stats

MDPI

Article

An Improved Soft Island Model of the Fish School Search Algorithm with Exponential Step Decay Using Cluster-Based Population Initialization

Liliya A. Demidova * and Vladimir E. Zhuravlev *

Institute for Information Technologies, Federal State Budget Educational Institution of Higher Education, MIREA—Russian Technological University, 78, Vernadsky Avenue, 119454 Moscow, Russia
* Correspondence: liliya.demidova@rambler.ru (L.A.D.); vovcrane@mail.ru (V.E.Z.)

Abstract: Optimization is a highly relevant area of research due to its widespread applications. The development of new optimization algorithms or the improvement of existing ones enhances the efficiency of various fields of activity. In this paper, an improved Soft Island Model (SIM) is considered for the Tent-map-based Fish School Search algorithm with Exponential step decay (ETFSS). The proposed model is based on a probabilistic approach to realize the migration process relying on the statistics of the overall achievement of each island. In order to generate the initial population of the algorithm, a new initialization method is proposed in which all islands are formed in separate regions of the search space, thus forming clusters. For the presented SIM-ETFSS algorithm, numerical experiments with the optimization of classical test functions, as well as checks for the presence of some known defects that lead to undesirable effects in problem solving, have been carried out. Tools, such as the Mann–Whitney U test, box plots and other statistical methods of data analysis, are used to evaluate the quality of the presented algorithm, using which the superiority of SIM-ETFSS over its original version is demonstrated. The results obtained are analyzed and discussed.

Keywords: evolutionary optimization; swarm intelligence; Fish School Search; Soft Island Model; center-bias operator; unevenness defect; Mann–Whitney U test

Academic Editor: Wei Zhu

Received: 25 December 2024
Revised: 16 January 2025
Accepted: 20 January 2025
Published: 22 January 2025

Citation: Demidova, L.A.; Zhuravlev, V.E. An Improved Soft Island Model of the Fish School Search Algorithm with Exponential Step Decay Using Cluster-Based Population Initialization. *Stats* **2025**, *8*, 10. https://doi.org/10.3390/stats8010010

1. Introduction

Despite the rapid development of computing technology, the growth of its performance has not always kept pace with the ever-accelerating increase in the volume of data to be processed and analyzed everywhere. To reduce the existing gap, algorithms and methods are required that can not only cope with huge volumes of data but also take into account their complexity, variability, incompleteness and vagueness. Decision making under such conditions becomes a real challenge for classical approaches to data processing.

For example, the problem of pattern recognition was initially considered as a mathematical problem, where the object is described by a set of features, and the methods of analysis are reduced to strict computations. The approaches used for this purpose, such as the k-nearest neighbors method (kNN) [1] and histograms of oriented gradients (HOG) [2], although they showed high efficiency in simple tasks of recognizing different objects and shapes in images, were gradually replaced by more flexible models of neural networks, for which the previously mentioned continuous increase in the amount of information was even useful, positively affecting the amount of training data. Such intelligent methods for analyzing and processing information have recently even surpassed average human

capabilities. The best model for image classification for the ImageNet dataset, presented back in 2014, outperformed the average human results in terms of accuracy [3]. Currently, intelligent algorithms have many applications in such fields as medicine [4,5], economics [6], social sciences [7], natural sciences [8–10] and many others.

One of the key tasks in data processing and analysis is optimization, the relevance of which is confirmed by the variety of its applications. The need for optimization arises, for example, in the course of genetic research [11], route planning in complex transportation networks [12], managing connections to a database [13], tuning hyperparameters of machine learning models [14–16], directly in the learning process itself [17,18] and in many other areas. The solution of an optimization problem in the general case is to find extreme (minimum or maximum) values of some arbitrary objective function. From the mathematical point of view, this problem consists of finding such a vector of parameters $\overline{x}^* \in \mathbb{R}^D$ that $f(\overline{x}^*) = \min\limits_{\overline{x} \in \Omega} f(\overline{x})$ or $f(\overline{x}^*) = \max\limits_{\overline{x} \in \Omega} f(\overline{x})$, where $f : \mathbb{R}^D \to \mathbb{R}$ is the function to be optimized; D is the number of parameters of the function (dimension of the search space); Ω is the region of admissible values that defines the search space.

As it was mentioned earlier, traditional mathematical optimization methods, such as algorithms based on Newton's method [19] or variations of gradient descent [20], are not flexible enough in the context of ever-increasing data complexity, structure and volume; therefore, heuristic optimization algorithms [21] based on various heuristics that do not impose any additional requirements on the objective function, except for the possibility of its calculation, are gaining popularity. An important feature of such algorithms is their ability to work under conditions of the uncertainty and incompleteness of data, which makes them a versatile and flexible tool for solving complex practical problems. However, population-based algorithms also have limitations. For example, in [22], the authors discuss the lack of rigorous mathematical proofs or guarantees of convergence of such algorithms. In order to resolve this problem, they propose a framework that combines population-based algorithms with deterministic algorithms for which convergence has been proven. However, the hybrid algorithms obtained in this way become more computationally complex compared to their original versions.

Despite the fact that some heuristic optimization algorithms can cope with specific problems or classes of problems much better than other algorithms, one cannot unequivocally make conclusions about their exclusive superiority. It is known that the performance of an algorithm evaluated in finding an optimal solution, averaged over all possible problems, does not depend on the specifics and principles of the optimization algorithm, so it is equal for any two algorithms. This statement is a consequence of the so-called "No Free Lunch" theorem proven in 1996 [23]. Thus, there cannot exist any one universal algorithm that demonstrates superiority over all other algorithms in solving absolutely every problem. For this reason, in order to solve real practical problems, it always makes sense to consider and apply different optimization algorithms, analyze the results obtained by using them and make conclusions about which particular algorithm is best suited in a given case. For example, ref. [24] proposes a four-stage approach for analyzing different heuristic algorithms to identify the most suitable one for solving a specific problem. The proposed approach suggests first testing the algorithms for the presence of defects, such as the center-bias operator [25], then evaluating their accuracy and speed in solving test problems and finally excluding algorithms that do not belong to the Pareto front based on the analysis results.

On the other hand, modern heuristic algorithms are often hybrid, combining several fundamentally different optimization methods chosen "on the fly" from assumptions about the terrain of the function to be optimized, made based on the analysis of past iterations [26].

However, even if there is a classifier that perfectly accurately determines the type of problem to be solved and the most suitable optimization algorithm for it in the course of its work, it is not guaranteed that among the set of heuristics that this hybrid algorithm possesses, there will necessarily be one for which the results cannot be surpassed by any other algorithm, since for this purpose, the hybrid algorithm must include an infinite set of all possible optimization algorithms. Therefore, the study and improvement of already existing heuristic algorithms still remains an urgent task, even despite the seemingly "losing" position of some algorithms in relation to others, expressed by the results of their work on known optimization problems. The ideas and principles obtained in the course of studying the possibilities of the development and modification of such algorithms can be useful, among other things, for creating better hybrid approaches.

This paper proposes an island modification of the Tent-map-based Fish School Search algorithm with Exponential step decay (ETFSS) [27], proposed in co-authorship by one of the authors of this paper. As the name suggests, this algorithm is based on the Fish School Search (FSS) algorithm, which is inspired by the behavior of a school of fish in an aquarium and belongs to the population-based optimization algorithms, a subclass of a broader category of heuristic algorithms. The Soft Island Model (SIM) [28] is taken as the basis of the island model in this paper, but it has undergone extensive modifications. Specifically, we enhanced the probabilistic approach inherent to SIM by incorporating statistics from each individual island, and also, we excluded situations of the complete disappearance of islands by implementing a new hyperparameter controlling the minimum number of fish (agents) that should remain on each island during each iteration of the algorithm. Additionally, we propose a new approach to population initialization involving the cluster-based generation of agents relative to their islands.

To confirm statistically significant differences in either direction between the results obtained by different algorithms during their comparison, we use the Mann–Whitney U test. The choice of this test is justified by the fact that the samples being compared are independent, and their distributions do not necessarily follow a normal distribution. Additionally, the application of the Mann–Whitney U test is widely used in the context of population-based optimization algorithm comparisons [29–31].

The rest of the paper is organized as follows. Section 2 is devoted to the review of works related to the current study. Section 3 describes all key aspects of the research, including a detailed description of the Tent-map Based Fish School Search algorithm with Exponential step decay in Section 3.1, an improved Soft Island Model in Section 3.2, a new approach to cluster-based population initialization in Section 3.3, a description of known defects in population algorithms, such as the center-bias operator (CBO) [25] and the unevenness defect (UD) [32] and how to detect them in Section 3.4, and details and mathematical definitions for all test functions used in the experiments in Section 3.5. Section 4 presents the results of the conducted experiments, including analyzing the performance of the proposed algorithm and comparing it with the original island-free version of ETFSS under different values of hyperparameters in Section 4.1, testing the algorithm for the presence of the aforementioned defects in Section 4.2 and demonstrating an example of using the proposed algorithm on a real data problem in Section 4.3. Section 5 summarizes the results of the study and suggests directions of future research.

