A robust experimental evaluation of the co-evolutionary islands model

Caio E. Marcondes caio.marcondes@sou.unifal-mg.edu.br
Department of Computer Science, Universidade Federal de Alfenas, Alfenas, 37133-754, Brazil

Guilherme A. Gouveia guilherme.gouveia@sou.unifal-mg.edu.br
Department of Computer Science, Universidade Federal de Alfenas, Alfenas, 37133-754, Brazil

Iago A. Carvalho iago.carvalho@unifal-mg.edu.br
Department of Computer Science, Universidade Federal de Alfenas, Alfenas, 37133-754, Brazil

Abstract

The islands model is a co-evolutionary framework for developing evolutionary algorithms. In this model, we have two or more islands, and every island contains a different population that evolves semi-isolated from the others, exchanging solutions among them periodically through a migration operator. This work performs a robust and extensive evaluation of this model. We evaluate implementations ranging from two to fifty islands. Furthermore, we evaluate this model using five different evolutionary algorithms: the genetic algorithm, the ant colony optimization, the particle swarm optimization, the differential evolution, and the CLONAL algorithm, whereas every algorithm can be employed to evolve none, one, or more islands. This evaluation is carried out using the 2015 IEEE Congress of Evolutionary Computation's Black Box Optimization Competition as test functions. A statistical analysis of our experimental data demonstrated that the use of several islands is beneficial to the results of the evolutionary algorithms. In fact, we were able to infer that the use of 8, 10, or more islands led to the best results found.

Keywords

Islands model, co-evolution, hybrid algorithms.

1 Introduction

The islands model is a simple and effective framework for developing co-evolutionary heuristics. In this framework, there are two or more islands, and each island denotes a different population. These islands evolve semi-isolated from each other, whereas the only communication mechanism is a migration operator that exchanges solutions among the islands. Algorithms developed through this framework can be easily implemented in distributed and parallel environments (Whitley et al., 1998).

We denote a heuristic developed using this framework as an island-based evolutionary algorithm (IBEA). The main difference between an IBEA and a single-population evolutionary algorithm (SPEA) is the separation of individuals into islands, also denominated subpopulations. Islands interact with each other through migration, spreading genetic information of individuals among the whole population. Without

this migration operator, an IBEA is equivalent to a set of separate runs of SPEA (Skolicki and De Jong, 2005).

An island p_m is a set of individuals. An IBEA uses a set $P = \{p_1, p_2, \ldots, p_m\}$ of m islands, whereas p_m^i is the i-th individual of island p_m. Thus, we argue that P denotes the set of all individuals of an IBEA and that $|P| = \sum_{i=1}^{m} |p_i|$. In the remaining of this paper, we will employ the terms *population*, *subpopulation*, and *island* as synonymous, and they will refer to a single island (or population) $p_m \in P$.

In addition to the common parameters of the evolutionary algorithms, an IBEA also has the migration rate (m_r), the migration size (m_s), and the number of islands (m). Parameter m_r represents the interval of generations between two migration processes, and m_s denotes the total number of individuals that migrate each m_r generation. Both m_r, m_s and m have great influence on IBEA convergence (Tomassini, 2005; Da Silveira et al., 2023).

Algorithm 1 shows a pseudo-code of a general IBEA. First, it initializes all islands $p_i \in P$ from lines 1 to 3. Next, each island evolves for m_r generations from lines 5 to 9. Then, the migration operator is applied in line 10. The algorithm executes until a stopping criterion is met from lines 4 to 11). Finally, the best solution found among all islands is selected and returned in line 12 and the algorithm is ended. One can see that this is a synchronous IBEA implementation. A distributed version of IBEA can be easily developed by computing the evolutionary process from lines 5 to 9 in parallel. Details regarding this parallel implementation can be found in Alba (2005) and Luque and Alba (2011).

Algorithm 1 The islands model framework

```
1:  for i ← 1 to m do                                  ▷ loop for all islands p_i ∈ P
2:      initialize island p_i
3:  end for
4:  while stopping criterion is not met do
5:      for j ← 1 to m_r do                            ▷ loop for a total of m_r generations
6:          for i ← 1 to m do                          ▷ loop for all islands p_i ∈ P
7:              evolve one generation of the island p_i
8:          end for
9:      end for
10:     apply the migration operator
11: end while
12: return best individual of all islands
13: end
```

Our work uses two different migration operators: (i) the island-migration; and (ii) a neighborhood-based migration operator (Tomassini, 2005). The island-migration is exemplified in Figure 1 for an IBEA with $m = 4$. We can see from this figure that an individual from population p_i can migrate to any population $p_i \in P$, also being able to stay in their current population.

The neighborhood-based operator in our algorithm, on the other hand, defines a subset of neighbour islands that are connected. This connection is set randomly and a total of $\max\{m - 1, 4\}$ connections are set by the operator before the start of the evolutionary process. Thus, in this operator, an individual can only migrate between these connected islands.

This paper studies the co-evolutionary islands model as described below. We per-

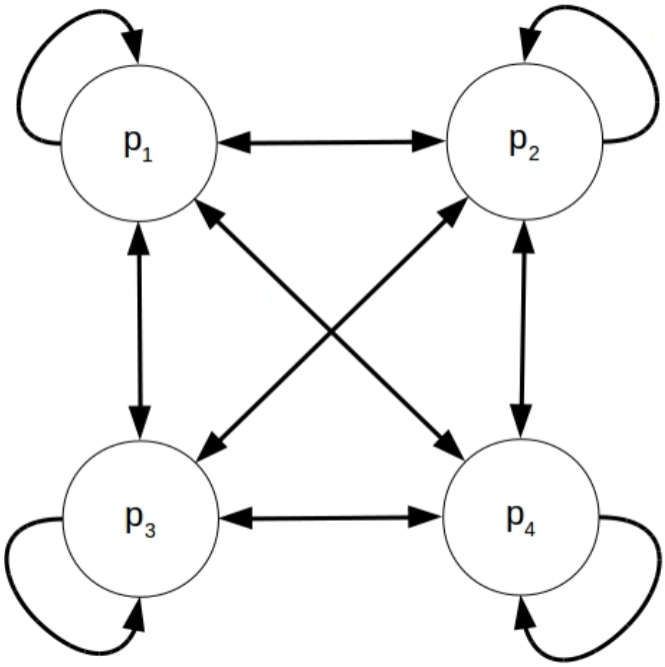


Figure 1: The islands-migration operator. An individual can freely migrate among all subpopulations

form an extensive experimental study with IBEAs with up to 50 islands. Furthermore, the IBEAs evaluated in this work employ from 1 to 5 different evolutionary algorithms: a genetic algorithm (GA) (Reeves, 2010), a differential evolution algorithm (DE) (Storn and Price, 1997), a particle swarm optimization algorithm (PSO) (Kennedy and Eberhart, 1995), an ant colony optimization algorithm (ACO) (Socha and Dorigo, 2008), and a clonal selection algorithm (CLONAL) (De Castro and Von Zuben, 2000). The developed algorithms are evaluated using the test functions of the 2015 IEEE CEC competition on learning-based real-parameter single objective optimization (Liang et al., 2014).

The remainder of this paper is organized as follows. Section 2 reviews the related work. Next, Section 3 presents the benchmark objective functions along with the solution representation for this problem, while Section 4 presents the five different algorithms employed in our IBEA. Then, Section 5 analyzes the developed IBEA, presenting and analyzing a robust computational experiment with IBEAs ranging from 2 to 50 islands. Finally, Section 6 draws the concluding remarks of our paper.

2 Related work

Several studies on IBEA can be found in the literature. The work developed by Lässig and Sudholt (2013) demonstrated the impact of migration parameters in the solution's diversity and convergence of IBEA. It indicated that an IBEA with a high migration rate is similar to an SPEA. Moreover, a small migration rate can retard the evolutionary process, such that a greater number of function evaluations are needed to achieve good solutions. Furthermore, it also exposes that an IBEA with a dense-connected topology achieves better results with larger migration intervals, while a small migration interval is indicated to be used with a sparse-connected topology.

A concept of migration through copies is described by Schwehm (1996). According to him, when an individual migrates from subpopulation p_i to a subpopulation p_j, it continues to exist at subpopulation p_i. In contrast, the work developed by Martin et al. (1997) presented a process that, when an individual migrates from subpopulation p_i to a subpopulation p_j, the individual no longer exists in subpopulation p_i. However, do not exist any work in the literature that compares these two different migration assimilation strategies.

An IBEA without the migration operator, being all subpopulations isolated from each other, was studied and compared to SPEA in Cantú-Paz and Goldberg (2003). It shows that an IBEA, when all subpopulations are isolated from each other, is equivalent to multiple runs of an SPEA. Computational experiments have reported that a single run of an SPEA without population subdivision can outperform an IBEA with isolated islands for a set of additively separable functions.

