary algorithms with adaptive Lévy mutations. In: Evolutionary computation, 2001. Proceedings of the 2001 congress on, 2001. IEEE, pp 568–575
52. Mantegna RN (1994) Fast, accurate algorithm for numerical simulation of Levy stable stochastic processes. Phys Rev E 49(5):4677
53. Gong W, Cai Z, Ling C (2010) DE/BBO: a hybrid differential evolution with biogeography-based optimization for global numerical optimization. Soft Comput 15(4):645–665. doi:10.1007/s00500-010-0591-1
54. Haghrah A, Mohammadi-Ivatloo B, Seyedmonir S (2014) Real coded genetic algorithm approach with random transfer vectors-based mutation for short-term hydro–thermal scheduling. IET Gener Transm Distrib 9(1):75–89
55. Akay B, Karaboga D (2012) Artificial bee colony algorithm for large-scale problems and engineering design optimization. J Intell Manuf 21(4):1001–1014
56. Ray T, Liew KM (2003) Society and civilization: an optimization algorithm based on the simulation of social behavior. IEEE Trans Evol Comput 7:386–396
57. Mezura-Montes E, Coello CAC Useful infeasible solutions in engineering optimization with evolutionary algorithms. In: Proceedings of the 4th Mexican international conference on advances in artificial intelligence, 2005, pp 652–662
58. Gandomi AH, Yang XS, Alavi AH (2013) Cuckoo search algorithm: a metaheuristic approach to solve structural optimization problems. Eng Comput 29(1):17–35