2. Related Work

In [33–40], such classical population-based optimization algorithms as Artificial Bee Colony (ABC), Culture Algorithm (CA), Differential Evolution (DE), Genetic Algorithm (GA), Hill Climbing (HC), Memetic Algorithm (MA), Particle Swarm Optimization (PSO) and Ant Lion Optimizer (ALO) are presented. Despite some of these algorithms being com-

paratively old, the heuristics implemented in them are still used both in other algorithms and in the modernizations of their basic versions. For example, the DE algorithm was first presented back in 1997, but its numerous modifications are still used for solving complex optimization problems [29].

In [41], the original Fish School Search (FSS) algorithm is presented, which is a basic version of ETFSS, an island modification of which is the subject of this paper. The heuristics used in the FSS algorithm to optimize the objective function are borrowed from the behavior of real fish that join groups (schools) in order to increase their chances of survival. Driven by swarm intelligence, a school of fish is able to search for food sources and avoid predators much more efficiently than each fish individually.

In [42,43], the application of the ETFSS algorithm is considered both for solving classical optimization test problems and for selecting weights for the hidden layer of the Extreme Learning Machine (ELM) model. In both cases, the performance of ETFSS is compared to other well-known algorithms, such as the Genetic Algorithm (GA) [36] and Particle Swarm Optimization (PSO) [39], as well as its original version, Fish School Search (FSS) [41]. As a result of the experiments, the authors draw conclusions not only about the superiority of ETFSS over other analogs in the context of solving the problems considered in the paper but also about the feasibility of its application to real practical problems.

In [44], a hybrid island model inspired by the natural phenomenon of stigmergy is presented. Natural stigmergy is a mechanism of a spontaneous indirect interaction between individuals, which consists of leaving marks in the environment that stimulate further activity of other individuals. By means of stimergy, individual agents of populations self-organize and maintain a certain level of cooperation through indirect communication. The Stgm-IM model proposed by the authors was tested in relation to different popular optimization algorithms, and as a result, the authors recorded statistically significant superiority of the island model over the basic versions of the algorithms.

In [45], the optimization problem of an augmented kinetic turbine with a sectioned diffuser parameterized with thirty degrees of freedom was considered. Since this problem is computationally complex, for its solution, the authors propose to use an island model capable of parallelizing the main optimization algorithm. As a result of the work, it is concluded that the use of the island model not only helps to significantly accelerate computation but also leads to better results.

In [46], different coevolutionary optimization scenarios were considered and analyzed. The authors particularly emphasized the "self-play" scenario, in which several "players" compete with each other in a "game" in order to determine one champion who "plays" better than the others. In the context of the coevolutionary approach to optimization, each "player" represents a particular algorithm, and "self-play" refers to the solution of an optimization problem of some objective function. By competing with each other, a winner is identified among the algorithms, which can be considered the most suitable for solving a particular problem. The authors have proven that the basic principles of the "No Free Lunch" theorem are inapplicable for such a coevolutionary scenario, since there exist such hybrid approaches that realize the mechanism of selecting specific algorithms under changing conditions, and selected algorithms show higher performance compared to other algorithms on average for all possible problems. This fact justifies the use of island models and only increases the relevance of research in this direction.

In [47,48], modern approaches to the implementation of co-evolutionary modifications for existing optimization algorithms are presented using Phasor Particle Swarm Optimization (PPSO) [49] and Brain Storm Optimization (BSO) [50] as examples. As a result, both works achieve much better results due to the integration of the co-evolution model into the problem-solving process by classical algorithms.

In [30,51,52], different ways to adapt the parameters of the Differential Evolution (DE) algorithm [35] directly in the process of solving an optimization problem are considered. Since this algorithm is very sensitive to the values of hyperparameters, its behavior can be flexibly controlled depending on the available information about the terrain of the objective function obtained during previous iterations. To analyze and incorporate this information, the authors apply different mathematical and statistical models, including Taylor series and Cauchy and Student's t-distribution. The resulting self-adaptive algorithms show qualitative results on well-known benchmarks, such as CEC 2017 [53] and CEC 2022 [54].

3. Materials and Methods

3.1. The Tent-Map-Based Fish School Search Algorithm with Exponential Step Decay

As already mentioned, the ETFSS algorithm is a modification of the Fish School Search (FSS) algorithm. The idea of the original algorithm is to simulate the behavior of fish swimming around the aquarium and interacting with each other in search of as much food as possible. In the context of the optimization problem, a school of M such fish represents a population of M points (agents) in the feature space $\mathbb{R}^D$, a search area represents the aquarium and the amount of food is determined by the value of the objective (optimizable) function f. Each iteration of the algorithm consists of four main stages: individual movement, feeding, collective instinctive movement and collective volitional movement.

3.1.1. Individual Movement Stage

Each agent takes a random step in a random direction, the maximum length of which is set as a hyperparameter. If the value of the objective function has not improved after the move, the agent returns to the initial position. Thus, at the t-th iteration of the algorithm, random moves $\Delta\overline{x}_i^{(t)}$ for the i-th agent $\overline{x}_i$ are calculated as follows:

$$\Delta\overline{x}_i^{(t)} = step_{ind}^{(t)} \cdot \overline{rand}, \tag{1}$$

where $\overline{rand}$ denotes a vector of random numbers drawn from $[0,\ 1)$; the dimensionality of $\overline{rand}$ is the same as the dimensionality of the agent $\overline{x}_i$; $step_{ind}^{(t)}$ denotes the maximum step length for an individual movement during the t-th iteration; $i = \overline{1, M}$.

Before the movement is directly implemented, the differences of the objective function $\Delta f_i^{(t)}$ between the agents' initial positions $\overline{x}_i^{(t)}$ and the agents' potential positions $\overline{x}_i^{(t)} + \Delta\overline{x}_i^{(t)}$ after the individual movement has not yet been implemented are calculated:

$$\Delta f_i^{(t)} = f\left[\overline{x}_i^{(t)}\right] - f\left[\overline{x}_i^{(t)} + \Delta\overline{x}_i^{(t)}\right], \tag{2}$$

where $i = \overline{1, M}$; square brackets denote the calculation of the function.

A relocation is performed only if it leads to an improvement of the objective function value in the context of solving the minimization problem:

$$\overline{x}_{i,ind}^{(t+1)} = \begin{cases} \overline{x}_i^{(t)} + \Delta\overline{x}_i^{(t)}, & \Delta f_i^{(t)} > 0 \\ \overline{x}_i^{(t)}, & \Delta f_i^{(t)} \leq 0 \end{cases}, \tag{3}$$

where $\overline{x}_{i,ind}^{(t+1)}$ denotes the position of the i-th agent after making an individual movement; $i = \overline{1, M}$.

3.1.2. Feeding Stage

Depending on how much the value of the objective function has improved or deteriorated during the random moves performed at the first stage, the weight w_i of each i-th agent changes accordingly. It is obvious that the weights will increase only for those agents that changed their position during the individual movement, since the movement was performed only if the value of the objective function improved. The weights of the other agents remaining in their places will decrease.

The weight of the agents is bounded: it can only take values from the range $[1,\ w_{max}]$, where w_{max} is one of the hyperparameters of the algorithm. At initialization, each agent is assigned a weight of $0.5 \cdot w_{max}$. The weight of the i-th agent at the t-th iteration of the algorithm changes as follows:

$$w_i^{(t+1)} = w_i^{(t)} + \frac{\Delta f_i^{(t)}}{\max\limits_j \Delta f_j^{(t)}}, \tag{4}$$

where $w_i^{(t)}$ and $w_i^{(t+1)}$ denote the weight of the i-th agent at the t-th and $(t+1)$-th iterations of the algorithm, respectively; $i = \overline{1, M}$; $j = \overline{1, M}$; the weight $w_i^{(t+1)}$ after calculating Formula (4) is limited to the range $[1,\ w_{max}]$.

According to the formulas above, if the i-th agent remained in place during the individual movement, it means that $\Delta f_i^{(t)} \leq 0$, which in turn leads to a negative value in the numerator of the fraction of Formula (4) and hence a decrease in the weight of the agent.

After the feeding stage, new differences in the values of the objective function $\Delta f_{i,ind}^{(t)}$ are calculated according to the individual movement already performed:

$$\Delta f_{i,ind}^{(t)} = \begin{cases} \Delta f_i^{(t)}, & \Delta f_i^{(t)} > 0 \\ 0, & \Delta f_i^{(t)} \leq 0 \end{cases}. \tag{5}$$

where $i = \overline{1, M}$.

3.1.3. Collective-Instinctive Movement Stage

The entire population moves along the average direction of improvement of the objective function value calculated during individual movement. The position of each i-th agent $\overline{x}_i$ at the t-th iteration of the algorithm during the collective-instinctive movement $\overline{x}_{i,ins}^{(t+1)}$ is specified as follows:

$$\overline{x}_{i,ins}^{(t+1)} = \overline{x}_{i,ind}^{(t+1)} + \frac{\sum_{j=1}^{M} \Delta \overline{x}_j^{(t)} \cdot \Delta f_{j,ind}^{(t)}}{\sum_{j=1}^{M} \Delta f_{j,ind}^{(t)}}, \tag{6}$$

where M is the number of agents in the population; the second term in the right part of the equation (fraction) is a constant equal for each agent at the t-th iteration of the algorithm; $i = \overline{1, M}$; $j = \overline{1, M}$.