A work that deals with different migration policies was developed by Alba and Troya (2000). It showed that migration policies suffer great influence from migration topologies. Another work that dealt with different migration policies and their effects on IBEA was presented by Cantú-Paz (2001). It showed various migration policies and studied their impact on IBEA convergence. It showed that choosing random individuals for migration does not increase the selection pressure. Thus, it can be appropriate for studying other migration parameters (Skolicki and De Jong, 2005).

Another work that compared various migration policies was developed by Magalhães et al. (2015). It presented an IBEA with ring topology and showed that migration policies based on elitist strategies are efficient in solving simple problems. Besides, complex problems are better handled by migration policies based on similarities and fitness comparisons.

A comparison between different migration topologies was developed by Sekaj (2004). That work compared the most often used topologies, such as the ring, array, grid, star, and fully-connected topologies. It showed that models with sparse connections have greater solution diversity, as they spread the individuals' genetic information slowly to the whole population. On the other hand, densely connected topologies with a high migration rate behave like a single population.

Skolicki and De Jong (2005) studied the influence of migrations size and interval on IBEA. The authors demonstrated that IBEA algorithms are very sensitive to these two variables. Their work showed that a small migration interval should be avoided to achieve better results. Besides, this work also correlated a high migration size with IBEA's performance degradation.

Tomassini (2005) presented various models of distributed evolutionary algorithms. It presented IBEA, lattice cellular models, coevolutionary models, and some other non-conventional models. Various parameters of these models were presented.

The speed at which good solutions of an island spread through the population was studied by Alba and Luque (2005). Computational experiments performed with different migration intervals, and migration size demonstrated how to adjust these parameters for different topologies.

Several IBEA's migration topologies were studied and compared by Ruciński et al. (2010). Besides the migration topologies, this work varied the number of islands, using two IBEA to optimize three different mathematical functions. It presented various migration topology parameters, such as the degree of connectivity and the clustering coefficient, demonstrating that these topologies' parameters have a great influence on IBEA.

A multicultural migration policy was developed by Araujo and Merelo (2011). An individual can migrate from population $p_i \in P$ to population $p_j \in P$ if and only if its genotype greatly differs from the population p_j average genotype. Thus, it can increase the solution's diversity in all subpopulations. This policy has shown to be more effective than others when one has small-sized subpopulations.

A method for analyzing the runtime of parallel evolutionary algorithms was presented by Lässig and Sudholt (2014). The presented method can be used to estimate the upper bound of IBEA's running time through the generalization of its single-population version's running time. This method was validated through computational experiments performed to optimize two pseudo-Boolean functions. Running times' upper bound for IBEA with ring topology, hypercube topology, torus graph topology and fully-connected topology were drawn.

A framework for developing IBEA for multi-objective optimization was presented by Vargas et al. (2015). It showed that subpopulation dynamics can greatly improve Evolutionary Algorithms' results in the case of multi-objective problems. Furthermore, it compared IBEA with other distributed models of SPEA, showing that IBEA overcame these models for multi-objective optimization.

Mambrini and Sudholt (2015) proposed two adaptive migration schemes for IBEA. These schemes adjust the migration interval by analyzing whether islands have managed to find improved solutions since the last migration. If an island finds an improved solution, its migration interval decreases to spread this solution to the whole population. On the other hand, if an island does not find an improved solution since the last migration, this island's migration interval increases to avoid large communication effort (when dealing with parallel hardware). Computational experiments showed that both frameworks could reduce the communication effort compared to a traditional migration scheme. A similar work was developed by Osorio et al. (2011, 2013). However, their algorithms only ran for a fixed number of generations.

Duarte et al. (2021) proposed an islands model based on stigmergy, a natural phenomenon by which groups of individuals develop organized and cooperative behavior to perform tasks and increase their survival. In their model, the migration rates between islands can change during the evolutionary process, being adapted to benefit the best populations. This stigmergy island model was shown to outperform the traditional islands model. Later, Duarte et al. (2022) improved the stigmergy model by distributing constraint-handling methods across the several islands of their model.

Skakovski and Jedrzejowicz (2019) proposed an IBEA with no migration operator composed of several DE algorithms. In their model, every DE algorithm implements a decloning operator (Jedrzejowicz and Skakovski, 2016) that dynamically changes its population size. Computational experiments performed by the authors demonstrated that the IBEA implemented with the decloning operator outperforms another implemented using the classical DE algorithm in solving a scheduling problem. Later, Skakovski and Jedrzejowicz (2022) extended their previous work by demonstrating that the previously proposed IBEA algorithm outperforms the classical one even if no migration operator is applied.

Da Silveira et al. (2019) contrasted an IBEA constructed with several GA with its SPEA variant for solving bioinformatics problems. The authors demonstrated that the IBEA implementation could find better results than the SPEA, considering its running time and solution quality.

Silva et al. (2017) proposed an IBEA with multiple DE algorithms for solving a calorie-restricted diet problem. In their implementation, every island contained the

same algorithm but with different parameters. Computational experiments demonstrated that their IBEA could obtain higher-quality solutions than a DE. Later, Xavier et al. (2023) expanded this work by developing an IBEA with multiple GA and DE algorithms. This latter algorithm outperformed all other algorithms in the literature in solving the calorie-restricted diet problem.

Da Silveira et al. (2023) evaluated several IBEA implemented using multiple GAs over different migration topologies. The authors evaluated many parameters of the algorithms, including the comparison between synchronous and asynchronous migration, the migration size, and the migration interval. Their results indicate the best parameter setting for each migration topology when solving four NP-hard combinatorial optimization problems.

Da Silveira et al. (2021) evaluated an IBEA that employs three different algorithms within its islands: $i)$ a GA; $ii)$ a DE; and $iii)$ a PSO. The authors demonstrated that the use of multiple algorithms is beneficial in terms of solution quality in comparison to an IBEA implemented with several GAs.

Our paper extends the work of Da Silveira et al. (2021) using the island-migration model of Tomassini (2005). In addition to the three algorithms employed by Da Silveira et al. (2021), we also evaluate a CLONAL algorithm and an ACO algorithm within IBEA. Furthermore, we extend the evaluation of these algorithms for IBEAs with up to 50 islands.

3 The benchmark objective functions and their computational representation

The proposed algorithms were evaluated using 15 different global optimization problems, here denoted as benchmark objective functions. These benchmark objective functions were previously employed in the 2015 IEEE CEC competition on learning-based real-parameter single objective optimization (Liang et al., 2014).

Table 1 presents the employed functions. The first and second columns respectively present the function's number and name. The third column gives the number of dimensions d of the benchmark objective function. Finally, the fourth column denotes the optimal value of the objective function. One can observe that the value of d varies in the set $\{10, 30\}$ for all benchmark objective functions. The higher the value of d, the harder to solve the benchmark objective function.

We denote a solution for a benchmark objective function as an individual p_m^i of an IBEA. For the problem we solved, a solution is represented as a d-dimensional vector of real numbers in the interval $[-100, 100]$. Figure 2 represents a solution for $d = 10$. We denote each position of this vector as a *gene*. Furthermore, we denote the value of applying solution p_m^i in solving a proposed benchmark objective function as it *fitness value* $f(p_m^i)$.

3.2	7.4	9.1	0.9	1.5	0.5	8.8	3.2	5.8	4.6

Figure 2: Solution representation for a benchmark optimization function with $d = 10$

4 Algorithms employed within IBEA

Our work implements and evaluates a synchronous IBEA as described in Algorithm 1. As stated in previous sections, we implemented five different algorithms within IBEA.

Table 1: Benchmark objective functions evaluated in this work

Number	Function's name	d	f_{min}
F1	Rotated Bent Cigar Function	$10, 30$	100
F2	Rotated Discus Function	$10, 30$	200
F3	Shifted and Rotated Weierstrass Function	$10, 30$	300
F4	Shifted and Rotated Schwefel's Function	$10, 30$	400
F5	Shifted and Rotated Katsuura Function	$10, 30$	500
F6	Shifted and Rotated HappyCat Function	$10, 30$	600
F7	Shifted and Rotated HGBat Function	$10, 30$	700
F8	Shifted and Rotated Expanded	$10, 30$	800
F9	Shifted and Rotated Expanded Scaer's F6 Function	$10, 30$	900
F10	Hybrid Function 1 (N = 3) Rastrigin	$10, 30$	1000
F11	Hybrid Function 2 (N = 4) Rastrigin	$10, 30$	1100
F12	Hybrid Function 3 (N = 5)	$10, 30$	1200
F13	Composition Function 1 (N = 5)	$10, 30$	1300
F14	Composition Function 2 (N = 3)	$10, 30$	1400
F15	Composition Function 3 (N = 5)	$10, 30$	1500

We opted for a traditional implementation of these algorithms that does not employ sophisticated methods or operators. They are described below.

4.1 Genetic Algorithm

Holland et al. (1992) proposed the GA, inspired by the evolution theory presented by Charles Darwin. GA updates its population based on the crossing, mutation, and evolution operators. It was initially proposed for binary representation of the individuals. However, several variants proposed in the literature extended their application to deal with real-valued variables (Reeves, 2010), as in this work.

