Soft Comput (2018) 22:8273–8288
https://doi.org/10.1007/s00500-017-2773-6

METHODOLOGIES AND APPLICATION

Multi-objective imperialistic competitive algorithm with multiple non-dominated sets for the solution of global optimization problems

Zhavat Sherinov[1] · Ahmet Ünveren[1]

Published online: 21 August 2017

Abstract In this paper, we propose a multi-objective imperialistic competitive algorithm (MOICA) for solving global multi-objective optimization problems. The MOICA is a modified and improved multi-objective version of the single-objective imperialistic competitive algorithm previously proposed by Atashpaz-Gargari and Lucas (IEEE Congr Evolut Comput 7:4661–4666. doi:10.1109/CEC.2007.4425083, 2007). The presented algorithm utilizes the metaphor of imperialism to solve optimization problems. Accordingly, the individuals in a population are referred to as countries, of which there are two types—colonies and imperialists. The MOICA incorporates competition between empires and their colonies for the solution of multi-objective problems. To this end, it employs several non-dominated solution sets, whereby each set is referred to as a local non-dominated solution (LNDS) set. All imperialists in an empire are considered non-dominated solutions, whereas all colonies are considered dominated solutions. In addition to LNDS sets, there is one global non-dominated solution (GNDS) set, which is created from the LNDS sets of all empires. There are two primary operators in the proposed algorithm, i.e., assimilation and revolution, which use the GNDS and LNDS sets, respectively. The significance of this study lies in a notable feature of the proposed algorithm, which is that no special parameter is used for diversity preservation. This enables the algorithm to prevent extra computation to maintain the spread of solutions. Simulations and experimental results on multi-objective benchmark problems show that the MOICA is more efficient compared to a few existing major multi-objective optimization algorithms because it produces better results for several test problems.

Keywords Multi-objective metaheuristics · Imperialistic competitive algorithm · Multiple non-dominated sets · Global optimization

1 Introduction

Several real-world problems must be solved by optimizing more than one objective. In a few cases, one objective must be minimized, while the other must be maximized (Sherinov et al. 2011). In this paper, we propose a multi-objective version of the imperialistic competitive algorithm (ICA) (Atashpaz-Gargari and Lucas 2007) for solving global multi-objective optimization problems based on imperialistic competition. The ICA is a global optimization strategy, in which the initial algorithm population consists of two types of countries, i.e., imperialists and colonies. Imperialistic competition is the most important part of the algorithm and causes the colonies to converge to the global minimum of the objective function. Moving the colonies toward their relevant imperialist—*assimilating*—and generating new countries in each empire using *revolution* are other important parts of the algorithm.

A large number of multi-objective optimization algorithms have been proposed. Among these are multi-objective evolutionary algorithms (MOEAs) (Deb 2001; Fonseca and Fleming 1993; Horn et al. 1994; Srinivas and Deb 1995; Zitzler and Thiele 1998). A priority of multi-objective optimization algorithms is to simultaneously find several Pareto-optimal solutions. Another priority is to optimize conflicting

Communicated by V. Loia.

✉ Ahmet Ünveren
ahmet.unveren@emu.edu.tr

1 Computer Engineering Department, Eastern Mediterranean University, Mağusa, Mersin 10, Turkish Republic of Northern Cyprus, Turkey

objectives when one must be minimized and the other must be maximized. Consequently, multi-objective optimization algorithms have gained popularity in the last two decades. Therefore, the aim of this study is to develop a multi-objective optimization algorithm based on imperialistic competition—specifically, the multi-objective imperialistic competitive algorithm (MOICA)—which uses a population of countries of the following two types: imperialists and colonies. In every empire, there is an imperialist, which is considered the local best for that empire. Accordingly, the MOICA generates a local non-dominated solution (LNDS) set for each empire. Then, it calculates the global non-dominated solution (GNDS) set of the LNDSs of each empire, which is the final set of non-dominated solutions.

In terms of search and competition, the ICA is similar to particle swarm optimization (PSO), which simulates the social behaviors of animals, such as bird flocking (Kennedy and Eberhart 1995). Additionally, the ICA has a local best in each empire (i.e., the imperialist country) and a global best, which is the strongest of the imperialists. By beginning with several empires, each with several colonies, competition can occur between empires. This competition leads to the development of powerful empires and the collapse of weaker empires (Atashpaz-Gargari and Lucas 2007). The ICA is one of the numerous algorithms used for solving optimization problems. Among these are search heuristics, such as genetic algorithms (GAs), which belong to the class of evolutionary algorithms. These algorithms generate solutions to optimization problems by imitating the process of natural evolution (Mitchell 1999). Another example is ant colony optimization (ACO), which is inspired by the behavior of ants foraging for food (Dorigo and Blum 2005). In contrast, simulated annealing is an example of a global optimization algorithm. It is a generic probabilistic metaheuristic that locates a good approximation to the global optimum of a given function in a large search space (Kirkpatrick et al. 1983).

Multiple applications of the ICA exist, particularly in engineering. In computer engineering, the ICA is applied to data clustering and image processing for solving problems such as skin color detection and template matching (Seyedmohsen and Abdullah 2014). For example, Duan et al. (2010) presented a template matching method based on a chaotic ICA that used a correlation function. They prevented the problem of falling into a local best solution by introducing chaotic behavior into the ICA, which improved its global convergence. Another example of the application of the ICA is the integrated product mix-outsourcing optimization problem (Nazari-Shirkouhi et al. 2010). Vedadi et al. (2015) applied the ICA in electrical engineering by presenting an ICA-based maximum power point-tracking algorithm to rapidly and precisely find the global maximum power point of a power-voltage string under partial shading conditions. Goudarzi et al. (2013) used the ICA as a heuristic technique for optimizing the location of capacitors in radial distribution systems. Another example of the application of the ICA is in geoscience, where it is used to locate the critical failure surface and compute the safety factor in slope stability analysis based on the limit equilibrium approach (Kashani et al. 2014). Jordehi (2016) proposed a solution to flexible AC transmission systems (FACTS) allocation problems so that low overload and voltage deviation values result from line outage contingencies and demand growth. In this study, thyristor-controlled phase shifting transformers and thyristor-controlled series compensators have been used as FACTS devices.

Variants of the ICA have been presented in the literature. Niknam et al. (2011) proposed an efficient hybrid algorithm based on the modified ICA (MICA) and k-means, referred to as the K-MICA, to optimally cluster n objects into k clusters. This approach was used to overcome local optima obstacles. The K-MICA was tested for robustness and compared favorably to several algorithms, including ACO, PSO, GA, TS, honey bee mating optimization, and k-means. Razmjooy et al. (2013) proposed a hybrid algorithm by combining the ICA and an artificial neural network to solve skin classification problems. The authors used a multilayer perceptron network to manage problem constraints and the ICA to search for high-quality and minimum-cost solutions. Ebrahimzadeh et al. (2012) proposed a novel hybrid intelligent method for recognizing the common types of control chart patterns. The proposed method included the following two primary modules: a clustering module and a classifier module. The authors used a combination of the MICA and the k-means algorithm in the former module to cluster input data. In addition, a mutation operator was introduced into the proposed algorithm by changing the assimilation process.

In this paper, a new multi-objective ICA (MOICA) is proposed. The MOICA uses the idea of imperialism by incorporating competition between empires. Every empire has a set of imperialists and a set of colonies. The primary idea in this algorithm is to have an LNDS for every empire. Therefore, all imperialists in an empire are considered to be non-dominated solutions, whereas all colonies are considered to be dominated solutions. Moreover, in addition to LNDS sets, there is one GNDS set, which is created from the LNDS sets of all empires. Two main operators of the proposed algorithm, i.e., assimilation and revolution, use the GNDS and LNDS sets during the assimilation and revolution of colonies, respectively. Another significant feature of the proposed algorithm is that no special parameter is used for diversity preservation, which enables algorithm to prevent extra computations to maintain the spread of solutions. The proposed algorithm with the assimilation and revolution operators produces good results that are comparable with the state-of-the-art algorithms used in this study.

The remainder of this paper is organized as follows: Section 2 presents a brief description of a single-objective ICA. In Sect. 3, the proposed MOICA is discussed in detail. Thereafter, in Sect. 4, experimental results and simulations are presented by comparing the proposed algorithm with other multi-objective optimization algorithms. Lastly, in Sect. 5, the conclusions of this study are presented.

2 ICA review

The primary idea of the ICA is the competition that occurs between empires because the aim of each empire is to possess more colonies. This competition—along with *assimilation*, or moving colonies toward their relevant imperialist, and *revolution*, or abrupt changes in sociopolitical characteristics—enables the algorithm to reach the global optimum of a cost function. During competition among empires, it is possible for a colony to become better than the imperialist of its empire. In this case, the ICA switches the positions of the imperialist and colony; thus, the colony becomes the imperialist and the former imperialist becomes a colony. The power of each empire is inversely proportional to its cost function. Therefore, the lower the cost of the empire, the more powerful it is (Atashpaz-Gargari and Lucas 2007). When an empire has exhausted its colonies, it becomes powerless and as a result, it collapses and terminates. Consequently, the number of empires gradually decreases until only one of the most powerful imperialist states remains. However, the ICA's termination criterion is reaching the user-specified number of iterations, regardless of state. Similarly, the algorithm will continue to the number of iterations specified by the user even if only one empire remains, because it is not ensured that the optimum solution has been found when only one empire remains.