3.1.4. Collective-Volitive Movement Stage

In order to maintain the balance between exploration and exploitation, all agents move either towards the common barycenter $\overline{B}^{(t)}$ at the current t-th iteration or in the opposite direction, depending on whether the total weight of all agents has increased or decreased during the previous stages. Each agent takes its own random step in this direction, the maximum length of which is the second hyperparameter of the algorithm.

The common barycenter of agents is calculated as follows:

$$\overline{B}^{(t)} = \frac{\sum_{j=1}^{M} w_j^{(t+1)} \cdot \overline{x}_{j,ins}^{(t+1)}}{\sum_{j=1}^{M} w_j^{(t+1)}}. \tag{7}$$

Then, Formula (8) is applied to calculate the agents' positions after the collective-volitive movement $\overline{x}_{i,vol}^{(t+1)}$ if $\sum_{j=1}^{M} w_j^{(t+1)} > \sum_{j=1}^{M} w_j^{(t)}$; otherwise, Formula (9) is applied:

$$\overline{x}_{i,vol}^{(t+1)} = \overline{x}_{i,ins}^{(t+1)} - step_{vol}^{(t)} \cdot \overline{rand} \cdot \frac{\overline{x}_{i,ins}^{(t+1)} - \overline{B}^{(t)}}{\left\| \overline{x}_{i,ins}^{(t+1)} - \overline{B}^{(t)} \right\|}, \tag{8}$$

$$\overline{x}_{i,vol}^{(t+1)} = \overline{x}_{i,ins}^{(t+1)} + step_{vol}^{(t)} \cdot \overline{rand} \cdot \frac{\overline{x}_{i,ins}^{(t+1)} - \overline{B}^{(t)}}{\left\| \overline{x}_{i,ins}^{(t+1)} - \overline{B}^{(t)} \right\|}, \tag{9}$$

where $\overline{rand}$ denotes a vector of random numbers drawn from $[0,\ 1)$; the dimensionality of $\overline{rand}$ is the same as the dimensionality of the agent $\overline{x}_i$; $step_{vol}^{(t)}$ denotes the maximum step length for the collective-volitive movement during the t-th iteration; $i = \overline{1, M}$; double vertical lines $\|\cdot\|$ denote the norm of the vector.

After all of the described stages, each agent moves along the coordinates of the realized collective-volitive movement, i.e., $\overline{x}_i^{(t+1)} = \overline{x}_{i,vol}^{(t+1)}$, and the step lengths for individual and collective-volitive movement decay according to a predetermined linear law.

Figure 1 shows an example of population development during four main stages in the optimization of a function of two variables $f(x_1, x_2)$. White dots with a blue contour on the graphs denote agents, the same blue color represents the traces of their movement during the stage and green and red contours denote agents that increased and decreased their weight during the feeding stage, respectively. The background color in the subfigures represents the landscape of the objective function, ranging from lower values (blue) to higher values (red).

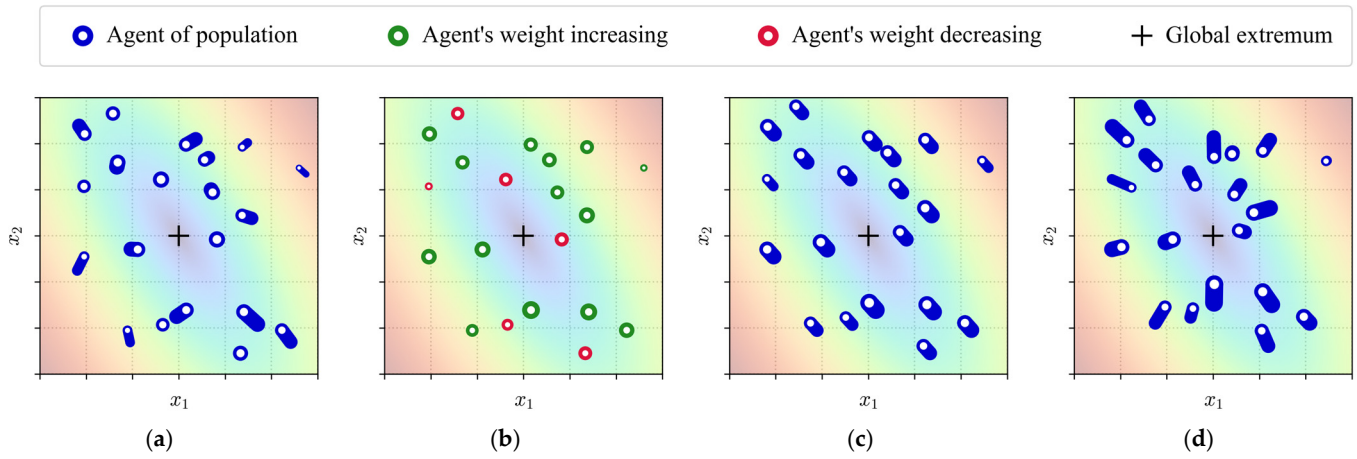

Figure 1. The four main stages of one iteration of the FSS algorithm from left to right: (**a**) individual movement; (**b**) feeding; (**c**) collective-instinctive movement; (**d**) collective-volitive movement.

In the modification of the FSS algorithm proposed in [27] called ETFSS, the authors propose to use chaotic tent-maps to generate the pseudo-random numbers used at different stages of optimization, and the step lengths for the stages of individual and collective-

volitive movement of agents to decay according to the exponential law at each t-th iteration of the algorithm:

$$step_{ind}^{(t+1)} = step_{ind}^{(1)} \cdot \exp\left(\frac{-5t}{T}\right), \tag{10}$$

$$step_{vol}^{(t+1)} = step_{vol}^{(1)} \cdot \exp\left(\frac{-5t}{T}\right), \tag{11}$$

where $step_{ind}^{(1)}$ and $step_{vol}^{(1)}$ denote the initial values for the maximum step lengths of individual and collective-volitive movement, respectively; T is the total number of iterations of the algorithm.

Algorithm 1 summarizes all of the ETFSS steps.

Algorithm 1: ETFSS pseudocode.

Input: f—objective function for minimization;
M—total number of agents (population size);
T—total number of iterations of the algorithm;
w_{max}—maximum weight of an agent;
$step_{ind}^{(1)}$, $step_{vol}^{(1)}$—initial values for the maximum step lengths.

for each i-th agent in population **do**:
  Initialize random position of the agent $\overline{x}_i^{(1)}$;
  Initialize the weight of the agent equal to $0.5 \cdot w_{max}$;
  Calculate the initial value of objective function $f\left[\overline{x}_i^{(1)}\right]$;
end for
for each $t-$th iteration of the total number of T **do**:
  for each i-th agent in population **do**:
    Calculate individual movement $\Delta\overline{x}_i^{(t)}$ using Formula (1);
    Calculate the value of the objective function with the individual movement applied $f\left[\overline{x}_i^{(t)} + \Delta\overline{x}_i^{(t)}\right]$;
    Calculate the difference of the objective function values $\Delta f_i^{(t)}$ using Formula (2);
    Apply the individual movement using Formula (3);
    Update the weight of the agent after the individual movement using Formula (4);
    Limit the weight to the range $[1,\ w_{max}]$;
    Recalculate the difference of the objective function values after the individual movement has been applied using Formula (5);
    Apply the collective-instinctive movement using Formula (6);
  end for
  Calculate common barycenter of all agents using Formula (7);
  for each i-th agent in population **do**:
    Apply the collective-volitive movement using Formulas (8) or (9);
    Calculate the value of the objective function after the movement;
  end for
  Update $step_{ind}^{(t+1)}$ using Formula (10) and $step_{vol}^{(t+1)}$ using Formula (11);
  Find the local best solution (found during the t-th iteration);
  Recalculate the global best solution (found over all iterations up to the t-th).
end for

3.2. The Modified Soft Island Model

In order to improve the convergence of population-based optimization algorithms, as well as to ensure the possibility of speeding up their operation through parallelization, island models are used. The principle of their work is based on dividing the entire population of M agents into N isolated groups, called islands, where each group solves the optimization problem autonomously. But at certain time intervals (for example, every few iterations of the algorithm), information is exchanged between the islands by means of the migration of agents from one island to another. The structure and nature of migration are determined by the topology of connections between islands [55], which can take various forms: ring [56,57], torus [58], lattice [59] and others.

The Soft Island Model (SIM) proposed in [28] realizes a probabilistic approach to the implementation of migrations. Thus, during migrations, each agent stays on its initial island with probability P and moves to any other island except the initial one with probability $1 - P$. In this case, the value of probability P does not change during the optimization process and is a hyperparameter. The main difference between this model and others is that the migration topology presented in SIM is based on probabilities, so it assumes that the number of agents at islands changes during iterations. This model has been implemented for such well-known population-based algorithms as differential evolution (DE) [35] and particle swarming (PSO) [39] and has shown a statistically significant improvement in the results of optimization problems compared to the corresponding classical island-free versions of these algorithms.