Algorithm 2 shows the pseudo-code of our GA implementation. First, the population is initialized in line 1 and evaluated in line 2. Then, the loop from lines 3-8 is performed until a stopping criterion is met. Every iteration of this loop is denominated as a *generation* of the GA. On every iteration, the selection, crossing, mutation, and evolution operators are applied sequentially. Finally, the best individual found by GA is returned in line 9.

Algorithm 2 The Genetic Algorithm (GA)

```
1: Population initialization
2: Population evaluation
3: while stopping criterion is not met do
4:     Selection
5:     Crossing
6:     Mutation
7:     Evolution
8: end while
9: return the best individual found
```

4.1.1 Selection operator

The selection operator obtains one or two random individuals to undergo the evolution process. First, it selects a random individual from population p_i using a roulette algorithm (Reeves, 2010). Then, it generates a pseudo-random number ι using a real uniform distribution $U(0, 1)$. If ι is smaller or equal to the crossing rate parameter, then this previously selected individual is used as input for the mutation operator, thus not applying the crossing operator. Otherwise, it selects a different individual using the same roulette algorithm and employs both individuals as input for the crossing operator.

4.1.2 Crossing operator

The GA employs the classic *2-point crossover* operator (Reeves, 2010). This operator mixes the genes of the selected individuals and creates a new individual. This new individual will be used as input for the mutation operator.

4.1.3 Mutation operator

The mutation operator changes the value of a single gene of the previously generated individual. The gene to be mutated is chosen using an integer uniform distribution $U(1, d)$ and its new value is generated in the same way as in the population initialization, that is, using a continuous uniform distribution $\mathcal{U}(-100, 100)$. This operation is applied with a certain probability regulated by the parameter mutation rate.

4.1.4 Evolution operator

The previous operators are applied until the number of generated individuals is equal to the original population size. The selection operator constructs a temporary pool of solutions containing the original population and all of the generated individuals. Then, it selects half of them to undergo to the next generation.

This selection process simply sorts the temporary pool of solutions according to their fitness. Then, the individuals with the best fitness values are selected and the others are discarded. One may observe that this operator preserves the original population's size.

4.2 Differential Evolution

The DE algorithm was first proposed by Storn and Price (1997). It is a stochastic evolutionary algorithm being first proposed to deal with real-valued variables. The pseudocode of DE is shown in Algorithm 3. First, the population is initialized in line 1 and evaluated in line 2. Next, the loop from lines 3-7 is performed until a stopping criterion is met. Every iteration of this loop is a *generation* of the DE. In every generation, mutation, crossing, and selection operators are applied sequentially. Finally, the best individual found by DE is returned in line 8.

4.2.1 Mutation operator

The mutation operator is applied using three different individuals p_m^i, p_m^j, and p_m^k from the population. These three individuals are randomly chosen with the same probability. The mutation operator consists of modifying the genes of p_m^i by the vector difference between the chromosomes of p_m^j and p_m^k multiplied by a perturbation factor $F \in [0, 2]$, which is a parameter of DE. This operator creates a new individual c and can be described as

$$c = p_m^i + F \cdot (p_m^j \times p_m^k).$$

This new individual c will be used as input for the crossing operator.

Algorithm 3 The Differential Evolution (DE)

1: Population initialization
2: Population evaluation
3: **while** stopping criterion is not met **do**
4: Mutation
5: Crossing
6: Selection
7: **end while**
8: **return** the best individual found

4.2.2 Crossing operator

The crossing operator selects a random individual p_m^l from the population with a uniform probability such that $p_m^l \neq p_m^i \neq p_m^j \neq p_m^k$ and combines p_m^l with c. The crossing operator constructs a new individual a by randomly selecting the genes from p_m^l and c with a crossing probability $\beta \in [0, 1]$. For every gene in $[1, \ldots, d]$, it generates a pseudo-random number using a real uniform distribution $U(0, 1)$. If the generated number is smaller or equal to the crossing probability β, then the i-th gene of individual a will be copied from that of c. Otherwise, the i-th gene of a will be the same as that of p_m^l.

4.2.3 Selection operator

The selection operator of DE is carried out as the evolution operator of GA, as described in Section 4.1.4.

4.3 Particle Swarm Optimization

The PSO is an evolutionary algorithm based on the concept of swarms of particles (Wang et al., 2018). It tries to mimic the intelligent collective behavior of some animals such as birds, fishes, and insects (Kennedy and Eberhart, 1995). These swarms collaborate to locate food, with each member adjusting their search pattern based on their own learning experiences and those of other members (Wang et al., 2018). In this algorithm, the particles are located in a d-dimensional space, and the position of a particle corresponds to its fitness value.

The pseudo-code of PSO is shown in Algorithm 4. Initially, all particles are initialized in line 1. Then, the position of all particles is stored as its best position in line 2 and the best global position is stored in line 3. Then, the loop from lines 4-15 is performed until a stopping criterion is met. We denote each iteration of this loop as a generation for the PSO. In every generation, we individually update the particle's speed in line 6 and the particles's position in line 7. Then, the best position for the particle and the global best position are updated in lines 8-13. Finally, the global best position is returned in line 16.

4.3.1 Updating the paticle's speed and position

Let v_{ij} be the speed of a particle j in dimension i. Additionally, let x_{ij} denotes the position of particle j at dimension i. We update the particle's speed as

$$v_{ij} = v_{ij} \cdot w + c_1 \cdot r_1 \cdot (p_{ij}^{best} - x_{ij}) + c_2 \cdot r_2 \cdot (g_i^{best} - x_{ij}),$$

where p_{ij}^{best} corresponds to the best position for the particle and g_i^{best} is the global best position at dimension i. Furthermore, w is a fixed value that corresponds to the inertia

Algorithm 4 Particle Swarm Optimization (PSO)

```
1: Initialize particle positions and velocities
2: Initialize the best position for each particle
3: Initialize the global best position
4: while stopping criterion is not met do
5:     for for every particle do
6:         Update particle's speed
7:         Update particle's position
8:         if fitness of the particle is better than its best position then
9:             Update the best position of the particle
10:        end if
11:        if fitness of the particle is better than the global best position then
12:            Update the global best position
13:        end if
14:    end for
15: end while
16: return global best solution
```

and c_1, c_2 are the social components of PSO, which are real-valued parameters in the interval $[0.1, 3]$.

After the speed update, we update the particle's position as

$$x_{ij} = v_{ij} + x_{ij}.$$

4.4 Ant Colony Optimization

The ACO is inspired by the foraging behavior of ant colonies (Dorigo and Stützle, 2019). Initially, ants move randomly in search of food, creating multiple routes. Then, based on food quality and quantity, they carry food back, leaving pheromone trails. The probability of selecting a path depends on pheromone concentration and evaporation rate.

The pseudo-code of ACO is shown in Algorithm 5. Initially, the pheromone trails are initialized in line 1. Then, the initial position of all ants defined in line 2 and the best fitness value is stored in line 3. Then, the loop from lines 4-9 is performed until a stopping criterion is met. We denote each iteration of this loop as a generation for the ACO. In every generation, we individually update the ants position in line 5 and evaluates their fitness line 6. Then, the pheromone trails are updated in line 7, and the ant with the best fitness value is stored in line 8. Finally, the ant with the best fitness value, across all generations of the ACO, is returned in line 10.

We implemented the continuous ACO algorithm as described by Bilchev and Parmee (1995).

4.5 Clonal Selection Algorithm

The CLONAL is an evolutionary algorithm inspired by the selection principle of immunobiologic systems (De Castro and Von Zuben, 2000). It is based on the idea that each living being will be exposed to the same antigen repeatedly, always forming antibodies capable of reacting against that antigen. Combining this idea with concepts of evolutionary algorithms, the implementation of CLONAL relies on a population of antibodies (solutions) that, with each iteration, will be cloned and modified. After their

Algorithm 5 Ant Colony Optimization algorithm (ACO)

```
1: Initialize the pheromones
2: Define the initial position of the ants
3: Stores the ant with the best fitness value
4: while stopping criterion is not met do
5:     Update the position of the ants
6:     Evaluate the fitness of the ants
7:     Update the pheromone trails
8:     Stores the ant with the best fitness value
9: end while
10: return the ant with the best fitness value
```

modifications, the clones best prepared to deal with the antigens are selected to be part of the next population (Carvalho and Ribeiro, 2019).

Algorithm 6 shows the pseudo-code of our CLONAL implementation. Initially, all antigens are randomly initialized in line 1. Then, the antigen with the best fitness value is stored in line 2. The loop from lines 3-8 corresponds to a generation of CLONAL. In every generation, we clone (line 4), hypermutate (line 5), evaluate (line 6), and select (line 7) the antigens that will undergo to the next generation. Finally, the best antigen found in returned in line 9.