Algorithm – ICA (Atashpaz-Gargari and Lucas 2007)

1. Select a few random points on a function and initialize empires.
2. Move colonies toward their relevant imperialist (i.e., assimilation).
3. Randomly replace a few colonies with newly generated colonies (i.e., revolution).
4. Compute the costs of an imperialist and all colonies.
5. If there is a colony in an empire that has a lower cost than that of the imperialist, then exchange their positions.
6. Compute the total cost of all empires relative to the power of the imperialist and its colonies.
7. Select the weakest colony, i.e., a colony with the highest cost, from the weakest empire, i.e., the empire with the highest cost, and give it to the empire that is the most likely to possess it, thereby engendering imperialistic competition.
8. Eliminate powerless empires.
9. If the termination condition is not satisfied, return to Step 2.

After random initialization of the population, the objective function is evaluated and the individuals in the population are assigned their cost values. The individuals of size N with the minimum costs are selected to be imperialists. The remaining individuals become colonies that are proportionally distributed among imperialists based on their costs.

2.1 Assimilation

Assimilation is the process of moving the colonies toward the imperialist within the same empire. This process is one of the most important parts of the ICA because it is related to the improvement in the colonies of a particular empire. Figure 1 describes the movement of a colony toward its imperialist in a randomly deviated direction to search for different points around the imperialist. As shown in the figure, the new position of the colony is x, which is a random variable with a uniform distribution. Thus, we have, $x \sim U(0, \beta \times d)$, where β is a number between 1 and 2 and d is the distance between the colony and imperialist (Atashpaz-Gargari and Lucas 2007). In addition, θ is a random variable with a uniform distribution, which is $\theta \sim U(-\gamma, \gamma)$, where γ is a parameter that adjusts the deviation from the original direction (Atashpaz-Gargari and Lucas 2007).

The mathematical formulation of ICA assimilation may be demonstrated as follows: Let

$$Col\_Pos = [p_1, p_2, \ldots, p_n] \tag{1}$$

be the vector containing the colony's position and let

$$Imp\_Pos = [p'_1, p'_2, \ldots, p'_n] \tag{2}$$

be the vector containing the imperialist's position, where n is the dimension of the optimization problem. Now, let d be the vector containing the elementwise difference of (1) and (2) as follows:

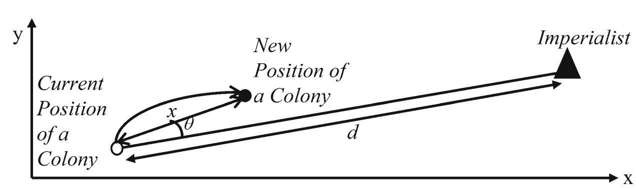

Fig. 1 Moving colonies toward their relevant imperialist in a randomly deviated direction

$$d = [p_1 - p_1', p_2 - p_2', \ldots, p_n - p_n']. \quad (3)$$

Therefore, the calculation of the new colony's position is

$$Col\_Pos\_New = Col\_Pos + \theta * \beta * r * d, \quad (4)$$

where r is a random variable with a uniform distribution between 0 and 1. The value of θ is selected to be in a range of ($-\pi/4, \pi/4$) radians, and β is selected to be approximately 2 (Atashpaz-Gargari and Lucas 2007).

2.2 Revolution

Revolution is the process of generating new countries in the empire (Atashpaz-Gargari and Lucas 2007). This occurs owing to an abrupt change in sociopolitical characteristics. While generating new countries, a few of the current countries (colonies) are randomly replaced by the newly created countries.

2.3 Imperialistic competition

Imperialistic competition occurs after the assimilation and revolution operations are applied to the colonies during each iteration of the algorithm, as shown in Fig. 1. Imperialistic competition starts with the computation of the total cost of all empires. The total cost of an empire can be expressed as follows (Atashpaz-Gargari and Lucas 2007):

$$\mathrm{TC}_k = \mathrm{IC}_k + \varepsilon * \mathrm{mean}(\mathrm{CE}_k), \quad (5)$$

where TC_k is the total cost of empire k, IC_k is the imperialist cost of empire k, CE_k is the cost of the colonies of empire k, and ε is a small value of approximately 0.1 to make the total cost of an empire depend mostly on the imperialist (a larger value for ε will make the total cost depend on the imperialist and the colonies of the empire).

Competition among the empires is realized by excluding the weakest empire from the competition and allowing other empires to compete for the weakest colony in the excluded weakest empire. The following mathematical formulation describes the possession probabilities of the competing empires for the weakest colony (Atashpaz-Gargari and Lucas 2007):

$$p_k = \left| \frac{\mathrm{NTC}_k}{\sum_{i=1}^{N} \mathrm{NTC}_i} \right|, \quad (6)$$

where p_k is the possession probability of empire k, N is the number of imperialists, and NTC_k is normalized total cost, which is computed as

$$\mathrm{NTC}_k = \mathrm{TC}_k + \max(\mathrm{TC}_i). \quad (7)$$

The final step in the competition between imperialists is to obtain a vector containing the differences between possession probabilities and the uniformly distributed random values between (0, 1) as follows:

$$D = [p_1 - r_1, p_2 - r_2, \ldots, p_N - r_N] \quad (8)$$

where N is the number of imperialists. The possessor of the weakest colony in the weakest empire is the one whose corresponding index in vector D contains the maximum value.

3 Proposed MOICA

3.1 Algorithm overview

The proposed MOICA implements the idea of imperialism by incorporating competition among empires. The primary concept of the MOICA is that there are several non-dominated solution sets, i.e., imperialists, per empire and one GNDS set, which contains the best imperialists among all empires. All empires attempt to possess other empires' colonies based on their power. Therefore, all empires have the opportunity to assume control of one or more colonies of the weakest empire. In an iteration of the algorithm, the colonies of each empire make changes with respect to their positions by changing their cost values. As previously mentioned, a colony, $\boldsymbol{C}$, in an empire may become better than a few of the current set of imperialists. In such a case, the new colony, $\boldsymbol{C}$, with better cost becomes a member of the empire's imperialists, that is, a member of the set of non-dominated solutions. Thus, the previous imperialist, $\boldsymbol{I}$, which is dominated by $\boldsymbol{C}$, becomes a colony.

The MOICA has an important yet simple feature in its implementation. Specifically, it has several non-dominated solution sets, which makes it different from numerous other multi-objective optimization algorithms. Initially, there are $\boldsymbol{N}$ empires. Therefore, every empire possesses Pareto-optimal solutions, or LNDSs. Therefore, the total number of LNDSs will initially be $\boldsymbol{N}$. Moreover, there is a set of GNDSs, which is obtained from the $\boldsymbol{N}$ LNDSs. Because the set of LNDSs for each empire is updated during iterations, the GNDS is updated accordingly. This implies that the algorithm has one GNDS throughout its implementation, whereas the number of LNDSs gradually decreases as empires collapse during competition. Figure 2 illustrates an example of three empires (E_1, E_2, and E_3) with their colonies and LNDS sets, i.e., imperialists, which are set in bold.

Imperialists that are taken into an area in Fig. 2 are the best imperialists among all empires that form the GNDS. There is a possibility that none of the imperialists will be included in the GNDS of an empire. An example of this scenario is E_1 in Fig. 2. Therefore, the use of the GNDS in this algo-

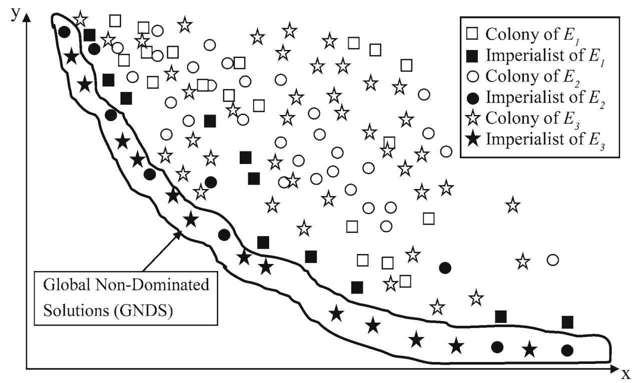

Fig. 2 GNDS and LNDS sets of three empires

rithm is considerably important because the colonies of all empires are assimilated toward randomly selected imperialists from the GNDS, which enables the algorithm to prevent local optima. If we consider only one empire in Fig. 2, for example, E_2, it is apparent that the circles in bold form the non-dominated solution set, E_2. The assimilation and revolution operations are detailed in the following sections.