In this paper, we propose to modify SIM as applied to the ETFSS algorithm by adding the possibility of changing the values of migration probabilities for agents by taking into account the overall achievements of the islands to which they belong. For this purpose, at each t-th iteration after the feeding stage, the difference in its total weight $\Delta W_k^{(t)}$ compared to the previous iteration is calculated for each k-th island:

$$\Delta W_k^{(t)} = \begin{cases} 0, & W_k^{(t)} < W_k^{(t-1)} \\ W_k^{(t)} - W_k^{(t-1)}, & W_k^{(t)} \geq W_k^{(t-1)} \end{cases}, \tag{12}$$

where $W_k^{(i)}$ denotes the sum of weights of all agents belonging to the k-th island at the t-th iteration of the algorithm, i.e., $W_k^{(t)} = \sum_i^M w_i^{(t)}$; $k = \overline{1, N}$; $i = \overline{1, M}$; N is the number of islands; M is the number of agents in one island (the population of the island).

The migration process is proposed to be carried out after the completion of all four main ETFSS stages and the decay of step lengths at each t-th iteration of the algorithm. Based on analogy with the original Soft Island Model, for the presented algorithm, it is necessary to pre-specify the value of the hyperparameter P.

The probability of an agent belonging to the k-th island to migrate to the h-th island depends on whether h is equal to k:

$$P_h^{(t)} = \begin{cases} P + \frac{\Delta W_h^{(t)}}{\sum_{j=1}^N \Delta W_j^{(t)}}(1 - P), & h = k, \\ \frac{\Delta W_h^{(t)}}{\sum_{j=1}^N \Delta W_j^{(t)}}(1 - P), & h \neq k. \end{cases} \tag{13}$$

It should be noted that the agent stays on the same island if $h = k$ and migrates to h-th island otherwise.

According to Formula (13), each agent will stay on its initial island during the t-th iteration with probability not less than P. The probability of migration to another island does not exceed $1 - P$ and depends on how the total weights of agents of all islands have

changed. The implication is that islands closer to the global extremum are more likely to increase their total weights during the individual movement stage, so their populations are more likely to increase. Thus, the most "successful" islands are given more resources to refine the optimal solution. At the same time, the idea of the probabilistic approach naturally counteracts "monopolization" among islands, since in the process of migrations, any island can get a new agent that represents a solution that is so much better than the other agents that it can significantly shift the barycenter of the island in its direction, strongly affecting its overall results during the collective-volitive movement.

However, this approach does not exclude the possibility of the complete disappearance of islands in the process of the optimization. With a large number of iterations, it may happen that only one of all islands remains, i.e., the algorithm degenerates into a classical version of ETFSS and ceases to take advantage of the advantages of the island model. To prevent the complete disappearance of islands, we propose to set the minimum number of agents L that can include one island as an additional hyperparameter of the algorithm. At each t-th iteration on each k-th island, it is proposed to randomly select exactly L agents that will not participate in the migration process at the current iteration, so they will remain on their islands regardless of the probabilities calculated by Formula (13). Thus, it makes sense to generate populations so that the number of agents on each island is initially greater than L. Then, at any iteration of the algorithm, no island can completely disappear, because there will always be at least L agents on it.

Algorithm 2 summarizes the modified Soft Island Model applied to the ETFSS.

Algorithm 2: SIM-ETFSS pseudocode.

Input: f—objective function for minimization;
N—total number of islands;
S—initial number of agents belonging to one island (population size);
T—total number of iterations of the algorithm;
w_{max}—maximum weight of an agent;
$step_{ind}^{(1)}$, $step_{vol}^{(1)}$—initial values for the maximum step lengths;
P—probability hyperparameter.

1. **for** each k-th island of the total number of N **do**:
2. Initialize S agents as the population of the island (lines 1–5 of Algorithm 1);
3. **end for**
4. **for** each t-th iteration of the total number of T **do**:
5. **for** each k-th island of the total number of N **do**:
6. Perform the steps of the ETFSS (lines 7–21 of Algorithm 1) for k-th island;
7. Calculate the difference in the total weight $\Delta W_k^{(t)}$ using Formula (12);
8. **for** each h-th island of the total number of N **do**:
9. Calculate the probability of an agent to migrate to the h-th island from the k-th island using Formula (13);
10. **end for**
11. **for** each i-th agent of the k-th island **do**:
12. Migrate to another island or not according to the calculated probabilities;
13. **end for**
14. **end for**
15. Update step lengths (line 22 of Algorithm 1);
16. Update local and global best solutions (lines 23–24 of Algorithm 1).
17. **end for**

3.3. Cluster-Based Population Initialization

The initial generation of populations in island models usually includes $N \times S$ random points in the search space, where N is the number of islands and S is the initial number of agents on each island. Thus, N different groups of S agents in each group are obtained. However, this approach does not create conditions for competition between islands, since each island is in an equal situation with the others during initialization. As a result, the effect of cooperative interaction between islands is rather random, at least at the very beginning of population development.

In this paper, we propose a new approach for the generation of initial populations, in which each island forms its own space within a search area in which agents belonging to it are generated. The sizes of such regions are set so that the sum of the hypervolumes of all islands equals the hypervolume of the entire search area. Thus, the length of the side of the hyperrectangle, within which the agents of one island will be generated, along the *dim* axis is calculated as follows:

$$r_{dim}^{(\text{island})} = \frac{r_{dim}^{(\text{space})}}{\sqrt[D]{N}}, \tag{14}$$

where $r_{dim}^{(\text{island})}$ is the length of the side of the hyperrectangle of one island along the *dim* axis; $r_{dim}^{(\text{space})}$ is the length of the side of the hyperrectangle of the whole search area along the *dim* axis; D is the dimensionality of the problem to be solved; N is the number of islands.

For each island, a random point is generated, which is its center, and a population is generated around it, taking into account the size of the hyperrectangle of the island. Figure 2 shows examples of the classical approach, Figure 2a, and the approach proposed in this paper, Figure 2b, for generating an initial population in a two-dimensional search space and its distribution over three islands. The rectangles with dotted borders in the Figure 2b indicate the boundaries of the islands of the corresponding color.

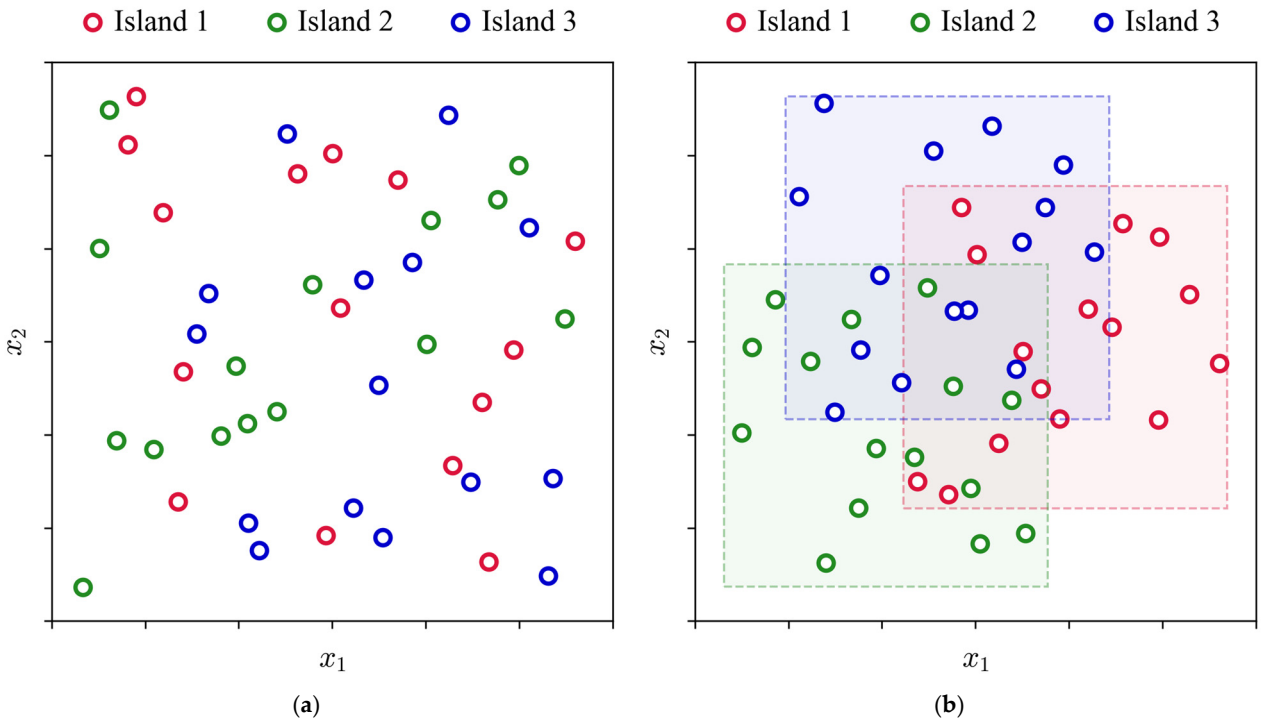

Figure 2. Examples of ways to generate the initial population: (**a**) the classical approach with random generation over the entire search area; (**b**) the approach proposed in this paper with the cluster-based generation of agents for each island.