Algorithm 6 The Clonal Selection Algorithm (CLONAL)

```
1: Initialize the antigens
2: Stores the antigen with the best fitness value
3: while stopping criterion is not met do
4:     Clone
5:     Hypermutate
6:     Evaluate
7:     Selection
8: end while
9: return the best antigen found
```

4.5.1 Cloning operator

A *clone* is an exact copy of an antigen. The cloning operator creates a vector of clones for every antigen of the population. The number of clones is defined by the *cloning scale*, which is a parameter of the algorithm, and is the same for every antigen of CLONAL.

4.5.2 Hypermutation operator

The hypermutation operator applies mutations in the previously created clones. Our hypermutation operator is the *proportinal hypermutation operator* (Carvalho and Ribeiro, 2019). This operator ranks all original antigens of the population according to their fitness values. Then, the number of mutations applied to them is inversely proportional to its ranking. Therefore, we apply a smaller number of mutations to the antigens with the best fitness value and a larger number of mutations to antigens with the worst fitness values. It guarantees that CLONAL intensifies the search around good-quality antigens, while low-quality antigens are employed to explore the search space.

A mutation of CLONAL is the same of the GA's mutation as described in Section 4.1.3. The clones of the antigen with the best fitness value has only one variable mutated. Conversely, the clones of the antigen with the worst fitness value are mutated d times, where d is equal to the number of dimensions of the problem, as presented in Section 3.

4.5.3 Selection operator

The selection operator is applied individually for every antigen and their clones. It chooses, among the original antigen and his mutated clones, the one with the best fitness value. This selected solution goes to the next generation, while all others are discarded. This operator guarantees that the number of antigens do not vary during the execution of CLONAL.

5 Computational experiments

The computational experiments were carried out on a personal computer using the Ubuntu 22.04.1 operating system. The experiments ran on a single core of an Intel i5-12500 processor with 4.6 Ghz clock and 16 Gb of RAM. All algorithms were developed from scratch in C and compiled using the GNU GCC 11.3.0. Additionally, all experiments were repeated 30 times with different seeds to the Mersene-Twister pseudo-random number generator (Matsumoto and Nishimura, 1998) to obtain the average fitness and its standard deviation. We set three different stopping criteria: (i) a maximum running time of 10 seconds; (ii) a total of 5000000 objective function evaluations; and (iii) a total of 1000 generations without improvement of the objective function value.

5.1 Hyperparameter optimization for each algorithm

In this first experiment, we optimize the parameters for the 5 algorithms described in Section 4. This experiment was carried out separately for each algorithm with a single population, i.e., their SPEA implementations. The objective was to find the best set of parameters for the algorithms to be employed in the remaining experiments.

The hyperparameters optimization was carried out using benchmark objective functions $F3, F5$, and $F12$. These three functions were randomly selected and removed from the remaining experiments to prevent the overfitting of the algorithms to the underlying optimization problems.

The iRace package (López-Ibáñez et al., 2016) was employed for this experiment. This package offers a well-known suite for automatic algorithm configuration in optimization and learning tasks. We used the *Iterated Race* method available in the iRace package developed in R programming language.

Table 2 shows the results of this experiment. The first column presents the algorithm's name, while the second column gives the name of the parameter being optimized. The third column shows the considered interval, i.e., the minimum and maximum value evaluated for each parameter. Finally, the fourth column presents the results obtained through the application of the Iterated Race procedure. As these are the best parameter settings for each algorithm individually, they will be carried out in the remaining experiments.

5.2 Evaluation of the evolutionary algorithms

The second experiment separately evaluates the five evolutionary algorithms implemented in this work. The objective is simply to assert their performance in optimizing the proposed benchmark objective functions. This experiment was carried out in

Table 2: The hyperparameter optimization experiment

Algorithm	Parameter	Test interval	Best value
GA	Population size	$[30, 10000] \in \mathbb{N}$	3229
	Mutation rate	$[0, 1] \in \mathbb{R}$	0.13
	Crossing rate	$[0, 1] \in \mathbb{R}$	0.08
DE	Population size	$[30, 10000] \in \mathbb{N}$	1665
	Mutation rate	$[0, 1] \in \mathbb{R}$	0.05
	Pertubation factor	$[0, 2] \in \mathbb{R}$	0.82
PSO	Population size	$[30, 10000] \in \mathbb{N}$	9111
	Social component 1	$[0.1, 3] \in \mathbb{R}$	0.61
	Social component 2	$[0.1, 3] \in \mathbb{R}$	0.74
ACO	Population size	$[30, 10000] \in \mathbb{N}$	2828
	Number of candidates	$[3, 30] \in \mathbb{N}$	20
CLONAL	Population size	$[30, 10000] \in \mathbb{N}$	46
	Number of clones	$[2, 100] \in \mathbb{N}$	72

the remaining 12 benchmark objective functions described in Table 1 excluding those employed in the first experiment. The objective was to assess the performance of the five developed algorithms in solving the proposed benchmark objective functions. Although this experiment not being the objective of our work, the obtained results will be employed in the discussion of our final and most important experiment given in Section 5.4.

The result of this experiment is given in Table 3. The first column shows the number of the benchmark objective function. The second and third columns present the results for the GA, whereas the second column (Average) gives the average result for 20 repetitions of the algorithm using different seeds for the pseudo-random number generator and the third column (SD) displays the standard deviation of this same data. The remaining columns present the same data for the DE, the PSO, the ACO, and the CLONAL algorithms. Every line summarizes the results for $d = \{10, 30\}$. Additionally, the best result in every line, lexicographically considering the average result and the standard deviation, is highlighted in bold.

Table 3: Comparison between the five proposed evolutionary algorithms

	GA		DE		PSO		ACO		CLONAL	
	Average	SD	Average	SD	Average	SD	Average	SD	Average	SD
F1	3.65×10^{7}	1.38×10^{7}	$\mathbf{7.75 \times 10^{3}}$	1.87×10^{3}	9.31×10^{3}	7.02×10^{3}	1.92×10^{9}	4.35×10^{8}	1.69×10^{5}	1.53×10^{5}
F2	2.18×10^{3}	4.99×10^{2}	1.46×10^{3}	5.78×10^{2}	$\mathbf{2.00 \times 10^{2}}$	1.75×10^{-2}	1.62×10^{4}	2.87×10^{3}	5.35×10^{3}	2.01×10^{3}
F4	9.83×10^{2}	1.09×10^{2}	9.88×10^{2}	3.17×10^{2}	6.54×10^{2}	1.18×10^{2}	1.57×10^{3}	8.60×10^{1}	$\mathbf{4.01 \times 10^{2}}$	3.66×10^{-1}
F6	6.00×10^{2}	3.84×10^{-2}	6.00×10^{2}	4.85×10^{-2}	$\mathbf{6.00 \times 10^{2}}$	4.90×10^{-2}	6.02×10^{2}	3.81×10^{-1}	6.00×10^{2}	1.55×10^{-1}
F7	7.00×10^{2}	6.32×10^{-2}	7.00×10^{2}	4.45×10^{-2}	$\mathbf{7.00 \times 10^{2}}$	3.71×10^{-2}	7.13×10^{2}	2.36	7.00×10^{2}	3.06×10^{-1}
F8	8.04×10^{2}	5.12×10^{-1}	8.02×10^{2}	6.35×10^{-1}	$\mathbf{8.01 \times 10^{2}}$	3.78×10^{-1}	1.16×10^{3}	2.17×10^{2}	8.04×10^{2}	1.36
F9	9.03×10^{2}	1.28×10^{-1}	9.03×10^{2}	2.14×10^{-1}	$\mathbf{9.02 \times 10^{2}}$	3.68×10^{-1}	9.03×10^{2}	1.46×10^{-1}	9.03×10^{2}	3.37×10^{-1}
F10	3.36×10^{3}	5.66×10^{2}	1.63×10^{3}	2.65×10^{2}	$\mathbf{1.59 \times 10^{3}}$	2.12×10^{2}	1.75×10^{4}	6.83×10^{3}	7.23×10^{3}	4.50×10^{3}
F11	1.10×10^{3}	2.62×10^{-1}	1.10×10^{3}	3.43×10^{-1}	$\mathbf{1.10 \times 10^{3}}$	8.32×10^{-1}	1.11×10^{3}	5.66×10^{-1}	1.10×10^{3}	9.76×10^{-1}
F13	1.62×10^{3}	5.44×10^{-1}	1.62×10^{3}	1.16	$\mathbf{1.62 \times 10^{3}}$	3.49×10^{-1}	1.65×10^{3}	1.07×10^{1}	1.62×10^{3}	7.54×10^{-1}
F14	1.59×10^{3}	2.21	$\mathbf{1.59 \times 10^{3}}$	2.37	1.59×10^{3}	5.15	1.60×10^{3}	1.87	1.59×10^{3}	3.85
F15	1.92×10^{3}	3.31	1.91×10^{3}	4.33×10^{-1}	$\mathbf{1.90 \times 10^{3}}$	9.25×10^{-1}	2.01×10^{3}	1.90×10^{1}	1.91×10^{3}	8.56×10^{-1}

One can see from Table 3 that the PSO outperformed the other algorithms for 9

out of the 12 evaluated benchmark objective functions. Besides that, DE obtained the best results for the benchmark objective function $F14$, while CLONAL outperformed the other algorithms for the function $F4$. Therefore, it indicates that the PSO is the best among the five evaluated algorithms in optimizing the proposed benchmark objective functions.