Non-dominance in the proposed algorithm is calculated based on fronts; solutions that are assigned a value of 1 belong to the first front, while solutions with a front value of 2 are assigned to the second front, and so on. As a result, the LNDS and GNDS sets contain only solutions that belong to the first front. Another significant feature of the proposed algorithm is that no special parameter is used for diversity preservation, which enables the algorithm to prevent extra computation to maintain the spread of solutions. Even though a share parameter is not used in the MOICA, the solution spread in the results obtained from our simulations and experiments was excellent owing to the assimilation technique used in the algorithm. As described in the previous section, all colonies of an empire move toward one imperialist available in the empire. However, in the proposed algorithm, the colonies of an empire move toward one of the imperialists, $\boldsymbol{I}$, in the GNDS set. The imperialist, $\boldsymbol{I}$, toward which the colonies move is randomly selected in each iteration from the GNDS set. Therefore, the prevention of a share parameter is derived from the multi-objective nature of the algorithm, in which every solution in a non-dominated solution set is valid, so that there is no single solution. For clarity, we first describe the proposed algorithm. Then, each part of the algorithm is detailed.

Algorithm – Primary procedure of the MOICA

1. Begin
2. Initialization
    a. Initialize problem parameters, such as the objective function name, number of variables, and the lower and upper bounds of decision variables.
    b. Initialize algorithm parameters, such as population size, number of initial empires, number of iterations, and other coefficients used in the assimilation and revolution operations.
3. Evaluate objective functions and assign cost values to each country.
4. Apply non-domination sorting [15].
5. Create initial empires.
6. **For each** iteration i **do**:
    a. **For each** empire j **do**:
        i. Obtain the GNDS.
        ii. Apply *assimilation*: Move colonies toward one randomly selected imperialist in the GNDS set and apply *economic changes* with probability p_e.
        iii. Apply *revolution*: Generate new countries from the LNDS set according to probability p_r and revolution rate α.
        iv. Evaluate objective functions and assign cost values to all colonies.
        v. Update the LNDS for empire j.
        vi. Calculate the total power of empire j.
    b. **End for**
7. Unite similar empires.
8. Apply imperialistic competition and terminate powerless empires.
9. **End for**
10. Display results.

3.2 Non-domination sorting

Various methods have been proposed in the literature for determining non-dominance. In these methods, each solution in a search space is assigned a rank value, which indicates whether the solution is dominated by other solutions. In most cases, the lower the rank value, the less the solution is dominated by other solutions. For example, a rank value of one indicates that the solution is non-dominated. Another approach of ascertaining non-dominance is to not assign a rank value to solutions and divide them into fronts instead (Deb et al. 2002), which is the approach that we use in this study. Figure 3 illustrates non-dominated solutions with fronts for the minimization problem.

A front with a value of one contains non-dominated solutions, whereas the front value of two is the set of solutions dominated by the solutions from the first front only. Solutions with a front value of three are dominated by the solutions from the previous fronts. Therefore, in the proposed algorithm, every empire has its own LNDS. This LNDS is intended to include the imperialists of the empire; there is no single imperialist in the empire. This implies that all other solutions have

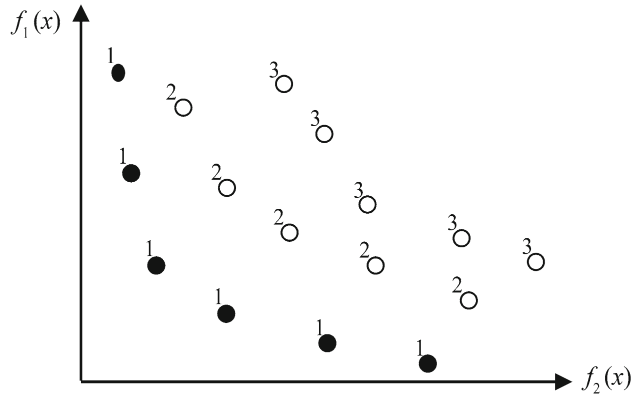

Fig. 3 Non-dominance using fronts

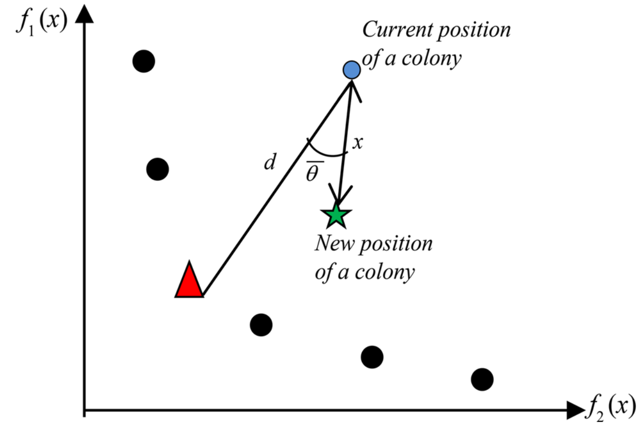

Fig. 4 Assimilation of a colony toward a randomly selected imperialist from the GNDS set

front values larger than one, such that dominated solutions are considered the colonies of the empire.

3.3 Assimilation

Assimilation, the movement of colonies toward imperialists, is implemented similarly, as explained in Sect. 2. However, there are several imperialists in the GNDS set. Thus, one imperialist should be selected to serve as a target for the movement of colonies. The use of the GNDS in assimilation instead of LNDS sets enables the algorithm to escape local minima faster. The selection of the target imperialist is randomly performed for each empire in each iteration of the algorithm.

Figure 4 illustrates the assimilation procedure implemented in this algorithm. In the figure, the black circles and red triangle indicate non-dominated solutions—the GNDS set— that are imperialists of the whole population. The red triangle is the randomly selected target imperialist toward which the colonies are moving. For simplicity, only one moving colony is shown in the figure and is indicated by a blue circle. Parameters such as θ, d, and x are explained in Sect. 2; however, the values used for a few parameters are different, which will be discussed later. Owing to randomized selection of the target imperialists and deviation θ, the diversity in the algorithm is preserved. In Fig. 4, the angle is denoted by θ, because the deviation θ is used in the decision space, which may not be the same as in the objective space. Therefore, even if deviations in the decision and objective spaces differ, deviation still exists in the objective space, which is denoted by $\bar{\theta}$.

To improve the local search of the proposed algorithm, another new operation is added immediately after assimilation. This operation is the influence of *economic changes* on the empire, which has a probability of being engaged, as described in the pseudocode below. The higher the value of p_e, the lower the probability of performing the operation. In most cases, a value of 0.9 is used, to incite a few economic changes. *UpperBound* and *LowerBound* are vectors that indicate the decision space of the decision variables for the given objective function. *rand()* is a uniformly generated random value between (0, 1). The variables and parameters *Col_Pos_New*,*Col_Pos*, θ, β and r are the same as in Sect. 2. However, d is different in this operation because it contains an elementwise difference of a colony and a randomly selected imperialist from the GNDS.

Procedure: Assimilation with local search: economic changes

1. Randomly select an imperialist I_G from the GNDS.
2. **for each** colony in empire i **do**
3. set d to the elementwise difference of a colony and I_G
4. *Col_Pos_New* = *Col_Pos* + $\theta * \beta * r * d$
5. **end for**
6. **if** rand() $> p_e$ **do**
7. R = UpperBound – LowerBound;
8. **for each** decision variable i in R **do**
9. $w(i)$ = (abs(*UpperBound*(i))*rand())$^{rand()/R(i)}$ – (abs(*LowerBound*(i))*rand())$^{rand()/R(i)}$;
10. **end for**
11. *ColoniesOfEmpire* = *ColoniesOfEmpire* .* w;
12. **end if**

3.4 Revolution

The revolution operation in the proposed algorithm is completely different from the one in ICA because there is no random generation of new colonies. The new revolution operation has two parts, which are performed based on probability p_r. The first part is the generation of a new colony by the random selection of elements from two randomly selected imperialists in the LNDS set of one empire. If there is only one imperialist in the LNDS set, then one more individual is randomly generated. For the second part of the revolution process, a few imperialists are modified and replaced by randomly chosen colonies.

Procedure: Revolution

1. **if** *rand()* > p_r
2. **for each** colony in empire i **do**
   a. Select two imperialists $\boldsymbol{I}_1$ and $\boldsymbol{I}_2$ from LNDS (if the set contains one imperialist only, then generate one more randomly)
   b. Generate two random points P_1 and P_2 between 1 and the length of individuals
   c. Split every imperialist into three blocks using points P_1 and P_2
   d. Generate new colony $\boldsymbol{C}$ by combining the first and third blocks from $\boldsymbol{I}_1$ and the second block from $\boldsymbol{I}_2$
   e. Replace colony i in an empire with $\boldsymbol{C}$
3. **end for**
4. **else**
5. **for** $i = 1$ to *RevolutionRate* * *NumberOfColoniesInEmpire*
   a. Select one imperialist $\boldsymbol{I}_i$ from LNDS randomly
   b. Update $\boldsymbol{I}_i$ by adding to its every element a random value between (0.001, 0.09) or (−0.09, −0.001)
6. **end for**
7. Update randomly selected colonies by newly generated ones
8. **end if**

The GNDS is used to select imperialists for updating colonies during assimilation. On the other hand, imperialists from the LNDS of the same empire are used in the revolution process. Therefore, both assimilation and revolution of colonies enable the algorithm to escape local minima and reach global optimal solutions.