The proposed approach creates unequal conditions for islands when initiating their populations, thus increasing competition between them. In addition, the roles of islands in the task of optimizing certain parts of the search area are specified, and migrations become more reasonable.

Although the cluster-based generation of populations does not cover the entire search area, as can be seen in Figure 2b, it cannot be said that the remaining space will be ignored by the algorithm. Some SIM-ETFSS stages implement search diversification (exploration) in one way or another, so during optimization, individual agents may move to areas that were not initially covered by any island, if, of course, moving to these areas is justified and can potentially lead to an improvement of the objective function based on the heuristics used in the algorithm.

Algorithm 3 shows the main steps of the cluster-based initialization of island populations.

Algorithm 3: Cluster-based initialization of island populations.

Input: N—total number of islands;
S—initial number of agents belonging to one island (population size);
D—dimensionality of the problem to be solved;
$r_1^{(\text{space})}$, $r_2^{(\text{space})}$, …, $r_D^{(\text{space})}$—lengths of the search area along all D dimensions.

1. **for** each *dim*-th dimension of the total number of D **do**:
2. Calculate the length of the island area along *dim*-th axis using Formula (14);
3. **end for**
4. **for** each k-th island of the total number of N **do**:
5. Place the island area at a random location in the entire search space;
6. Initialize island population of S agents inside its area.
7. **end for**

Line 5 of Algorithm 3 assumes that the island area must be completely within a search space, i.e., a random agent satisfying the search space constraints can be generated at any point in the island area.

3.4. Testing for Defects

As mentioned earlier, the creation of new optimization algorithms and improvement of existing ones is an urgent direction for research in the area under study, so at the moment there is a huge number of different heuristics used in solving optimization problems. However, fundamentally new approaches do not always turn out to be exceptionally useful, and sometimes even on the contrary, they have some implicit defects that have an extremely negative impact on the optimization process. For this reason, it makes sense to check the SIM-ETFSS algorithm presented in this paper for such known defects as the center-bias operator (CBO) [25] and the unevenness defect (UD) [32].

3.4.1. The Center-Bias Operator

The defect caused by the presence of the center-bias operator manifests itself in the uneven exploration of the entire available search area, giving greater preference to the space closer to its center. As a result, population-based algorithms possessing this defect show better results based on those problems for which the global extremum of the objective function coincides or is located in the vicinity of the center of the given search area, with much less qualitative results in cases where the optimal solution is located on the periphery. To achieve this undesirable effect, the center-bias operator itself does not necessarily have to be directly implemented in the algorithm but can be indirectly

manifested due to the internal logic of its operation or the features of the applied heuristics. Figure 3 shows a visualization of the algorithm behavior with such a defect on the example of minimizing a function of the two variables $f(x_1, x_2)$. The background color in the subfigures represents the landscape of the objective function, ranging from lower values (blue) to higher values (red).

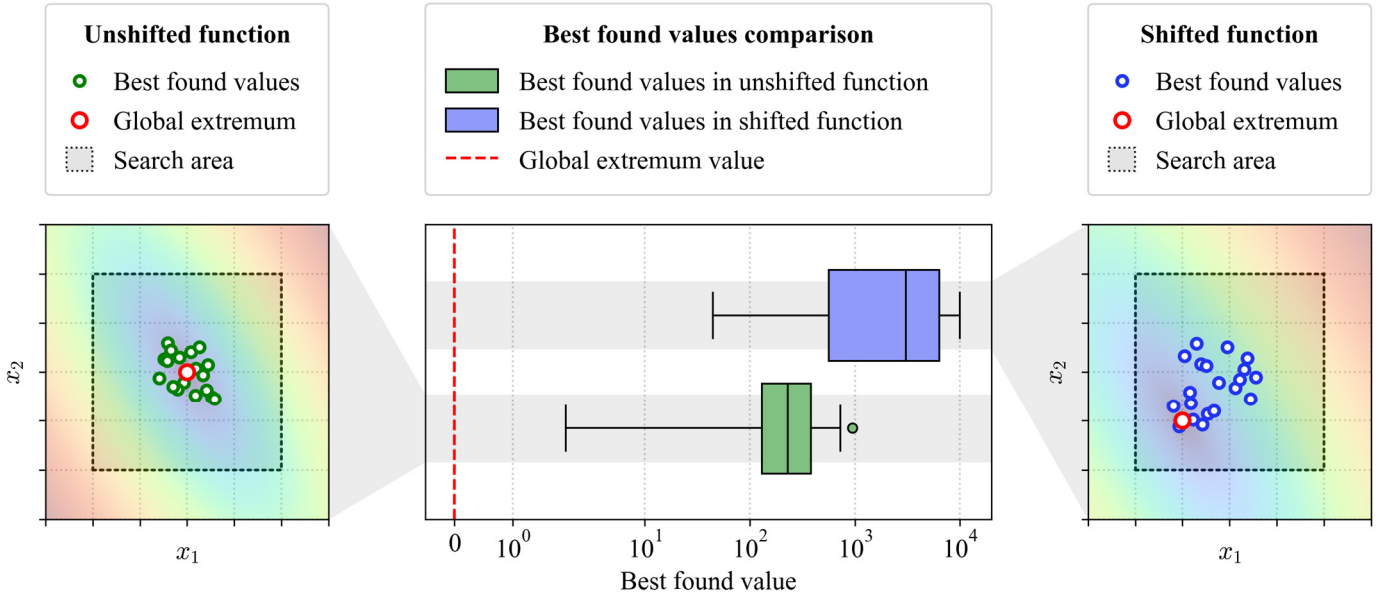

Figure 3. Visualization of the behavior of the algorithm with the center-bias operator.

In [24,25], a simple method of detecting the center-bias operator is presented. For this purpose, it is proposed to prepare a set of Q test functions and to select search areas for them in such a way that global extremums are located in close proximity to their centers.

Then, it is proposed to create a similar set of the same test functions, but the search areas for them should be shifted along each axis by 10% of the search range length (it is assumed that the search area is defined in the form of a hypercube, and the search range lengths are the same for all dimensions). For each test function from both sets, 20 independent runs of the optimization algorithm should be performed, followed by the calculation of errors (the difference between the actual optimal solution and the found one).

Thus, for each function f_q at $q = \overline{1, Q}$, there will be 20 values $\overline{V}_u^{(q)} = \left(v_{u,1}^{(q)}, v_{u,2}^{(q)}, \ldots, v_{u,20}^{(q)}\right)$ obtained with the "unbiased" search area and 20 values $\overline{V}_s^{(q)} = \left(v_{s,1}^{(q)}, v_{s,2}^{(q)}, \ldots, v_{s,20}^{(q)}\right)$ obtained with the "biased" search area. The estimate of the presence of the center-bias operator CBO_{score} is calculated using the following formula:

$$\mathrm{CBO}_{score} = \left(\prod_{q=1}^{Q} \frac{\sum_{j=1}^{20} v_{s,j}^{(q)}}{\sum_{j=1}^{20} v_{u,j}^{(q)}} \right)^{\frac{1}{Q}}. \tag{15}$$

The geometric mean obtained in Formula (15) is an estimate of the presence of the center-bias operator in the considered algorithm. If $\mathrm{CBO}_{score} > 10$, it is considered that the algorithm copes with the problems for which the sought optimal value is in the center of the search area an order of magnitude better than the others, which is a manifestation of a defect.

3.4.2. The Unevenness Defect

The unevenness defect manifests itself in the fact that the quality of optimization depends on the geometric shape of the search area. The closer its shape is to a hypercube, the better results such algorithms produce. In [32], it is shown that, contrary to logic, some population algorithms begin to show much lower quality results if the search area is reduced unevenly along different axes. In other words, narrowing the search area does not lead to the expected improvement in the quality of

optimization but, on the contrary, becomes the cause of the algorithm's performance degradation. Figure 4 shows a visualization of the behavior of the algorithm subject to the unevenness defect, using the example of minimizing a function of two variables $f(x_1, x_2)$. The background color in the subfigures represents the landscape of the objective function, ranging from lower values (blue) to higher values (red).