To assess this observation, an experimental analysis of the data was carried out following the statistical methodology of Garcia and Herrera (2008). This methodology has three steps, as described below. All steps assume a significance level $\alpha = 0.05$, i.e., the null hypothesis is rejected if a p-value smaller or equal to 0.05 is obtained.

In the first step, a Shapiro-Wilk normality test is applied to verify if the average fitness values obtained by the algorithms follow a normal distribution. The Shapiro-Wilk test found a p-value of 0.001 for all algorithms, thus indicating that the data does not follow a normal distribution. Therefore, a non-parametrical statistical test will be employed in the next steps.

The second step was to apply a Friedman's Test to verify whether the average fitness values of one of the heuristics statistically differ from that of another. The null hypothesis is that all five algorithms achieve the same average fitness value. The input data for the Friedman's test is ranked according to the methodology proposed by Carvalho (2019). The Friedman's test obtained a p-value of 0.002, thus rejecting the null hypothesis and indicating that exist a significant difference between the average fitness values of, at least, two of the algorithms evaluated in this work.

In the third step, we applied a post-hoc test for Friedman's, namely the Nemenyi's test (also known as Nemenyi–Damico–Wolfe–Dunn's test). This statistical test compares the results of multiple algorithms and evaluates the pair of hypothesis

$$\begin{cases} H_0: & \mu_i \leq \mu_j \\ H_1: & \mu_i \neq \mu_j, \end{cases} \qquad \forall (\mu_i, \mu_j) \in W,$$

whereas $W = \{\mu_{\text{GA}}, \mu_{\text{DE}}, \mu_{\text{PSO}}, \mu_{\text{ACO}}, \mu_{\text{CLONAL}}\}$, such that $\mu_{\text{GA}}, \mu_{\text{DE}}, \mu_{\text{PSO}}, \mu_{\text{ACO}}, \mu_{\text{CLONAL}}$ are respectively the average ranking obtained by GA, DE, PSO, ACO, and CLONAL in the second step. The null hypothesis (H_0) affirms that the average rankings μ_i and μ_j of all pairs of algorithms do not significantly differ from each other. Thus, the null hypothesis affirms that all algorithms have similar average rankings. On the other hand, the alternative hypothesis (H_1) suggests that the average rankings given by an algorithm μ_i statistically differ from those of another algorithm μ_j.

This post-hoc test rejected the null hypothesis and indicated that $\mu_{\text{PSO}} \leq \mu_j : j \in W \setminus \mu_{\text{PSO}}$. Thus, we can affirm that there exists a significant difference between the average rankings of PSO and all other algorithms evaluated in this work. Finally, we conclude that the PSO is the best among the evaluated heuristics in solving the proposed benchmark objective functions.

5.3 Hyperparameter optimization for IBEA

The third experiment performs the hyperparameter optimization of our IBEA with up to 50 islands. This IBEA employs one or more of the proposed algorithms as described in Section 4 with their best parameter settings, as given in Table 2.

We individually optimized the hyperparameters of every IBEA from 2 to 50 islands. Thus, for every value of $m \in \{2, 3, \ldots, 50\}$, the hyperparameters optimized were: (i) migration rate $m_r \in \{1, 10\}$; (ii) the migration size $m_s \in \{1, 50\}$; and (iii) the chance of applying the island-migration operator $im \in [0, 1]$. In every migration process, a random number v is obtained using a real uniform distribution $U(0, 1)$. If

$v \leq im$, then the island-migration operator is applied. On the other hand, if $v > im$, the neighborhood-migration operator is applied. Therefore, the greater the value of im, the greater the chance of applying the island-migration operator. Conversely, as smaller the value of im, the greater the chance of applying the neighborhood-migration operator. This experiment was performed in the same 12 objective functions employed in Section 5.2.

This hyperparameter optimization was performed considering every possible combination with repetition of the 5 algorithms described in Section 4. Therefore, for every value of m, the number of different combinations evaluated was

$$\binom{m+r-1}{r},$$

where $r = 5$ is the number of considered algorithms. For every possible combination, we optimized the parameters m_r, m_s, and im using the *Iterated Race* method available in iRace package.

Table 4 presents the results of this hyperparameter optimization experiment. For every value of m (columns 1 and 6), it presents the best value for m_r (columns 2 and 7), the best value for m_s (columns 3 and 8), and the best value for im (columns 4 and 9). Furthermore, the number of islands running each algorithm is shown in columns 5 and 10, whereas the first digit gives the number of GAs, the second digit presents the number of DEs, while the third digit denotes the number of PSOs, the fourth digit gives the number of ACOs, and the fifth digit represents the number of CLONALs.

It can be observed that, in general, PSO is the most employed algorithm in our IBEAs. Across all values of m, PSO was the chosen algorithm for 422 islands, while the GA, the DE, the ACO, and the CLONAL were respectively employed in 271, 202, 189, and 190 islands. Furthermore, the PSO was the most employed algorithm for 31 out of the 49 evaluated values of m. These results indicate that as better the algorithm in solving the proposed problem, the higher the chances of it being selected for an island. However, other algorithms are still needed, as these employ different evolutionary operators that benefit the optimization process.

Figure 3 shows the value of im for every evaluated value of m and a linear regression on this data. It can be observed that the value of im greatly varies across the experiment. In fact, the average value for im is 0.517 with a standard deviation of 0.241. Additionally, the linear regression $y = -0.002x + 0.584$ shows a decreasing tendency, but with $R^2 = 0.022$. This data suggests that the greater the number of islands, the better the neighborhood-migration operator is in optimizing the proposed benchmark objective functions.

5.4 Evaluation of IBEA with different number of islands

The fourth and last experiment determines the ideal number of islands to optimize the proposed benchmark objective functions. The results of this experiment are summarized in Table 5. The first column gives the number m of islands, while the second through thirteenth columns present the average results for the different benchmark objective functions. The results for the 12 evaluated benchmark objective functions (as displayed in the second through thirteenth columns) represent the average of 30 runs of IBEA using different seeds for the pseudo-random number generator. These numbers were computed as the average deviation from the optimal solution of these benchmark objective functions, given as

$$\frac{OPT - HEUR}{OPT},$$

Table 4: The hyperparameter optimization experiment for multiple island sizes

m	m_r	m_s	im	algorithms	m	m_r	m_s	im	algorithms
2	9	42	0.47773	1/0/1/0/0	27	9	33	0.38245	3/5/15/2/2
3	7	38	0.41116	1/0/2/0/0	28	9	43	0.92157	11/4/8/4/1
4	8	37	0.86306	1/0/2/0/1	29	9	40	0.82244	5/3/9/2/10
5	10	16	0.40838	1/0/2/0/2	30	10	36	0.65956	6/7/10/6/1
6	9	27	0.73783	1/0/2/1/2	31	9	15	0.42855	10/4/7/2/8
7	10	32	0.96086	2/0/1/2/2	32	10	20	0.33781	8/9/9/5/1
8	9	23	0.60785	2/2/3/1/0	33	9	4	0.15733	3/17/9/1/3
9	8	15	0.26441	3/1/4/1/0	34	10	39	0.60624	4/5/11/3/11
10	10	14	0.44005	3/4/2/0/1	35	9	36	0.23122	7/12/6/9/1
11	10	43	0.94423	4/0/5/1/1	36	10	29	0.78115	8/7/10/3/8
12	10	28	0.69794	2/3/4/0/3	37	10	26	0.50815	3/7/19/7/1
13	10	13	0.57998	4/4/3/0/2	38	10	21	0.435	5/1/13/18/1
14	8	36	0.32775	2/2/5/3/2	39	10	6	0.50945	8/5/12/11/3
15	9	41	0.91967	7/1/4/3/0	40	9	25	0.14614	3/12/11/8/6
16	7	24	0.86865	1/1/13/0/1	41	10	41	0.31392	15/8/9/1/8
17	9	21	0.4659	5/3/9/0/0	42	9	26	0.20789	9/1/18/6/8
18	8	25	0.28911	5/1/8/2/2	43	10	33	0.50571	2/7/13/11/10
19	10	16	0.38604	7/0/4/5/3	44	10	24	0.66713	15/5/9/12/3
20	7	22	0.22172	9/0/9/2/0	45	10	30	0.35783	14/8/14/8/1
21	9	31	0.17054	2/6/11/0/2	46	10	29	0.82661	12/0/18/2/14
22	10	39	0.31134	8/4/6/4/0	47	10	37	0.46702	3/4/16/13/11
23	9	20	0.96251	3/1/13/4/2	48	10	6	0.65432	11/9/10/5/13
24	9	11	0.1571	7/0/9/2/6	49	9	29	0.67786	13/9/16/0/11
25	10	13	0.25498	3/9/7/1/5	50	10	26	0.43544	7/11/7/8/17
26	10	31	0.57786	2/0/14/10/0					