3.5 Possessing an empire

Every empire is possessed by the set of imperialists, which is the non-dominated set of solutions within the empire itself and is defined as the LNDS in this algorithm. However, in terms of possession of the empire, it is possible that all individuals of an empire will be in the LNDS, and thus, there are no dominated solutions within an empire. Consequently, assimilation and revolution will not be applicable. Therefore, one more parameter Ø was added to this algorithm. It indicates the maximum percentage of imperialists that an empire can have. Consequently, when obtaining the LNDS of an empire, the determination of whether the percentage of imperialists exceeds Ø is made. If so, then the best maximum imperialists allowed are retained; the others are moved to the set of colonies. The total power of an empire is equal to the number of non-dominated solutions in the empire's population. Although an empire with a lower number of non-dominated solutions may contain better solutions than one with more non-dominated solutions, the total power is still equal to the cardinality measure, regardless of dominance.

3.6 Uniting similar empires

The MOICA uses different approaches to unite similar empires in comparison with ICA; in ICA, empires are united when an empire's imperialist is very close to another's imperialist. This is achieved by calculating the distance between the positions of two imperialists and comparing this calculated distance with the distance threshold parameter, which is originally set to 0.02. The distance threshold used here is not for diversity preservation; it is only used for measuring how close two empires are. If the distance is less than or equal to the specified threshold, then the empires are united.

In the proposed algorithm, this approach cannot be applied because there are several imperialists in an empire. Thus, all imperialists must be considered when comparing empires' similarity. Consequently, the empire similarity comparison is implemented using the generational distance metric (Van Veldhuizen and Lamont 1998), which enables the calculation of the generational distance between two or more sets of non-dominated solutions. The generational distance GD is defined as:

$$\mathrm{GD} = \frac{1}{|S^*|} \sum\nolimits_{r \in S^*} \min \left\{ d_{rx} | x \in S_j \right\}, \tag{9}$$

where S^* is a reference solution set for the evaluation of the solution set S_j and d_{xr} is the distance between the current solution x and reference solution r, given as

$$d_{rx} = \sqrt{(f_1(r) - f_1(x))^2 + (f_2(r) - f_2(x))^2 + (f_3(r) - f_3(x))^2 + \ldots + (f_k(r) - f_k(x))^2} \tag{10}$$

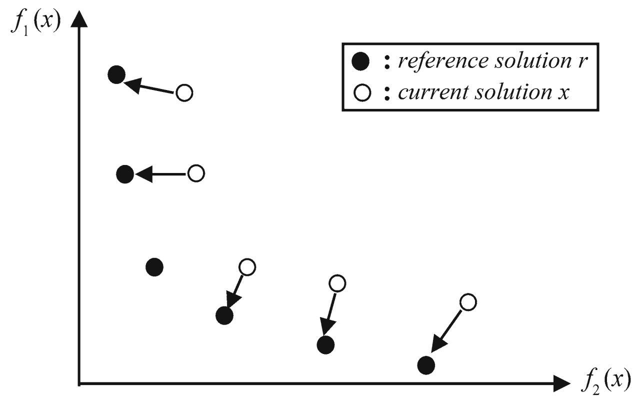

Fig. 5 Generational distance for uniting empires

where k is the number of objective functions to be optimized. Figure 5 illustrates an example computation of generational distance for two objective functions.

3.7 Imperialistic competition

Imperialistic competition plays an important role in this algorithm, gradually decreasing the number of weak empires and increasing the number of strong empires. The weakest empire in the proposed algorithm is the one with the smallest number of non-dominated individuals, whereas the strongest empire is the one with the largest number of non-dominated individuals. Imperialistic competition is constructed so that the stronger an empire is, the more chances it has of obtaining control of a weak colony in a weak empire. Consequently, it obtains possession of it. Weak empires slowly lose their colonies during this competition and are soon terminated because of their powerlessness, which means that these empires are left with no countries.

Procedure: Imperialistic Competition

1. Construct a vector of the total powers **P** for all empires.
2. Select the weakest empire **E** with the lowest total power.
3. Construct a vector of random values **R** $\sim U(0, 1)$ of size **P.**
4. Calculate **D** = **R** - **P** for each empire.
5. The empire with the maximum value in **D** will possess the randomly selected colony in empire **E.**
6. Terminate **E** if it has no colonies.

3.8 Computational complexity of MOICA

The time complexity for implementing non-domination sorting in the MOICA is the same as the time complexity for non-domination sorting in NSGA-II, i.e., $O(M(2\,N)^2)$, where M is the number of objectives and N is the number of solutions, i.e., the population size. Considering the time complexities of assimilation and revolution operations, in the worst case, it is possible for $N - 1$ colonies to be assimilated/revolt if there is only one dominating imperialist. Therefore, in every iteration, for both assimilation and revolution, the time complexity is $O(N)$. Another consideration is the time complexity for uniting similar empires, which is $O(K^2)$ in every iteration, where K is the number of empires in the population. Consequently, the overall time complexity of the MOICA is $O(M(2\,N)^2 + K^2)$. Comparing the time complexities of the MOICA and NSGA-II, we conclude that they are almost the same, since K^2 is related to the number of empires, which is usually very low in comparison with population size N, and could even be omitted.

4 Experimental results

This section details the experiments and simulations conducted in this study. To obtain the experimental results and verify the effectiveness of the proposed algorithm, several bi-objective and tri-objective optimization problems were selected from the literature as test problems. ZDT1, ZDT2, ZDT3, ZDT4, and ZDT6 were obtained from Zitzler et al. (2000), in addition to test problems from Kursawe (1990), Fonseca and Fleming (1998), and Schaffer (1987). Moreover, ten unconstrained test functions were employed from the Congress on Evolutionary Competition (CEC) 2009 Special Session and Competition (Zhang et al. 2009)—UF1, UF2, UF3, UF4, UF5, UF6, UF7, UF8, UF9, and UF10.

Table 1 details all the unconstrained test problems used in this study, except the CEC 2009 test functions, which can be found in (Zhang et al. 2009). The performance metrics used to evaluate our results with the Pareto-optimal solutions are hypervolume (HV) (Zitzler and Thiele 1998), epsilon indicator (EI) (Zitzler et al. 2003), and inverted generational distance (IGD). The IGD metric used in this study is the jMetal version.

In addition, this section compares the results of the proposed algorithm with those of state-of-the-art multi-objective optimization algorithms.

4.1 Discussion

All experimental results were obtained by executing each algorithm ten times. The maximum number of function evaluations was set to 25,000. For a few test functions, it was set to 5000 to verify the performance of the algorithms with higher and lower numbers of function evaluations. The population size was set to 100 for all algorithms. The dimension of the individuals in the population was set to 30 for all test functions. The tables below describe the average HV and EI, which were obtained from several executions of the given

Table 1 Unconstrained test problems used in this study

Problem	Objective functions	Variable bounds	n
Fonseca	$f_1(x) = 1 - e^{-\sum_{i=1}^{n}\left(x_i - \frac{1}{\sqrt{n}}\right)^2}$	$-4 \le x_i \le 4$	3
	$f_2(x) = 1 - e^{-\sum_{i=1}^{n}\left(x_i + \frac{1}{\sqrt{n}}\right)^2}$		
Kursawe	$f_1(x) = \sum_{i=1}^{n-1}\left(-10e^{\left(-0.2*\sqrt{x_i^2 + x_{i+1}^2}\right)}\right)$	$-5 \le x_i \le 5$	3
	$f_2(x) = \sum_{i=1}^{n}\left(\lvert x_i\rvert^a + 5sinx_i^b\right); a = 0.8; b = 3$		
Schaffer	$f_1(x) = x^2$	$-10 \le x \le 10^5$	1
	$f_2(x) = (x-2)^2$		
ZDT1	$f_1(x) = x_1$	$0 \le x_i \le 1$	30
	$f_2(x) = g(x)\left[1 - \sqrt{x_1/g(x)}\right]$		
	$g(x) = 1 + 9\left(\sum_{i=2}^{n} x_i\right)/(n-1)$		
ZDT2	$f_1(x) = x_1$	$0 \le x_i \le 1$	30
	$f_2(x) = g(x)\left[1 - (x_1/g(x))^2\right]$		
	$g(x) = 1 + 9\left(\sum_{i=2}^{n} x_i\right)/(n-1)$		
ZDT3	$f_1(x) = x_1$	$0 \le x_i \le 1$	30
	$f_2(x) = g(x)\left[1 - \sqrt{\frac{x_i}{g(x)}} - \frac{x_1}{g(x)}\sin(10\pi x_1)\right]$		
	$g(x) = 1 + 9\left(\sum_{i=2}^{n} x_i\right)/(n-1)$		
ZDT4	$f_1(x) = x_1$	$0 \le x_1 \le 1$	30
	$f_2(x) = g(x)\left[1 - (x_1/g(x))^2\right]$	$-5 \le x_i \le 5$	
	$g(x) = 1 + 10(n-1) + \sum_{i=2}^{n}\left[x_i^2 - 10\cos(4\pi x_i)\right]$	$i = 2,\ldots,n$	
ZDT6	$f_1(x) = 1 - e^{-4x_1} sin^6(6\pi x_1)$	$0 \le x_i \le 1$	30
	$f_2(x) = g(x)\left[1 - (f_1(x)/g(x))^2\right]$		
	$g(x) = 1 + 9\left[\frac{\sum_{i=2}^{n} x_i}{n-1}\right]^{0.25}$		

algorithm. The IGD was obtained from the average value from several executions of the algorithms.