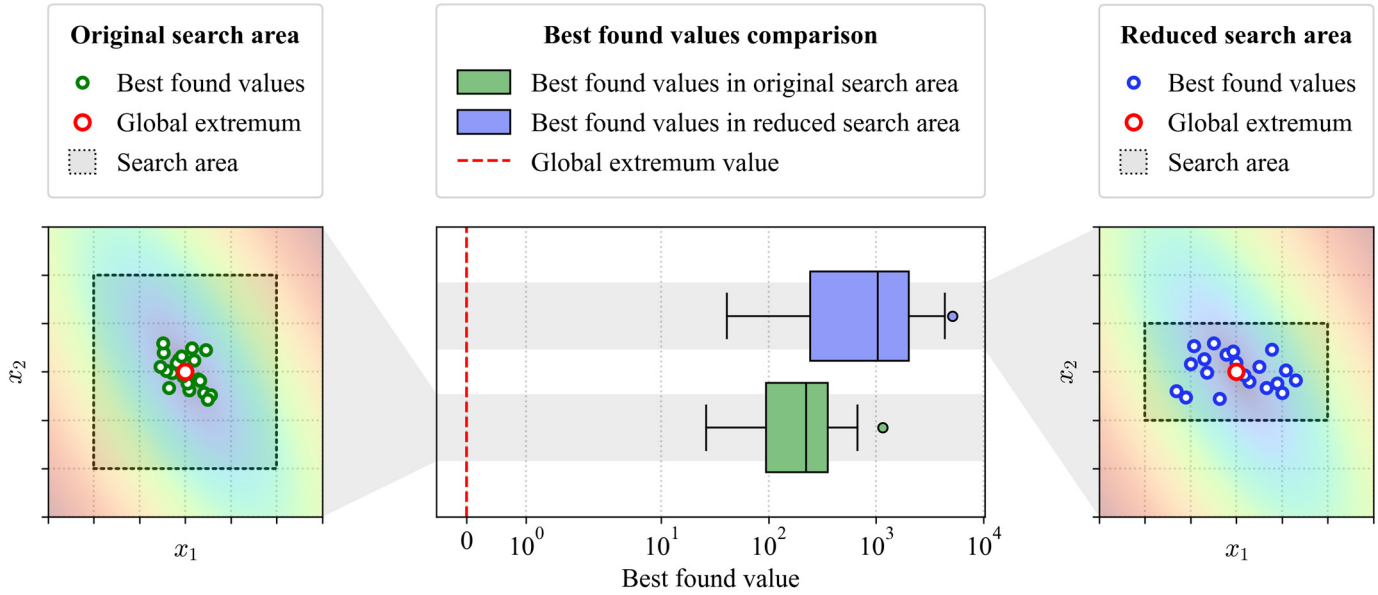

Figure 4. Visualization of the behavior of the algorithm with the unevenness defect.

The following approach is proposed in [32] for unevenness defect detection. Similar to the CBO detection method, first of all, it is proposed to prepare a set of Q different test functions and to select search areas for them so that global extremums are located in close proximity to their centers, and the areas themselves have the same length along all axes.

Then, it is proposed to create almost the same set of the same test functions but to reduce the search areas along each axis so that the length of the range $r_{dim}^{(\text{space})}$ along the dim axis becomes 2 times less than $r_{dim-1}^{(\text{space})}$. For example, if the original search area for a function of three variables was given as $x_1, x_2, x_3 \in [-60,\ 60]$, then it must be transformed to the following form after reduction:

$$\begin{cases} x_1 \in [-60,\ 60] \\ x_2 \in [-30,\ 30] \\ x_3 \in [-15,\ 15] \end{cases}.$$

As in the case of CBO detection, for each test function from both prepared sets it is proposed to perform 20 independent runs of the optimization algorithm with the subsequent calculation of errors (the difference between the actual optimal solution and the found one).

To detect the unevenness defect, it is proposed to use the one-tailed Mann–Whitney U test, which compares 20 results $\overline{V}_o^{(q)} = \left(v_{o,1}^{(q)},\ v_{o,2}^{(q)},\ \ldots,\ v_{o,20}^{(q)}\right)$ obtained for the first set of functions with 20 results $\overline{V}_r^{(q)} = \left(v_{r,1}^{(q)},\ v_{r,2}^{(q)},\ \ldots,\ v_{r,20}^{(q)}\right)$ obtained for the second set for all test functions f_q, where $q = \overline{1, Q}$. The median ratio is proposed to numerically represent the unevenness defect score $\text{UD}_{score}^{(q)}$ for the function f_q:

$$\text{UD}_{score}^{(q)} = \frac{\text{median}\left(v_{o,1}^{(q)},\ v_{o,2}^{(q)},\ \ldots,\ v_{o,20}^{(q)}\right)}{\text{median}\left(v_{r,1}^{(q)},\ v_{r,2}^{(q)},\ \ldots,\ v_{r,20}^{(q)}\right)}. \tag{16}$$

The lower the score calculated using Formula (16), the more the algorithm is susceptible to this defect.

3.5. Benchmark Functions

Table 1 lists all the test functions that will be used within the experiments with the SIM-ETFSS algorithm.

Table 1. Test functions for optimization. The letters "U" and "M" denote unimodal and multimodal functions, respectively. The letters "S" and "N" denote separable and non-separable functions, respectively.

Designation	Function Name	Type	Search Range for Each Dimension
f_1	Sphere	U, S	$[-100, 100]$
f_2	Schwefel 2.22	U, N	$[-100, 100]$
f_3	Schwefel 1.2	U, N	$[-100, 100]$
f_4	Schwefel 2.21	U, S	$[-100, 100]$
f_5	Rosenbrock	U, N	$[-30, 30]$
f_6	Step	U, S	$[-100, 100]$
f_7	Quartic with noise	U, S	$[-1.28, 1.28]$
f_8	Schwefel 2.26	M, S	$[-500, 500]$
f_9	Rastrigin	M, S	$[-5.12, 5.12]$
f_{10}	Ackley	M, N	$[-32, 32]$
f_{11}	Griewank	M, N	$[-600, 600]$
f_{12}	Drop-Wave	M, N	$[-5.12, 5.12]$
f_{13}	Alpine 1	M, S	$[-10, 10]$
f_{14}	HappyCat	M, N	$[-20, 20]$
f_{15}	HGBat	M, N	$[-15, 15]$
f_{16}	Discus	U, S	$[-100, 100]$
f_{17}	Bent Cigar	U, S	$[-100, 100]$
f_{18}	Xin-She Yang	M, N	$[-6.28, 6.28]$
f_{19}	Salomon	M, N	$[-20, 20]$
f_{20}	Zakharov	U, N	$[-10, 10]$

The test functions listed in Table 1 differ both in the terrain complexity and other characteristics. Since there are differences in the definitions of these functions by different authors [60,61], the definitions used in this paper are given below, with $\overline{x} \in \mathbb{R}^D$, where D is the dimensionality of the problem, i.e., $\overline{x} = (x_1, x_2, \ldots, x_D)^T$:

$$f_1(\overline{x}) = \sum_{i=1}^{D} x_i^2,$$

$$f_2(\overline{x}) = \sum_{i=1}^{D} |x_i| + \prod_{i=1}^{D} |x_i|,$$

$$f_3(\overline{x}) = \sum_{i=1}^{D} \left(\sum_{j=1}^{i} x_j \right)^2,$$

$$f_4(\overline{x}) = \max(|x_1|, \ldots, |x_D|),$$

$$f_5(\overline{x}) = \sum_{i=1}^{D-1} \left(100 \left(x_{i+1} - x_i^2 \right)^2 + (x_i - 1)^2 \right),$$

$$f_6(\overline{x}) = \sum_{i=1}^{D} \lfloor |x_i| + 0.5 \rfloor^2,$$

$$f_7(\overline{x}) = \sum_{i=1}^{D} i x_i^4 + \lambda, \text{ where } \lambda \text{ is a random value from } [0, 1),$$

$$f_8(\overline{x}) = 418.9828872724337 \cdot D - \sum_{i=1}^{D} x_i \sin\left(\sqrt{|x_i|} \right),$$

$$f_9(\overline{x}) = 10 \cdot D + \sum_{i=1}^{D} \left(x_i^2 - 10 \cos(2\pi x_i) \right),$$

$$f_{10}(\overline{x}) = -20\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}\right) - \exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi x_i)\right) + e + 20,$$

$$f_{11}(\overline{x}) = 1 + \frac{1}{4000}\sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D}\cos\left(\frac{x_i}{\sqrt{i}}\right),$$

$$f_{12}(\overline{x}) = 1 - \frac{1 + \cos\left(12\sqrt{\sum_{i=1}^{D} x_i^2}\right)}{2 + 0.5\sum_{i=1}^{D} x_i^2},$$

$$f_{13}(\overline{x}) = \sum_{i=1}^{D} |x_i \sin(x_i) + 0.1x_i|,$$

$$f_{14}(\overline{x}) = 0.5 + \left|\sum_{i=1}^{D} x_i^2 - D\right|^{0.25} + \frac{0.5\sum_{i=1}^{D} x_i^2 + \sum_{i=1}^{D} x_i}{D},$$

$$f_{15}(\overline{x}) = 0.5 + \sqrt{\left|\left(\sum_{i=1}^{D} x_i^2\right)^2 - \left(\sum_{i=1}^{D} x_i\right)^2\right|} + \frac{0.5\sum_{i=1}^{D} x_i^2 + \sum_{i=1}^{D} x_i}{D},$$

$$f_{16}(\overline{x}) = 10^6 x_1^2 + \sum_{i=2}^{D} x_i^2,$$

$$f_{17}(\overline{x}) = x_1^2 + 10^6 \sum_{i=2}^{D} x_i^2,$$

$$f_{18}(\overline{x}) = \left(\sum_{i=1}^{D} |x_i|\right) \cdot \exp\left(-\sum_{i=1}^{D} \sin\left(x_i^2\right)\right),$$

$$f_{19}(\overline{x}) = 1 - \cos\left(2\pi\sqrt{\sum_{i=1}^{D} x_i^2}\right) + 0.1\sqrt{\sum_{i=1}^{D} x_i^2},$$

$$f_{20}(\overline{x}) = \sum_{i=1}^{D} x_i^2 + \left(\frac{1}{2}\sum_{i=1}^{D} i x_i\right)^2 + \left(\frac{1}{2}\sum_{i=1}^{D} i x_i\right)^4.$$

Each of the 20 test functions assumes the solution of the minimization problem and has an optimal value equal to zero.