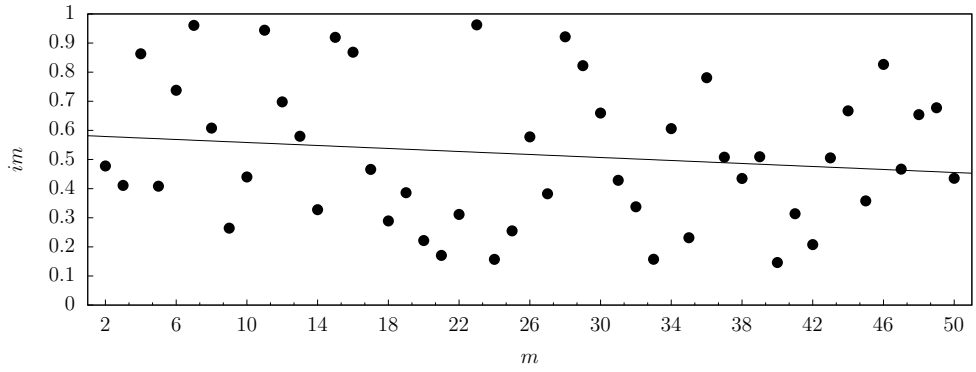


Figure 3: The value of im for every evaluated value of m

where OPT denotes the optimal value of the benchmark objective function and $HEUR$ represents the objective function value found by IBEA. Additionally, the penultimate column gives the average of the values presented in the second through thirteenth columns, and the last column denotes the standard deviation of this same range of

values.

Table 5: Average deviation from the optimal solution for IBEAs with different numbers of islands

	benchmark objective functions													
m	F1	F2	F4	F6	F7	F8	F9	F10	F11	F13	F14	F15	Average	SD
2	320.36	0.00	0.00	0.06	0.05	0.82	1.77	177.53	1.77	115.58	9.95	1.68	52.465	97.718
3	193.90	0.00	0.00	0.05	0.04	0.74	1.70	160.73	1.48	115.08	9.97	0.73	40.368	69.057
4	0.00	0.00	0.00	0.05	0.04	0.74	1.73	136.65	1.17	115.01	10.06	0.00	22.122	46.668
5	61.72	0.00	0.00	0.05	0.04	0.74	1.51	173.60	1.49	115.19	9.04	0.24	30.301	54.988
6	0.06	0.00	0.00	0.05	0.04	0.71	1.57	133.67	1.44	115.08	9.31	0.03	21.831	46.083
7	0.00	0.00	0.00	0.05	0.04	0.74	1.61	118.58	1.46	115.26	9.03	0.14	20.574	43.158
8	0.00	0.00	0.00	0.04	0.03	0.29	1.41	113.72	1.19	114.94	6.00	0.00	19.802	42.304
9	0.01	0.00	0.00	0.04	0.03	0.35	1.31	139.57	1.26	114.94	6.01	0.00	21.961	47.385
10	0.01	0.00	0.00	0.04	0.02	0.24	1.12	114.50	1.29	114.94	5.38	0.00	19.794	42.476
11	0.00	0.00	0.00	0.03	0.02	0.51	1.17	95.02	1.13	114.94	7.92	0.00	18.395	38.993
12	0.00	0.00	0.00	0.04	0.02	0.23	1.18	119.23	1.30	98.44	5.14	0.00	18.799	40.513
13	0.00	0.00	0.00	0.04	0.02	0.26	0.92	105.06	0.94	104.19	5.31	0.00	18.062	38.738
14	0.00	0.00	0.00	0.04	0.02	0.34	1.23	103.19	1.15	114.94	6.43	0.00	18.945	40.411
15	0.00	0.00	0.00	0.03	0.02	0.34	1.21	105.89	1.11	114.94	6.47	0.00	19.168	40.885
16	0.00	0.00	0.00	0.03	0.01	0.54	0.87	124.30	1.51	114.94	8.19	0.40	20.899	44.244
17	0.00	0.00	0.00	0.03	0.01	0.29	0.85	82.42	1.01	114.94	5.70	0.00	17.106	37.112
18	0.00	0.00	0.00	0.03	0.02	0.36	1.10	69.62	1.04	114.94	6.15	0.00	16.105	35.338
19	0.00	0.00	0.00	0.04	0.03	0.49	1.09	112.38	1.06	114.94	7.19	0.00	19.767	42.036
20	0.10	0.00	0.00	0.03	0.02	0.51	1.07	104.80	0.86	114.94	6.60	0.00	19.078	40.695
21	0.00	0.00	0.00	0.03	0.01	0.27	0.89	84.75	1.11	114.94	5.69	0.00	17.308	37.453
22	0.00	0.00	0.00	0.03	0.02	0.27	1.01	72.59	0.83	109.19	5.13	0.00	15.756	34.450
23	0.00	0.00	0.00	0.03	0.01	0.39	1.08	21.84	0.87	114.94	6.15	0.00	12.109	31.581
24	0.00	0.00	0.00	0.03	0.02	0.48	0.94	85.16	0.98	114.94	6.77	0.00	17.442	37.483
25	0.00	0.00	0.00	0.03	0.01	0.24	0.94	111.62	1.09	114.94	4.75	0.00	19.469	41.979
26	0.00	0.00	0.00	0.03	0.01	0.46	1.01	58.89	0.61	114.94	6.93	0.00	15.239	34.085
27	0.00	0.00	0.00	0.03	0.01	0.32	0.75	58.41	1.02	114.94	6.09	0.00	15.131	34.052
28	0.00	0.00	0.00	0.03	0.01	0.31	1.01	55.35	0.77	114.94	5.31	0.00	14.811	33.754
29	0.00	0.00	0.00	0.03	0.01	0.28	1.01	53.94	0.73	109.20	5.55	0.00	14.229	32.192
30	0.00	0.00	0.00	0.03	0.01	0.25	0.96	76.31	0.88	114.94	4.94	0.00	16.527	36.266
31	0.00	0.00	0.00	0.03	0.02	0.25	0.95	69.12	0.87	114.94	5.49	0.00	15.973	35.307
32	0.00	0.00	0.00	0.03	0.01	0.24	0.88	80.49	1.00	114.94	5.02	0.00	16.885	36.849
33	0.00	0.00	0.00	0.03	0.01	0.21	0.98	76.33	0.75	114.94	4.41	0.00	16.471	36.289
34	0.00	0.00	0.00	0.03	0.01	0.27	0.92	76.59	0.91	104.21	4.97	0.00	15.659	33.923
35	0.00	0.00	0.00	0.03	0.01	0.20	1.03	101.61	0.73	114.94	4.60	0.00	18.595	40.217
36	0.00	0.00	0.00	0.03	0.01	0.26	0.94	54.85	0.76	112.44	5.27	0.00	14.548	33.091
37	0.00	0.00	0.00	0.03	0.01	0.27	0.82	63.74	0.67	114.94	5.24	0.00	15.476	34.675
38	0.00	0.00	0.00	0.03	0.01	0.37	1.00	53.88	0.92	114.94	6.51	0.00	14.804	33.576
39	0.00	0.00	0.00	0.03	0.01	0.28	0.84	39.33	0.81	109.19	5.08	0.00	12.964	30.930
40	0.00	0.00	0.00	0.03	0.01	0.23	0.89	103.00	0.91	114.94	4.77	0.00	18.732	40.450
41	0.00	0.00	0.00	0.03	0.01	0.20	0.91	81.42	0.56	114.94	4.70	0.00	16.897	37.010
42	0.00	0.00	0.00	0.03	0.01	0.34	0.89	69.71	0.73	114.94	5.77	0.00	16.034	35.379
43	0.00	0.00	0.00	0.03	0.01	0.24	0.87	63.60	0.91	114.94	4.87	0.00	15.456	34.658
44	0.00	0.00	0.00	0.03	0.01	0.26	0.89	55.37	0.51	104.19	4.98	0.00	13.854	31.151
45	0.00	0.00	0.00	0.03	0.01	0.23	0.96	52.55	0.44	104.19	0.76	0.00	13.264	30.967
46	0.00	0.00	0.00	0.02	0.01	0.37	0.77	14.68	0.82	114.94	5.83	0.00	11.453	31.474
47	0.00	0.00	0.00	0.03	0.01	0.27	0.85	69.33	0.82	114.94	5.54	0.00	15.982	35.337
48	0.00	0.00	0.00	0.03	0.01	0.25	0.93	81.63	0.62	114.94	4.59	0.00	16.915	37.039
49	0.00	0.00	0.00	0.03	0.01	0.25	0.89	38.86	0.71	82.70	4.83	0.00	10.689	24.156
50	0.00	0.00	0.00	0.03	0.01	0.20	0.92	63.98	0.79	104.19	4.44	0.00	14.547	32.186

The data in Table 5 shows that the hardest benchmark objective functions to be optimized were $F10$ and $F13$, while $F2$ and $F4$ were the easiest ones. The IBEAs tend to obtain better results when the number of islands m increases. In fact, one can

construct a linear regression on the data presented in column *Average* of this table as $y = -0.296x + 26.059$, which demonstrates such a tendency. The smallest average result from optimal solutions was found for $m = 49$. Furthermore, for $m = 49$, we also obtained the smallest standard deviation of this same data. Despite that, the IBEA with 49 islands was able to overcome all other IBEA implementations for only 2 out of the 12 evaluated benchmark objective functions.