The proposed algorithm used the following parameters. The initial number of empires was set to 8. From several tests, it was evident that its performance was poor with far fewer or far more than 8 initial empires. The parameter θ had a random value between (0, 1) and β had a random value between (0, 5). The parameter for the percentage of imperialists Ø was set to 0.3, so that at most 30% of the empire's population was considered imperialist. Thus, 70% of the space was left for colonies in an empire, so more assimilations and revolutions were performed. The revolution rate α was set to 0.3, and the parameter used in the revolution process p_r was set to 0.5. The optimal value for the parameter for applying economic changes p_e may differ across test functions. For example, for UF9, the result was best when p_e was set to around 0.2; nonetheless, in most cases, it was found to be between 0.8 and 1 based on a trial-and-error approach. The values for the above parameters were chosen as the best suitable values for the proposed algorithm after the conduction of a number of experiments. Therefore, the parameters for the MOICA were tuned using a non-iterative algorithmic approach (Eiben and Smit 2011), such that the parameters were generated during initialization and were then tested.

Table 2 Hypervolume results for unconstrained test problems with 25,000 function evaluations

Function	Algorithm			
	MOICA	NSGA-II	SPEA2	OMOPSO
Fonseca	0.99441	0.99441	0.99447	**0.99453**
Kursawe	**1.00000**	**1.00000**	**1.00000**	**1.00000**
Schaffer	**0.97764**	0.81099	0.90619	0.97748
ZDT1	0.99242	0.99713	0.99699	**0.99725**
ZDT2	0.98534	0.99431	0.99392	**0.99444**
ZDT3	0.99824	0.99833	**0.99835**	0.99831
ZDT4	**0.98440**	0.72624	0.57710	0.02393
ZDT6	**0.97120**	0.93140	0.90948	0.97105

The bold values show the best results of the given methods in the indicated problems

The first three test problems addressed in this section are Fonseca, Kursawe, and Schaffer. Then, the ZDT set of problems is discussed and the results of the set of unconstrained problems from CEC 2009 are described. Values in bold are the best results obtained. All algorithms performed well in terms of convergence and divergence for each of the problems below. The cardinality measure, i.e., the number of non-dominated solutions, is important, as having more candidate solutions means more chances for good convergence.

Table 3 Epsilon indicator results for unconstrained test problems with 25,000 function evaluations

Function	Algorithm			
	MOICA	NSGA-II	SPEA2	OMOPSO
Fonseca	1.00470	1.00560	1.00520	**1.00310**
Kursawe	**1.04330**	1.04890	1.04890	1.04620
Schaffer	1.01020	1.01030	1.07750	**1.01010**
ZDT1	1.03240	1.03330	1.03580	**1.01360**
ZDT2	1.00350	1.00348	1.00329	**1.00250**
ZDT3	**1.00000**	0.99970	0.99950	0.99910
ZDT4	**1.03260**	3.43550	5.38910	22.01040
ZDT6	1.01610	1.31910	1.47070	**1.01590**

The bold values show the best results of the given methods in the indicated problems

One of the main features that distinguish the MOICA is the cardinality measure, which is very good for most problems. Supplementary materials related to the proposed method are available in Online Resource 1.

The five real-valued ZDT problems are presented in Table 1 (ZDT5, the omitted problem, is binary-encoded). Incidentally, since it is binary-encoded, ZDT5 has often been omitted from analysis elsewhere in the EA literature.

Tables 2, 3, and 4 contain the HV, EI, and IGD results of the MOICA, NSGA-II, SPEA2, and OMOPSO for the unconstrained test problems in Table 1. Table 4 also includes the MOEA/D-AWA algorithm (Qi et al. 2014). On average, the results for HV and EI are similar for all algorithms. However, the MOICA performs considerably better in terms of IGD.

As stated above, one of the features of the MOICA is its ability to produce many candidate solutions. The Schaffer test problem is an example that illustrates the cardinality measure of the MOICA. Figure 6 illustrates the Pareto found by all algorithms for the Schaffer test problem. Although the HV and EI results are good for all algorithms, as shown in

Table 4 IGD results for unconstrained test problems with 25,000 function evaluations

Function	Algorithm						
	MOICA	NSGA-II	SPEA2	OMOPSO	MOEA/D-AWA	Harmony NSGA-II	Harmony MOEAD
Fonseca	**1.1805E−4**	3.2267E−4	2.3427E−4	2.1033E−4	–	–	–
Kursawe	4.4896E−4	1.7598E−4	**1.3464E−4**	1.6165E−4	–	–	–
Schaffer	**2.3575E−5**	0.0373	0.0215	3.3629E−4	–	–	–
ZDT1	**2.5732E−5**	1.8641E−4	1.5222E−4	1.3782E−4	4.470E−3	8.03E−04	1.86E−03
ZDT2	**3.5707E−5**	1.9656E−4	1.7261E−4	1.4183E−4	4.482E−3	1.12E−03	3.01E−03
ZDT3	**7.4842E−5**	2.6488E−4	2.3718E−4	2.1859E−4	6.703E−3	5.01E−04	1.19E−03
ZDT4	**3.8724E−5**	0.0849	0.1365	1.1501	4.238E−3	8.33E−02	1.64E−04
ZDT6	**1.6200E−5**	0.0137	0.0219	1.2514E−4	4.323E−3	2.11E−04	1.91E−04

The bold values show the best results of the given methods in the indicated problems

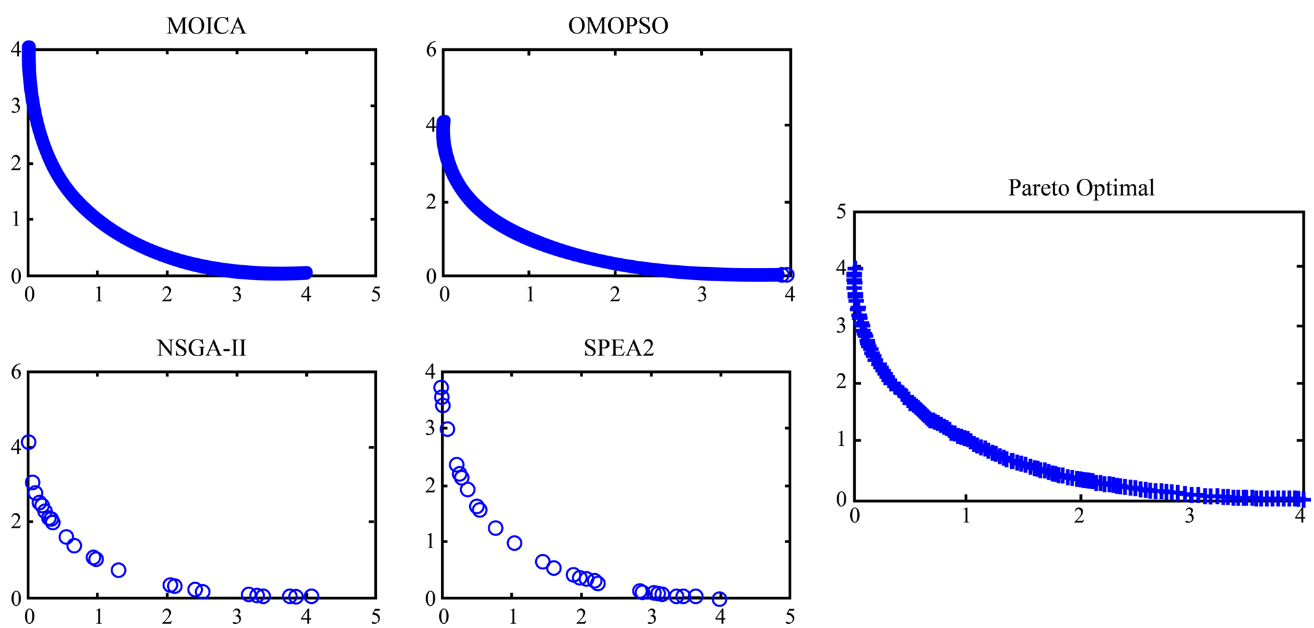

Fig. 6 Cardinality measure of MOICA, OMOPSO, NSGA-II, and SPEA2 on Schaffer

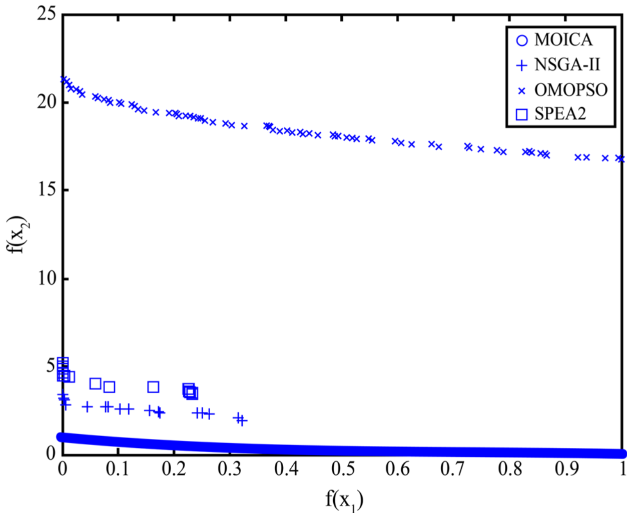

Fig. 7 Non-dominated MOICA, OMOPSO, NSGA-II, and SPEA2 solutions on ZDT4

Tables 2 and 3, respectively, Fig. 6 shows that the MOICA and OMOPSO have considerably better cardinality measures than NSGA-II and SPEA2.