This set of test functions will be later used both for approbation of the proposed SIM-ETFSS algorithm and for checking it for the presence of various defects.

The main advantage of this set of test functions in the context of approbation and the analysis of the proposed SIM-ETFSS algorithm is its diversity. This set contains both simple symmetric and separable functions (e.g., f_1 or f_4), and more complex asymmetric functions with difficult terrain (e.g., f_{11} or f_{15}). At the same time, the search area ranges of different functions vary significantly, which allows the algorithm's ability to solve problems of different data scales to be evaluated.

As mentioned earlier, this set consists of well-known classical test functions, but recently more complex benchmarks, such as CEC [53,54], have been increasingly used to approbate optimization algorithms. The complexity of such benchmarks usually consists of applying various biases, nonlinear transformations and compositions to classical test functions.

The lack of complexity in the presented set of test functions is partially compensated for by the fact that the proposed SIM-ETFSS algorithm is additionally tested for the presence of a center-bias operator and an unevenness defect. Thus, the possible dependence of the quality of optimization results on the position of the global extremum or the shape and scale of the search area is excluded.

4. Results

The evaluation of the island modification of the ETFSS algorithm presented in this paper and its comparison with the original version was performed using the Python 3.12.1 programming language in the JupyterLab 4.3.4 environment on a MacBook Pro 13 2017 A1708 (Apple Inc., Cupertino, CA, USA) test machine with macOS 13.7.1 (22H221). The characteristics of the test machine are summarized in Table 2.

Table 2. Characteristics of the test machine.

Unit	Parameter	Value
CPU	Type	Intel® Core™ i5-7360U
	Clock rate	2.3 GHz (2 cores)
	Architecture	64-bit
	Cache	4 MB shared L3
	Manufacturer	Intel Corporation, Santa Clara, CA, USA
RAM	Type	8 GB LPDDR3
	Clock rate	2133 MHz
	Manufacturer	Samsung Electronics Co., Ltd., Seoul, Republic of Korea

4.1. Algorithm Performance

To estimate the SIM-ETFSS algorithm proposed in this paper, 20 independent runs of the optimization process were performed with different hyperparameter values for each of the test functions of the dimensions 5, 10, 50 and 100 presented in Table 1. All the values of the hyperparameters of the algorithm that were searched during the experiments are presented in Table 3.

Table 3. Hyperparameter values for the experiments. Letters R and C denote classical and cluster-based population initialization, respectively. The last column shows the cumulative sum for the total number of variants for both single island (first term) and multiple islands (second term).

Hyperparameter of the SIM-ETFSS Algorithm	Variable Values: For Single Island	Variable Values: For Multiple Islands	Total Number of Variations
Test function	$f_1, f_2, \ldots, f_{20}$	$f_1, f_2, \ldots, f_{20}$	$20 + 20$
Dimensionality D	5, 10, 50, 100	5, 10, 50, 100	$80 + 80$
Number of islands N	1	2, 3, 4, 5	$80 + 320$
Probability P	1.0	0.0, 0.2, 0.4, 0.6, 0.8, 1.0	$80 + 1920$
Initialization method	R	R, C	$80 + 3840$
Number of iterations T	$20{\cdot}D$	$20{\cdot}D$	$80 + 3840$
Population size M	120	120	$80 + 3840$
Minimum island size L	$\left\lceil \frac{M}{N} \cdot 0.1 \right\rceil$	$\left\lceil \frac{M}{N} \cdot 0.1 \right\rceil$	$80 + 3840$
		Total:	3920

Since 20 independent runs were performed for each variant of the algorithm, a total of $3920{\cdot}20 = 78400$ runs were performed.

In [42,43], a comparison of the ETFSS algorithm with both its original version FSS and other classical algorithms, such as the Genetic Algorithm (GA) and Particle Swarm Optimization (PSO), was presented. The comparison was performed both on classical test functions and on the task of fitting values for the hidden layer weights of the Extreme Learning Machine (ELM) model. In both cases, the ETFSS algorithm showed its advantage over its counterparts, so in this paper, it was decided to focus on the comparison of the new SIM-ETFSS algorithm with its island-free version ETFSS, as well as on the analysis of the influence of hyperparameter values on the quality of optimization.

For the SIM-ETFSS with only one island, the classical population initialization method was used (Figure 2a), since this configuration is fully equivalent to the original island-free ETFSS algorithm, and the hyperparameter P, as well as the initialization method, do not affect the results produced in this case. For 2, 3, 4 and 5 islands, the value of hyperparameter P was chosen from the range 0.0 to 1.0 with a step of 0.2. Both the classical approach and the cluster-based initialization method presented in this paper were used to generate the initial population. The step lengths for the individual and collective-volitive movement of agents were respectively initialized with values equal to 1% and 0.5% of the search area range length from Table 1. The value for the hyperparameter w_{max} was set to 5000 for all experiments. Since all the compared algorithms use the same heuristic for optimization, inspired by fish behavior, they perform the same number of objective function evaluations for the same number

of iterations. Therefore, to ensure the same computational budget [31] for each algorithm, the number of iterations is fixed for each problem and depends on its dimensionality.

The values for the algorithm's hyperparameters presented in Table 3 and below it were chosen empirically, based on the necessity to provide variability in the values, the decisions of the authors of the original articles when conducting similar experiments [27,28,41] and the computational capabilities of the test machine (Table 2), which impose significant time constraints on running the algorithm with a large number of iterations or a large population size.

Thus, 49 variants of the SIM-ETFSS algorithm (1 variant for a single island and $4 \cdot 6 \cdot 2 = 48$ variants for multiple islands, according to the number of permuted values for N, P, and initialization methods, respectively, as shown in Table 3) were tested for each of the 80 tasks (20 test functions with 4 variants of each dimension), for each of which 20 independent runs were performed. Table 4 presents the combinations of hyperparameters that showed the best (lowest in the context of the minimization problem to be solved) in terms of median of the 20 optimal values found (across 20 runs) for each of the problems to be solved.

Table 4. Combinations of SIM-ETFSS hyperparameter values that showed the best median results for each of the 80 test problems. The letter "C" denotes cluster-based initialization of populations for each island in a separate region, and the letter "R" denotes the classical approach to initial population generation.