To compare all of our IBEA implementations, we performed another statistical evaluation on the data reported in Table 5. This statistical evaluation followed the same statistical procedure reported in Section 5.2. The results are as follows.

In the first step, a Shapiro-Wilk test found p-values smaller than 0.05 for all values of m. Thus, it indicated that parametrical statistical tests could not be used. Hence, the next steps will employ non-parametrical statistical tests.

In the second step, the null hypothesis of Friedman's test is that all IBEA implementations with different numbers of islands achieve the same average deviation from the optimal solution. The input for the Friedman's test was first preprocessed using the methodology of Carvalho (2019). It was observed that the smallest average ranking given by the preprocessing methodology of Carvalho (2019) was for the IBEA with $m = 49$, thus suggesting that this was the best IBEA implementation. The Friedman's test obtained a p-value of 0.003. Therefore, it rejected the null hypothesis and confirmed that there exists a significant difference between the average deviation from the optimal solution of two or more of the IBEAs contrasted in this experiment.

In the third step, we applied the post-hoc Nemenyi's test. This statistical test compares the results of multiple algorithms and evaluates the pair of hypothesis

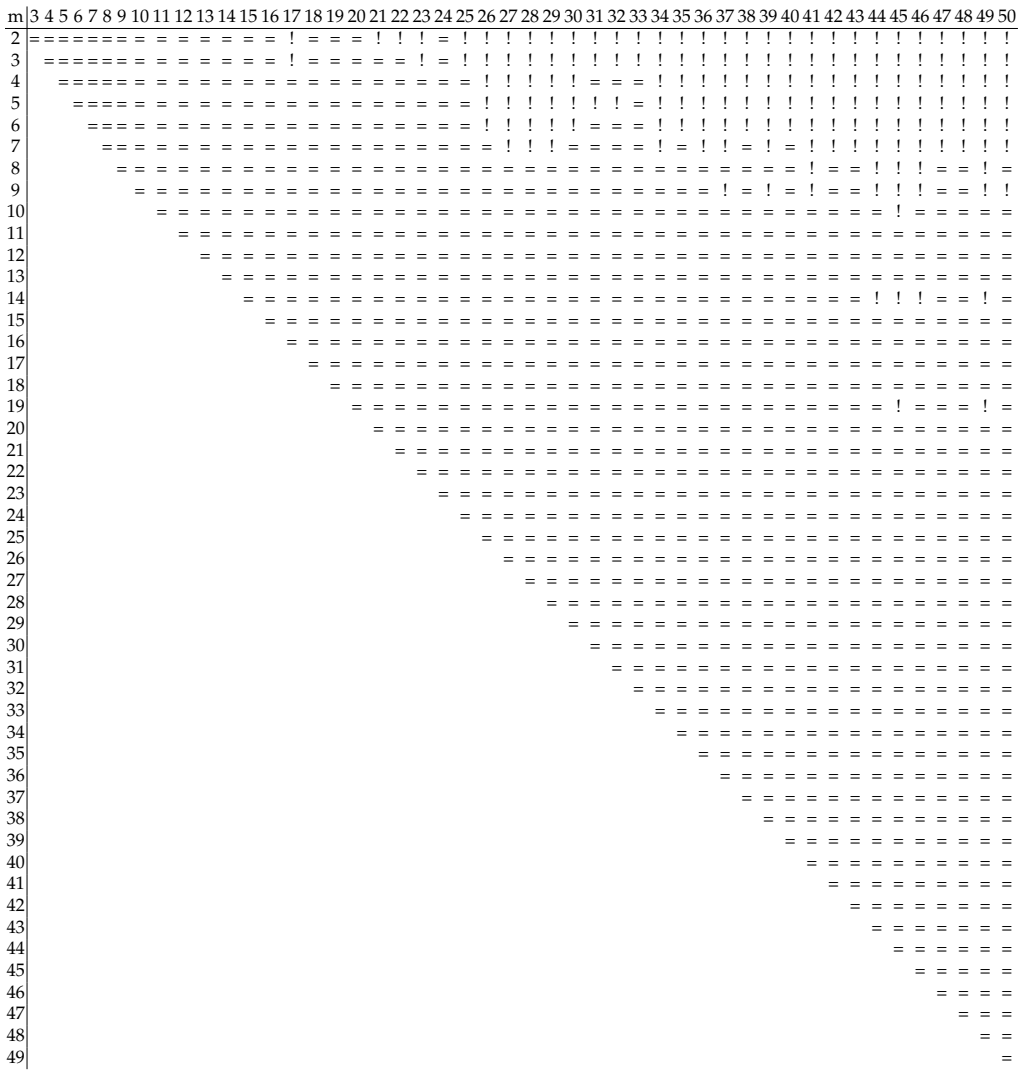
$$\begin{cases} H_0: & \varrho_i \leq \varrho_j \\ H_1: & \varrho_i \neq \varrho_j, \end{cases} \qquad \forall(\varrho_i, \varrho_j) \in R,$$

whereas $R = \{\varrho_2, \varrho_3, \ldots, \varrho_{50}\}$, such that ϱ_i corresponds to the average ranking obtained by the IBEA with $m = i$. The null hypothesis (H_0) affirms that the average rankings ϱ_i and ϱ_j of all pairs of IBEA implementations do not significantly differ from each other. Thus, the null hypothesis affirms that all algorithms have similar average rankings. On the other hand, the alternative hypothesis (H_1) suggests that the average rankings given by an IBEA implementation ϱ_i statistically differ from those of another implementation ϱ_j.

The Nemenyi's test rejected the null hypothesis. Thus, we can affirm that there exists a significant difference between the average rankings of an IBEA implementation and the others. The results of the Nemenyi's test are presented in Table 6. In that table, the first row and the first column denote the number of islands (m). Every other cell indicates whether the average ranking obtained by the IBEA implementation with x islands differs (represented by the ! symbol) or not (represented by the $=$ symbol) from that of the IBEA implementation with y islands.

The data reported in Table 6 indicates a significant difference between many of the evaluated IBEA implementations, but most of them are statistically similar. In special, we would like to highlight that the IBEA implementation with $m = 49$ (that has the smallest average deviation from the optimal solution and the smallest average ranking provided by the methodology of Carvalho (2019)) does not statistically differ from the IBEA implementations with $m \in \{10, \ldots, 13, 15, \ldots, 18, 20, \ldots, 50\}$.

Table 6: Results for the Nemenyi's test



6 Conclusions

This paper evaluated a co-evolutionary framework for developing evolutionary algorithms based on the concept of islands. An evolutionary algorithm developed through this framework is denoted as an island-based evolutionary algorithm (IBEA). In this framework, the algorithm's population is split into several islands that evolve semi-isolated but can interact using a migration operator.

We performed the most extensive and robust evaluation of this framework in the literature. We implemented and contrasted IBEAs with up to 50 islands, employing 2 migration operators, and using up to 5 different evolutionary algorithms: a genetic algorithm (GA), a differential evolution algorithm (DE), a particle swarm optimization algorithm (PSO), an ant colony optimization algorithm (ACO), and a clonal selection algorithm (CLONAL).

We can conclude that using the co-evolutionary islands model for developing evolutionary algorithms is beneficial in optimizing the proposed benchmark objective functions. Despite not being statistically superior to the others, the IBEA implementation with $m = 49$ found the best results among the evaluated algorithms, as a statistical analysis of our experiments pointed out that using 8, 10, or more islands leads to similar results if the model is properly tuned and optimized. Thus, we recommend the use of IBEAs with more than 10 islands when optimizing the proposed benchmark objective functions.

Furthermore, our experiments also demonstrated that combining several algorithms produces better results for the co-evolutionary islands model than using a single algorithm in all islands, as pointed out in Table 4. These different algorithms have mechanisms capable of exploring the solution space differently from each other, and the combination of all of these mechanisms can avoid premature convergence and aid in escaping from local optima.

We hope that our paper was able to establish some experimental ground on this distributed framework. We expect other researchers and practitioners to employ our data (Gouveia (2024)) and experiments to propose their island-based evolutionary algorithms. Future work may evaluate IBEA algorithms in solving combinatorial, linear, and non-linear optimization problems.