For ZDT1, ZDT2, and ZDT3, all algorithms performed equally well. However, with respect to the ZDT4 test problem, the MOICA performed considerably better than all other algorithms in this study. In the ZDT4 test problem, the MOICA demonstrated its power in terms of convergence and divergence. It was successful in this test problem and others because of the method by which it searches the available space. It does so by setting many different empires in the beginning of the algorithm for which LNDS sets are positioned in different parts of the search space. This enables the algorithm to search the whole search space and to consequently obtain good convergence and divergence. Figure 7 illustrates the Pareto found by four algorithms for the ZDT4 test problem.

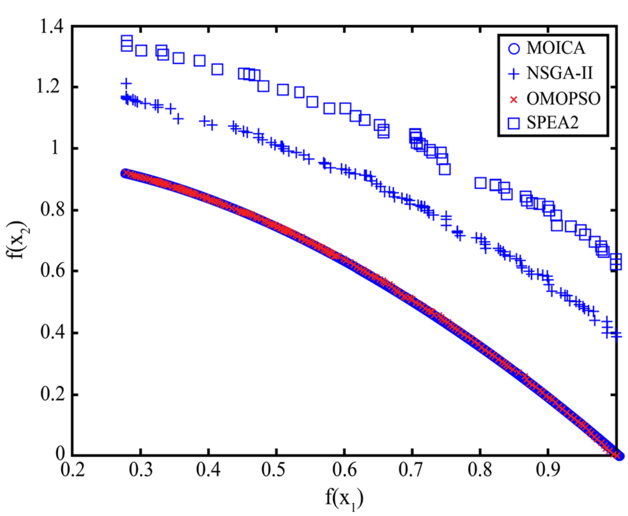

Fig. 8 Non-dominated MOICA, OMOPSO, NSGA-II, and SPEA2 solutions on ZDT6

Figure 7 demonstrates how the spread of solutions, convergence, and divergence are effectively preserved in the MOICA compared to the other algorithms. Figure 8 illustrates the ZDT6 test problem, which is another good example for illustrating the performance of the MOICA compared to the other algorithms. In ZDT6, both the MOICA and OMOPSO performed well compared to NSGA-II and SPEA2; however, NSGA-II performed better than SPEA2.

Figures illustrating the Paretos of algorithms for other test problems are not provided here because they have nearly the same Paretos as those determined herein.

Tables 5, 6, and 7 contain hypervolume, epsilon indicator, and IGD results for the UF1-UF10 unconstrained test

Table 5 Hypervolume results for the CEC 2009 unconstrained test problems with 25,000 function evaluations

Function	Algorithm			
	MOICA	NSGA-II	SPEA2	OMOPSO
UF1	0.98701	0.97802	0.98667	**0.98955**
UF2	**0.99671**	0.98934	0.98897	0.99279
UF3	0.92884	0.94478	0.98728	**0.99454**
UF4	**0.98849**	0.98838	0.98811	0.98728
UF5	**0.93230**	0.91786	0.90060	0.81056
UF6	0.93785	**0.94459**	0.94282	0.91297
UF7	0.97662	0.96890	0.95315	**0.98730**
UF8	**0.99348**	0.99280	0.99220	0.98945
UF9	**0.98309**	0.97468	0.95914	0.97845
UF10	**0.99334**	0.92076	0.92967	0.72864

The bold values show the best results of the given methods in the indicated problems

Table 6 Epsilon indicator results for the CEC 2009 unconstrained test problems with 25,000 function evaluations

Function	Algorithm			
	MOICA	NSGA-II	SPEA2	OMOPSO
UF1	**1.07890**	1.11220	1.06800	2.03310
UF2	1.14590	**1.12880**	1.13060	1.22760
UF3	1.70320	1.83810	**1.07410**	1.50760
UF4	**1.06600**	1.08930	1.07390	1.07410
UF5	**1.55180**	1.81000	1.86610	6.19530
UF6	1.62380	1.56080	**1.42700**	2.22700
UF7	1.07390	**1.03760**	1.05000	1.57840
UF8	**2.04970**	4.01980	2.91510	7.51780
UF9	4.19050	6.75860	**3.61460**	23.76640
UF10	**1.23930**	3.25450	2.96740	7.65290

The bold values show the best results of the given methods in the indicated problems

Table 7 IGD results for the CEC 2009 unconstrained test problems with 25,000 function evaluations

Function	Algorithm						
	MOICA	NSGA-II	SPEA2	OMOPSO	DMCMOABC	Harmony NSGA-II	Harmony MOEAD
UF1	**0.0035**	0.0047	0.0042	0.0041	0.0053	0.0037	0.0026
UF2	**0.0018**	0.0020	0.0024	0.0021	0.0050	0.0345	**0.0018**
UF3	0.0103	0.0084	0.0072	0.0072	0.0544	0.0085	**0.0067**
UF4	**0.0018**	**0.0018**	0.0019	0.0022	0.0254	0.0033	0.0021
UF5	0.1268	0.1117	0.1155	0.3579	0.0527	**0.0457**	0.0488
UF6	0.0127	0.0102	0.0119	0.0178	0.0393	**0.0089**	0.0092
UF7	0.0075	0.0068	0.0098	**0.0036**	0.0065	0.0113	0.0118
UF8	**0.0026**	0.0029	0.0027	0.0037	0.0665	0.0028	0.0054
UF9	0.0035	0.0035	**0.0030**	0.0056	0.0368	0.0044	0.0060
UF10	0.0037	0.0063	0.0046	0.0266	0.1119	**0.0036**	0.0059

The bold values show the best results of the given methods in the indicated problems

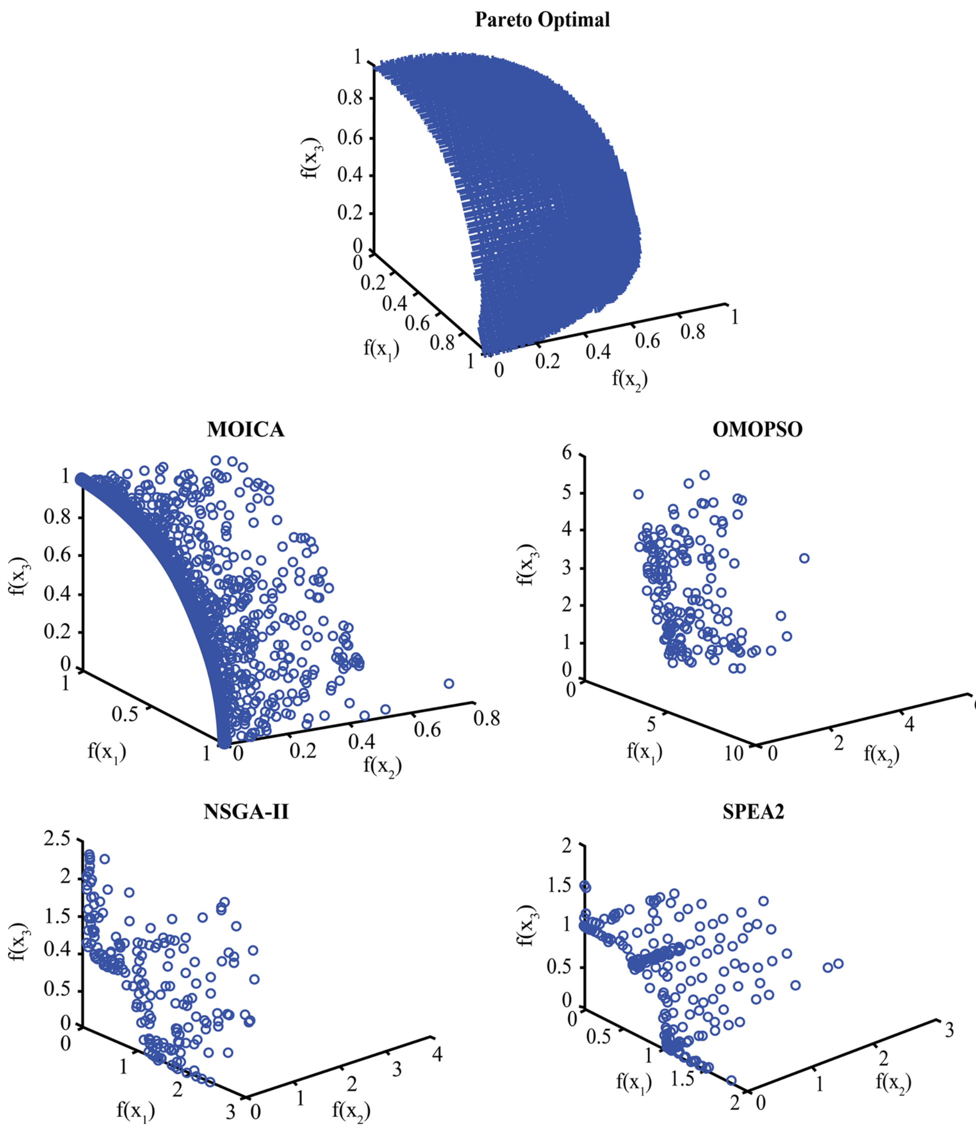

Fig. 9 Non-dominated MOICA, OMOPSO, NSGA-II, and SPEA2 solutions on UF10

Table 8 Hypervolume results for the CEC 2009 unconstrained test problems with 5000 function evaluations

Function	Algorithm			
	MOICA	NSGA-II	SPEA2	OMOPSO
UF1	0.97192	0.97651	**0.98519**	0.97075
UF2	0.98314	0.98818	0.98617	**0.98854**
UF3	0.91334	0.90554	0.89972	**0.97219**
UF4	**0.98567**	0.98377	0.98167	0.98521
UF5	**0.86472**	0.83347	0.80558	0.74621
UF6	**0.88661**	0.88456	0.88426	0.86270
UF7	0.97156	0.94940	0.93761	**0.97825**

The bold values show the best results of the given methods in the indicated problems

Table 9 Epsilon indicator results for the CEC 2009 unconstrained test problems with 5000 function evaluations

Function	Algorithm			
	MOICA	NSGA-II	SPEA2	OMOPSO
UF1	**1.22630**	1.57160	1.60980	1.45710
UF2	1.51470	1.31100	**1.27830**	1.35280
UF3	2.82790	2.37100	2.50600	**1.61580**
UF4	1.12490	1.14260	1.13340	**1.11650**
UF5	**2.87590**	4.36320	2.98440	5.24710
UF6	2.45680	**2.32760**	3.01710	5.07890
UF7	**1.12350**	1.23610	1.44560	1.43810

The bold values show the best results of the given methods in the indicated problems

Table 10 *IGD* results for the CEC 2009 unconstrained test problems with 5000 function evaluations

Function	Algorithm			
	MOICA	NSGA-II	SPEA2	OMOPSO
UF1	0.0054	**0.0045**	0.0051	0.0064
UF2	0.0033	0.0032	0.0033	**0.0030**
UF3	0.0152	0.0156	0.0153	0.0101
UF4	**0.0027**	0.0031	0.0033	**0.0027**
UF5	**0.2167**	0.3346	0.3409	0.5060
UF6	**0.0229**	0.0264	0.0243	0.0357
UF7	**0.0061**	0.0089	0.0117	0.0066

The bold values show the best results of the given methods in the indicated problems

problems from CEC 2009 with 25 function evaluations, for which the MOICA, on average, again produces reasonably good results.

The MOICA produced competitive results on the test functions from CEC 2009 compared to the other algorithms. Harmony NSGA-II and Harmony MOEAD (Doush and Bataineh 2015) also performed well, whereas DMC-MOABC (Xiang and Zhou 2015) performed the worst. Figure 9 presents the MOICA's results alongside those of the other algorithms, as well as the Pareto-optimal for the UF10 unconstrained test function. Tables 5, 6, and 7 illustrate that the MOICA performs better than the other algorithms with respect to UF10. In addition, it is clear from Fig. 4 that the MOICA is within the objective space of the Pareto-optimal, unlike the other algorithms.

Tables 8, 9, and 10 show the results for the UF1-UF7 test functions with a maximum of 5000 function evaluations. The MOICA's average performance is either similar or better than the performance of the other algorithms, even for such few function evaluations. This result likewise proves that the MOICA quickly converges to global optimal solutions.

Table 11 presents the MOICA's ranking compared with the algorithms used in the CEC 2009 competition for unconstrained functions. The ranking is based on the average IGD metric.

4.2 Friedman aligned ranks test

To check the statistical similarity of our results to those of other algorithms and determine the MOICA's rank among its competitors, we implemented the Friedman aligned ranks test for all average IGD scores achieved by the 13 algorithms in the CEC 2009 MOO contest along with the proposed MOICA. Table 12 shows the average rank values for all algorithms and the p value of the test. The subscripted numbers for the best scores indicate the order of the corresponding algorithms. The average rank value of the MOICA is the smallest, which indicates that the MOICA is the best-performing algorithm among the 13 analyzed. Meanwhile, the p value is very close to zero, indicating that there is significant statistical difference among the results of all algorithms, such that the MOICA is statistically different from its competitors. The Friedman aligned ranks test is also implemented over IGD scores obtained with 25,000 function evaluations by the most popular MO algorithms given in Table 7. Results in Table 13 indicate that the MOICA again performs best and is comparable to the six competing algorithms.

5 Conclusion

In this paper, we propose a MOICA for solving global multi-objective optimization problems. The search mechanism used in this algorithm starts several empires with LNDSs in positions around the search space. In addition, revolution operations, which enable the MOICA to competitively converge and diverge, were proposed and compared

Table 11 MOICA's ranking compared to algorithms in CEC 2009

UF1	IGD	UF2	IGD	UF3	IGD
MOICA	0.0035	MOICA	0.0018	MOEAD	0.00742
MOEAD	0.00435	MTS	0.00615	MOICA	0.0103
GDE3	0.00534	MOEADGM	0.0064	LiuLiAlgorithm	0.01497
MOEADGM	0.0062	DMOEADD	0.00679	DMOEADD	0.03337
MTS	0.00646	MOEAD	0.00679	MOEADGM	0.049
LiuLiAlgorithm	0.00785	OWMOSaDE	0.0081	MTS	0.0531
DMOEADD	0.01038	GDE3	0.01195	ClusteringMOEA	0.0549
NSGAIILS	0.01153	LiuLiAlgorithm	0.0123	AMGA	0.06998
OWMOSaDE	0.0122	NSGAIILS	0.01237	DECMOSA-SQP	0.0935
ClusteringMOEA	0.0299	AMGA	0.01623	MOEP	0.099
MOEP	0.0596	ClusteringMOEA	0.0228	NSGAIILS	0.10603
DECMOSA-SQP	0.07702	DECMOSA-SQP	0.02834	GDE3	0.10639
OMOEAII	0.08564	OMOEAII	0.03057	OMOEAII	0.27141
UF4	IGD	UF5	IGD	UF6	IGD
AMGA	0.03588	MOEP	0.0189	OWMOSaDE	0.103
MOICA	0.0018	MTS	0.01489	MOEAD	0.00587
MTS	0.02356	GDE3	0.03928	MOICA	0.0127
GDE3	0.0265	AMGA	0.09405	MTS	0.05917
DECMOSA-SQP	0.03392	MOICA	0.1268	DMOEADD	0.06673
AMGA	0.04062	LiuLiAlgorithm	0.16186	OMOEAII	0.07338
DMOEADD	0.04268	DECMOSA-SQP	0.16713	ClusteringMOEA	0.0871
MOEP	0.0427	OMOEAII	0.1692	MOEP	0.1031
LiuLiAlgorithm	0.0435	MOEAD	0.18071	DECMOSA-SQP	0.12604
OMOEAII	0.04624	MOEP	0.2245	AMGA	0.12942
MOEADGM	0.0476	ClusteringMOEA	0.2473	LiuLiAlgorithm	0.17555
OWMOSaDE	0.0513	DMOEADD	0.31454	OWMOSaDE	0.1918
NSGAIILS	0.0584	OWMOSaDE	0.4303	GDE3	0.25091
ClusteringMOEA	0.0585	NSGAIILS	0.5657	NSGAIILS	0.31032
MOEAD	0.06385	MOEADGM	1.7919	MOEADGM	0.5563
UF7	IGD	UF8	IGD	UF9	IGD
MOEAD	0.00444	MOICA	0.0026	MOICA	0.0035
LiuLiAlgorithm	0.0073	MOEAD	0.0584	DMOEADD	0.04896
MOICA	0.0075	DMOEADD	0.06841	NSGAIILS	0.0719
MOEADGM	0.0076	LiuLiAlgorithm	0.08235	MOEAD	0.07896
DMOEADD	0.01032	NSGAIILS	0.0863	GDE3	0.08248
MOEP	0.0197	OWMOSaDE	0.0945	LiuLiAlgorithm	0.09391
NSGAIILS	0.02132	MTS	0.11251	OWMOSaDE	0.0983
ClusteringMOEA	0.0223	AMGA	0.17125	MTS	0.11442
DECMOSA-SQP	0.02416	OMOEAII	0.192	DECMOSA-SQP	0.14111
GDE3	0.02522	DECMOSA-SQP	0.21583	MOEADGM	0.1878
OMOEAII	0.03354	ClusteringMOEA	0.2383	AMGA	0.18861
MTS	0.04079	MOEADGM	0.2446	OMOEAII	0.23179
AMGA	0.05707	GDE3	0.24855	ClusteringMOEA	0.2934
OWMOSaDE	0.0585	MOEP	0.423	MOEP	0.342