Acknowledgement

This study was financed in part by the *Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil* (CAPES) - Finance Code 001, the *Conselho Nacional de Desenvolvimento Científico e Tecnológico - Brasil* (CNPq), and the *Fundação de Amparo à Pesquisa do Estado de Minas Gerais - Brasil* (FAPEMIG).

References

Alba, E. (2005). *Parallel metaheuristics: a new class of algorithms*, volume 47. John Wiley & Sons.

Alba, E. and Luque, G. (2005). Theoretical models of selection pressure for deas: topology influence. In *2005 IEEE Congress on Evolutionary Computation*, volume 1, pages 214–221. IEEE.

Alba, E. and Troya, J. M. (2000). Influence of the migration policy in parallel distributed gas with structured and panmictic populations. *Applied Intelligence*, 12(3):163–181.

Araujo, L. and Merelo, J. J. (2011). Diversity through multiculturality: Assessing migrant choice policies in an island model. *IEEE Transactions on Evolutionary Computation*, 15(4):456–469.

Bilchev, G. and Parmee, I. C. (1995). The ant colony metaphor for searching continuous design spaces. In *Evolutionary Computing: AISB Workshop Sheffield, UK, April 3–4, 1995 Selected Papers*, pages 25–39. Springer.

Cantú-Paz, E. (2001). Migration policies, selection pressure, and parallel evolutionary algorithms. *Journal of heuristics*, 7(4):311–334.

Cantú-Paz, E. and Goldberg, D. E. (2003). Are multiple runs of genetic algorithms better than one? In *Genetic and Evolutionary Computation—GECCO 2003*, pages 801–812. Springer.

Carvalho, I. A. (2019). On the statistical evaluation of algorithmic's computational experimentation with infeasible solutions. *Information Processing Letters*, 143:24–27.

Carvalho, I. A. and Ribeiro, M. A. (2019). A node-depth phylogenetic-based artificial immune system for multi-objective network design problems. *Swarm and Evolutionary Computation*, 50:100491.

Da Silveira, L. A., de Lima, T. A., de Barros, J. B., Soncco-Álvarez, J. L., Llanos, C. H., and Ayala-Rincón, M. (2023). On the behavior of parallel island models. *Applied Soft Computing*, 148:110880.

Da Silveira, L. A., Soncco-Álvarez, J. L., de Lima, T. A., and Ayala-Rincón, M. (2019). Parallel island model genetic algorithms applied in np-hard problems. In *2019 IEEE Congress on Evolutionary Computation (CEC)*, pages 3262–3269. IEEE.

Da Silveira, L. A., Soncco-Alvarez, J. L., De Lima, T. A., and Ayala-Rincon, M. (2021). Heterogeneous parallel island models. In *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8. IEEE.

De Castro, L. N. and Von Zuben, F. J. (2000). The clonal selection algorithm with engineering applications. In *Proceedings of GECCO*, volume 2000, pages 36–39.

Dorigo, M. and Stützle, T. (2019). *Ant colony optimization: overview and recent advances*. Springer.

Duarte, G. R., de Castro Lemonge, A. C., da Fonseca, L. G., and de Lima, B. S. L. P. (2021). An island model based on stigmergy to solve optimization problems. *Natural Computing*, 20(3):413–441.

Duarte, G. R., de Lima, B. S. L. P., and de Castro Lemonge, A. C. (2022). A stigmergy-based island model for dynamic evaluation of constraint-handling techniques and differential evolution algorithms. *IEEE Transactions on Evolutionary Computation*, 27(3):701–715.

Garcia, S. and Herrera, F. (2008). An extension on"statistical comparisons of classifiers over multiple data sets"for all pairwise comparisons. *Journal of Machine Learning Research*, 9(Dec):2677–2694.

Gouveia, G. A. (2024). Heuristic scientific research on github. Accessed: 2024-07-14.

Holland, J. H. et al. (1992). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press.

Jedrzejowicz, P. and Skakovski, A. (2016). Improving performance of the differential evolution algorithm using cyclic decloning and changeable population size. *J. Univers. Comput. Sci.*, 22(6):874–893.

Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, volume 4, pages 1942–1948. IEEE.

Lässig, J. and Sudholt, D. (2013). Design and analysis of migration in parallel evolutionary algorithms. *Soft Computing*, 17(7):1121–1144.

Lässig, J. and Sudholt, D. (2014). General upper bounds on the runtime of parallel evolutionary algorithms. *Evolutionary computation*, 22(3):405–437.

Liang, J., Qu, B., Suganthan, P., and Chen, Q. (2014). Problem definitions and evaluation criteria for the cec 2015 competition on learning-based real-parameter single objective optimization. Technical Report Technical Report201411A, Zhengzhou University and Nayang Technologial University.

López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L. P., Birattari, M., and Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58.

Luque, G. and Alba, E. (2011). *Parallel genetic algorithms: theory and real world applications*, volume 367. Springer.

Magalhães, T. T., Krempser, E., and Barbosa, H. J. C. (2015). Migration policies to improve exploration in parallel islands model for optimization via metaheuristics. In *Proceedings of the XXXVI Ibero-Latin American Congress on Computational Methods in Engineering*, page 20p. Brazilian Association of Computational Methods in Engineering.

Mambrini, A. and Sudholt, D. (2015). Design and analysis of schemes for adapting migration intervals in parallel evolutionary algorithms. *Evolutionary computation*, 23(4):559–582.

Martin, W. N., Lienig, J., and Cohoon, J. P. (1997). Island (migration) models: evolutionary algorithms based on punctuated equilibria. In Baeck, T., Fogel, D., and Michalewicz, Z., editors, *Handbook of Evolutionary Computation*, pages 387–402. Taylor & Francis.

Matsumoto, M. and Nishimura, T. (1998). Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):3–30.

Osorio, K., Alba, E., and Luque, G. (2013). Using theory to self-tune migration periods in distributed genetic algorithms. In *2013 IEEE Congress on Evolutionary Computation*, pages 2595–2601. IEEE.

Osorio, K., Luque, G., and Alba, E. (2011). Distributed evolutionary algorithms with adaptive migration period. In *Intelligent Systems Design and Applications (ISDA), 2011 11th International Conference on*, pages 259–264. IEEE.

Reeves, C. R. (2010). Genetic algorithms. In *Handbook of metaheuristics*, pages 109–139. Springer.

Ruciński, M., Izzo, D., and Biscani, F. (2010). On the impact of the migration topology on the island model. *Parallel Computing*, 36(10):555–571.

Schwehm, M. (1996). Parallel population models for genetic algorithms. *Universität Erlangen-Nürnberg*, pages 2–8.

Sekaj, I. (2004). Robust parallel genetic algorithms with re-initialisation. In *Parallel Problem Solving from Nature-PPSN VIII*, pages 411–419. Springer.

Silva, J. G. R., Carvalho, I. A., Goliatt, L., da Fonseca Vieira, V., and Xavier, C. R. (2017). A differential evolution algorithm for computing caloric-restricted diets - island-based model. In Gervasi, O., Murgante, B., Misra, S., Borruso, G., Torre, C. M., Rocha, A. M. A., Taniar, D., Apduhan, B. O., Stankova, E., and Cuzzocrea, A., editors, *Computational Science and Its Applications – ICCSA 2017*, pages 385–400, Cham. Springer International Publishing.

Skakovski, A. and Jedrzejowicz, P. (2019). An island-based differential evolution algorithm with the multi-size populations. *Expert Systems with Applications*, 126:308–320.

Skakovski, A. and Jedrzejowicz, P. (2022). A multisize no migration island-based differential evolution algorithm with removal of ineffective islands. *IEEE Access*, 10:34539–34549.

Skolicki, Z. and De Jong, K. (2005). The influence of migration sizes and intervals on island models. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation*, pages 1295–1302. ACM.

Socha, K. and Dorigo, M. (2008). Ant colony optimization for continuous domains. *European journal of operational research*, 185(3):1155–1173.

Storn, R. and Price, K. (1997). Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341.

Tomassini, M. (2005). *Spatially Structured Evolutionary Algorithms: Artificial Evolution in Space and Time*. Springer.

Vargas, D. V., Murata, J., Takano, H., and Delbem, A. C. B. (2015). General subpopulation framework and taming the conflict inside populations. *Evolutionary computation*, 23(1):1–36.

Wang, D., Tan, D., and Liu, L. (2018). Particle swarm optimization algorithm: an overview. *Soft computing*, 22:387–408.

Whitley, D., Rana, S., and Heckendorn, R. B. (1998). The island model genetic algorithm: On separability, population size and convergence. *Journal of Computing and Information Technology*, 7:33–48.

Xavier, C. R., Silva, J. G. R., Duarte, G. R., Carvalho, I. A., Vieira, V. d. F., and Goliatt, L. (2023). An island-based hybrid evolutionary algorithm for caloric-restricted diets. *Evolutionary Intelligence*, 16(2):553–564.