Table 11 continued

UF10	IGD
MOICA	0.0037
MTS	0.15306
DMOEADD	0.32211
AMGA	0.32418
MOEP	0.3621
DECMOSA-SQP	0.36985
ClusteringMOEA	0.4111
GDE3	0.43326
LiuLiAlgorithm	0.44691
MOEAD	0.47415
MOEADGM	0.5646
OMOEAII	0.62754
OWMOSaDE	0.743
NSGAIILS	0.84468

Table 12 Friedman aligned ranks statistics and the corresponding p value over all algorithms in Table 11

Algorithm	Average value of Friedman aligned ranks over all CEC2009 UF problem instances	p value
MOEAD	$4.45_{(4)}$	1.6544e−06
GDE3	$7.50_{(6)}$	
MOEADGM	$8.50_{(8)}$	
MTS	$4.4_{(3)}$	
LiuLiAlgorithm	$5.90_{(5)}$	
DMOEADD	$4.35_{(2)}$	
NSGAIILS	$9.50_{(10)}$	
OWMOSaDE	$9.90_{(12)}$	
ClusteringMOEA	$9.60_{(11)}$	
AMGA	$8.30_{(7)}$	
MOEP	$9.50_{(10)}$	
DECMOSA-SQP	$8.70_{(9)}$	
OMOEAII	$10.60_{(13)}$	
MOICA	$1.70_{(1)}$	

Table 13 Friedman aligned ranks statistics and the corresponding p value over all algorithms in Table 7

Algorithm	Average value of Friedman aligned ranks over all CEC2009 UF problem instances	p value
MOICA	$3.15_{(1)}$	0.0521
NSGA-II	$3.6_{(4)}$	
SPEA2	$3.55_{(3)}$	
OMOPSO	$4.55_{(6)}$	
DMCMOABC	$6_{(7)}$	
Harmony NSGA-II	$3.7_{(5)}$	
Harmony MOEAD	$3.45_{(2)}$	

to three existing algorithms. Experimental results with three metrics showed that for most test functions, the MOICA was competitive with the baseline algorithms. The MOICA's success can be traced to its global non-dominated solutions set and approach to assimilating colonies toward these solutions, because assimilation, in which small deviations are utilized for better convergence and divergence, enables the MOICA to cover the entire search space.

Acknowledgements This study was funded by Eastern Mediterranean University (02).

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

References

Atashpaz-Gargari E, Lucas C (2007) Imperialist competitive algorithm: an algorithm for optimization inspired by imperialistic competition. IEEE Congr Evolut Comput 7:4661–4666. doi:10.1109/CEC.2007.4425083

Deb K (2001) Multiobjective optimization using evolutionary algorithms. Wiley, Chichester

Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans Evolut Comput 6(2):182–197. doi:10.1109/4235.996017

Dorigo M, Blum C (2005) Ant colony optimization theory: a survey. Theor Comp Sci 344:243–278. doi:10.1016/j.tcs.2005.05.020

Doush IA, Bataineh MQ (2015) Hybedrized NSGA-II and MOEA/D with harmony search algorithm to solve multi-objective optimization problems. In: Arik S, Huang T, Lai W, Liu Q (eds) Neural information processing. Springer, Switzerland, pp 606–614

Duan H, Xu C, Liu S, Shao S (2010) Template matching using chaotic imperialist competitive algorithm. Pattern Recogn Lett 31:1868–1875. doi:10.1016/j.patrec.2009.12.005

Ebrahimzadeh A, Addeh J, Rahmani Z (2012) Control chart pattern recognition using K-MICA clustering and neural networks. ISA Trans 51(1):111–119. doi:10.1016/j.isatra.2011.08.005

Eiben AE, Smit SK (2011) Evolutionary algorithm parameters and methods to tune them. In: Hamadi Y, Monfoy E, Saubion F (eds) Autonomous search. Springer, Berlin, pp 15–36

Fonseca CM, Fleming PJ (1993) Genetic algorithms for multiobjective optimization: formulation, discussion and generalization. In: Forrest S (ed) Proceedings of the 5th international conference on genetic algorithms. Morgan Kauffman Publishers, San Mateo, CA, pp 416–423

Fonseca CM, Fleming PJ (1998) Multiobjective optimization and multiple constraint handling with evolutionary algorithms—part II: application example. IEEE Trans Syst Man Cybern (A) 28:38–47. doi:10.1109/3468.650319

Goudarzi M, Vahidi B, Naghizadeh RA (2013) Optimum reactive power compensation in distribution networks using imperialistic competitive algorithm. Sci Int (Lahore) 25(1):27–31

Horn J, Nafploitis N, Goldberg DE (1994) A niched Pareto genetic algorithm for multiobjective optimization. In: Michalewicz Z (ed) Proceedings of the 1st IEEE conference on evolution computer. IEEE Press, Piscataway, NJ, pp 82–87

Jordehi AR (2016) Optimal allocation of FACTS devices for static security enhancement in power systems via imperialistic competitive algorithm (ICA). Appl Soft Comput 48:317–328. doi:10.1016/j.asoc.2016.07.014

Kashani AR, Gandomi AH, Mousavi M (2014) Imperialistic competitive algorithm: a metaheuristic algorithm for locating the critical slip surface in 2-dimensional soil slopes. Geosci Front 7(1):83–89. doi:10.1016/j.gsf.2014.11.005

Kennedy J, Eberhart R (1995) Particle swarm optimization. In: Proceedings of the IEEE international conference on neural network IV 1942–1948

Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. Science 220(4598):671–680. doi:10.1126/science.220.4598.671

Kursawe F (1990) A variant of evolution strategies for vector optimization. In: Schwefel H-P, Manner R (eds) Parallel problem solving from nature. Springer, Berlin, pp 193–197

Mitchell M (1999) An introduction to genetic algorithms. MIT Press, Cambridge

Nazari-Shirkouhi S, Eivazy H, Ghodsi R, Rezaie K, Atashpaz-Gargari E (2010) Solving the integrated product mix-outsourcing problem by a novel meta-heuristic algorithm: imperialist competitive algorithm. Expert Syst Appl 37:7615–7626. doi:10.1016/j.eswa.2010.04.081

Niknam T, Taherian FE, 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

Qi Y, Ma X, Liu F, Jiao L, Sun J, Wu J (2014) MOEA/D with adaptive weight adjustment. Evol Comput 22(2):231–264. doi:10.1162/EVCO_a_00109

Razmjooy N, Mousavi BS, Soleymani F (2013) A hybrid neural network imperialist competitive algorithm for skin color segmentation. Math Comput Model 57:848–856. doi:10.1016/j.mcm.2012.09.013

Schaffer JD (1987) Multiple objective optimization with vector evaluated genetic algorithms. In: Grefensttete JJ (ed) Proceedings of the 1st international conference on genetic algorithms. Lawrence Erlbaum, Hillsdale, NJ, pp 93–100

Seyedmohsen H, Abdullah AK (2014) A survey on the imperialist competitive algorithm metaheuristic: implementation in engineering domain and directions for future research. Appl Soft Comput 24:1078–1094. doi:10.1016/j.asoc.2014.08.024

Sherinov Z, Unveren A, Acan A (2011) An evolutionary multi-objective modeling and solution approach for fuzzy vehicle routing problem. In: International symposium on innovations in intelligent systems and applications (INISTA), pp 450–454. doi:10.1109/INISTA.2011.5946143

Srinivas N, Deb K (1995) Multiobjective function optimization using nondominated sorting genetic algorithms. Evolut Comput 2(3):221–248. doi:10.1162/evco.1994.2.3.221

Van Veldhuizen DA, Lamont GB (1998) Multiobjective evolutionary algorithm research: a history and analysis. Technical Report TR-98-03, Department of Electrical and Computer Engineering, Graduate School of Engineering, Air Force Institute of Technology (AFIT), Wright-Patterson AFB, OH

Vedadi M, Vahidi B, Hosseinian SH (2015) An imperialist competitive algorithm maximum power point tracker for photovoltaic string operating under partially shaded conditions. Sci Int (Lahore) 27(5):4023–4033

Xiang Y, Zhou Y (2015) A dynamic multi-colony artificial bee colony algorithm for multi-objective optimization. Appl Soft Comput 35:766–785. doi:10.1016/j.asoc.2015.06.033

Zhang Q, Zhou A, Zhao S, Suganthan PN, Liu W, Tiwari S (2009) Multiobjective optimization test instances for the CEC 2009 special session and competition. Technical Report CES-487, University of Essex, Essex, UK

Zitzler E, Deb K, Thiele L (2000) Comparison of multiobjective evolutionary algorithms: empirical results. Evolut Comput 8(2):173–195. doi:10.1162/106365600568202

Zitzler E, Thiele L (1998) Multiobjective optimization using evolutionary algorithms: a comparative case study. In: Eiben AE, Bäck T, Shoenauer M, Schwefel HP (eds) Parallel problem solving from nature. Springer, Berlin, pp 292–301

Zitzler E, Thiele L, Laumanns M, Fonseca CM, Grunert da Fonseca V (2003) Performance assessment of multiobjective optimizers: an analysis and review. IEEE Trans Evolut Comput 7(2):117–132. doi:10.1109/TEVC.2003.810758