Function	D=5	D=10	D=50	D=100
f_1	$N = 5,\ P = 0.4,\ (\mathrm{C})$	$N = 4,\ P = 0.6,\ (\mathrm{C})$	$N = 4,\ P = 0.4,\ (\mathrm{C})$	$N = 5,\ P = 0.2,\ (\mathrm{C})$
f_2	$N = 4,\ P = 0.8,\ (\mathrm{C})$	$N = 5,\ P = 0.2,\ (\mathrm{C})$	$N = 4,\ P = 0.2,\ (\mathrm{C})$	$N = 2,\ P = 0.2,\ (\mathrm{R})\ *$
f_3	$N = 5,\ P = 0.6,\ (\mathrm{C})$	$N = 5,\ P = 0.4,\ (\mathrm{C})$	$N = 5,\ P = 0.2,\ (\mathrm{C})$	$N = 5,\ P = 0.2,\ (\mathrm{C})$
f_4	$N = 5,\ P = 0.0,\ (\mathrm{C})$	$N = 5,\ P = 0.4,\ (\mathrm{C})$	$N = 5,\ P = 0.2,\ (\mathrm{C})$	$N = 5,\ P = 0.2,\ (\mathrm{C})$
f_5	$N = 5,\ P = 0.0,\ (\mathrm{C})$	$N = 5,\ P = 0.4,\ (\mathrm{C})$	$N = 5,\ P = 0.4,\ (\mathrm{C})$	$N = 5,\ P = 0.4,\ (\mathrm{C})$
f_6	$N = 5,\ P = 0.8,\ (\mathrm{C})$	$N = 5,\ P = 0.6,\ (\mathrm{C})$	$N = 5,\ P = 0.4,\ (\mathrm{C})$	$N = 5,\ P = 0.2,\ (\mathrm{C})$
f_7	$N = 5,\ P = 0.2,\ (\mathrm{C})$	$N = 4,\ P = 0.2,\ (\mathrm{C})$	$N = 5,\ P = 0.4,\ (\mathrm{C})$	$N = 5,\ P = 0.2,\ (\mathrm{C})$
f_8	$N = 2,\ P = 0.0,\ (\mathrm{R})\ *$	$N = 5,\ P = 0.2,\ (\mathrm{R})\ *$	$N = 2,\ P = 0.4,\ (\mathrm{R})\ *$	$N = 2,\ P = 0.4,\ (\mathrm{R})\ *$
f_9	$N = 4,\ P = 0.8,\ (\mathrm{C})$	$N = 5,\ P = 0.0,\ (\mathrm{C})$	$N = 4,\ P = 0.6,\ (\mathrm{C})$	$N = 5,\ P = 0.2,\ (\mathrm{C})$
f_{10}	$N = 5,\ P = 0.4,\ (\mathrm{C})$	$N = 4,\ P = 0.4,\ (\mathrm{C})$	$N = 5,\ P = 0.4,\ (\mathrm{C})$	$N = 3,\ P = 0.4,\ (\mathrm{C})$
f_{11}	$N = 5,\ P = 0.8,\ (\mathrm{C})$	$N = 5,\ P = 0.2,\ (\mathrm{C})$	$N = 5,\ P = 0.2,\ (\mathrm{C})$	$N = 5,\ P = 0.2,\ (\mathrm{C})$
f_{12}	$N = 5,\ P = 1.0,\ (\mathrm{C})$	$N = 5,\ P = 0.6,\ (\mathrm{C})$	$N = 5,\ P = 0.6,\ (\mathrm{C})$	$N = 5,\ P = 0.6,\ (\mathrm{C})$
f_{13}	$N = 4,\ P = 0.4,\ (\mathrm{C})$	$N = 5,\ P = 0.4,\ (\mathrm{C})$	$N = 4,\ P = 0.0,\ (\mathrm{C})$	$N = 3,\ P = 0.8,\ (\mathrm{C})$
f_{14}	$N = 5,\ P = 0.2,\ (\mathrm{C})$	$N = 5,\ P = 0.2,\ (\mathrm{C})$	$N = 4,\ P = 0.4,\ (\mathrm{C})$	$N = 5,\ P = 0.0,\ (\mathrm{C})$
f_{15}	$N = 5,\ P = 0.0,\ (\mathrm{C})$	$N = 5,\ P = 0.0,\ (\mathrm{C})$	$N = 5,\ P = 0.4,\ (\mathrm{C})$	$N = 5,\ P = 0.2,\ (\mathrm{C})$
f_{16}	$N = 5,\ P = 0.2,\ (\mathrm{C})$	$N = 4,\ P = 0.4,\ (\mathrm{C})$	$N = 5,\ P = 0.2,\ (\mathrm{C})$	$N = 5,\ P = 0.2,\ (\mathrm{C})$
f_{17}	$N = 5,\ P = 0.0,\ (\mathrm{C})$	$N = 5,\ P = 0.2,\ (\mathrm{C})$	$N = 4,\ P = 0.4,\ (\mathrm{C})$	$N = 5,\ P = 0.2,\ (\mathrm{C})$
f_{18}	$N = 5,\ P = 0.0,\ (\mathrm{C})$	$N = 2,\ P = 0.4,\ (\mathrm{C})$	$N = 2,\ P = 0.0,\ (\mathrm{R})\ *$	$N = 3,\ P = 1.0,\ (\mathrm{R})\ *$
f_{19}	$N = 4,\ P = 0.0,\ (\mathrm{C})$	$N = 5,\ P = 0.4,\ (\mathrm{C})$	$N = 4,\ P = 0.6,\ (\mathrm{C})$	$N = 5,\ P = 0.2,\ (\mathrm{C})$
f_{20}	$N = 5,\ P = 0.2,\ (\mathrm{C})$	$N = 5,\ P = 0.2,\ (\mathrm{C})$	$N = 5,\ P = 0.2,\ (\mathrm{C})$	$N = 3,\ P = 0.2,\ (\mathrm{C})$

* Combinations of hyperparameter values with classical random initialization.

As can be seen from Table 4, the original version of ETFSS did not show superiority over its island modification SIM-ETFSS in any of the experiments, as the number of islands in the best combination of hyperparameters for each task was greater than one. The best hyperparameter value P, meanwhile, varied widely both between different tasks and between different dimensions within the same task. Figure 5 shows a histogram of the number of the best median results obtained by the algorithm with different values of the hyperparameter P, according to Table 4.

Exclusively for the purpose of visualizing the dependence of the optimization quality on the choice of the P hyperparameter value, we normalized all the obtained results with respect to each of the 80 problems (20 test functions of 4 variants of dimensions): each b-th value $v_b^{(f_q\colon \mathbb{R}^D \to \mathbb{R})}$ obtained during the optimization of the function f_q of dimension D was normalized by the following formula:

$$z_b^{(f_q\colon \mathbb{R}^D \to \mathbb{R})} = \frac{v_b^{(f_q\colon \mathbb{R}^D \to \mathbb{R})} - \mu^{(f_q\colon \mathbb{R}^D \to \mathbb{R})}}{\sigma^{(f_q\colon \mathbb{R}^D \to \mathbb{R})}}, \tag{17}$$

where $\mu^{(f_q:\mathbb{R}^D\to\mathbb{R})}$ is the mean among all values $v_b^{(f_q:\mathbb{R}^D\to\mathbb{R})}$; $\sigma^{(f_q:\mathbb{R}^D\to\mathbb{R})}$ is the standard deviation among all values $v_b^{(f_q:\mathbb{R}^D\to\mathbb{R})}$; $q = \overline{1,20}$; $b = \overline{1,960}$; $D = 5$, 10, 50, 100.

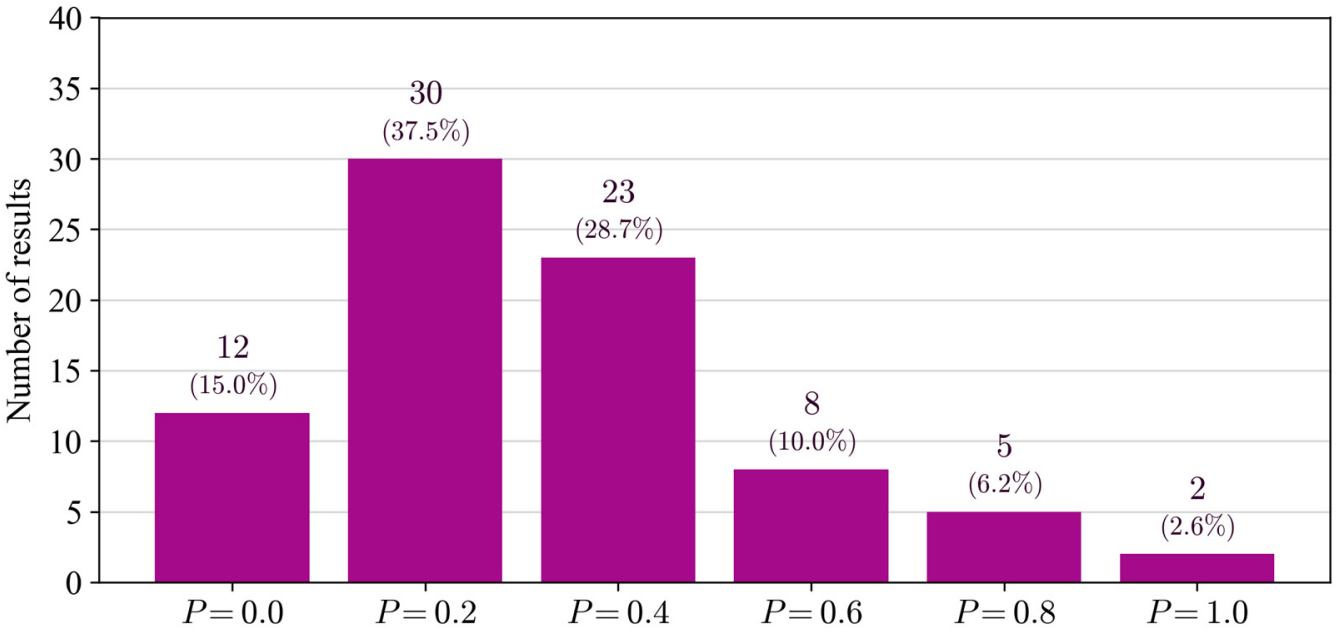

Figure 5. Histogram of the number of the best median results of all conducted experiments with different values of the hyperparameter P. The percentages of the total number of results are shown in brackets.

The maximum value for b, equal to 960, is obtained based on the fact that 48 different variants of the SIM-ETFSS algorithm were tested for each problem for with more than one island, each with 20 independent runs (Table 3). The results of the single-island algorithm runs are not included in the analysis because only a single value of $P = 1.0$ was used to obtain them, which does not make sense in the context of comparing different values of this hyperparameter.

The proposed normalization will make it possible to combine the results obtained using different values of hyperparameters for each of the 80 problems. This will allow the consistency of the obtained results to be visually estimated in the context of investigating the behavior of the algorithm in different conditions.

Thus, Figure 6 presents the analysis of all results (after normalization) for each of the 20 test functions with respect to different values of the hyperparameter P in the form of box plots. Since each test function assumes a minimization problem, lower positions of the boxes along the Y-axis on the diagram, particularly the box boundaries and the median lines within the boxes, correspond to better optimization results.

Figure 6 shows that using the value $P = 1.0$ in most cases leads to noticeably lower quality results. This behavior can be considered expected, since running an algorithm with such a hyperparameter is equivalent to N independent parallel runs of an algorithm with a smaller population size. This is due to the fact that the migration process cannot be carried out, because the minimum probability for each agent to stay on its island is equal to one.

On the other hand, setting $P = 0.0$ means a complete shutdown of external control over the migration process. In this case, the probabilities for each agent to "move" to another island are formed only on the basis of the total achievements of the whole island for the past iteration. As can be seen from Figure 6, this strategy turned out to be the most preferable in only one case (only for the function f_{18}).

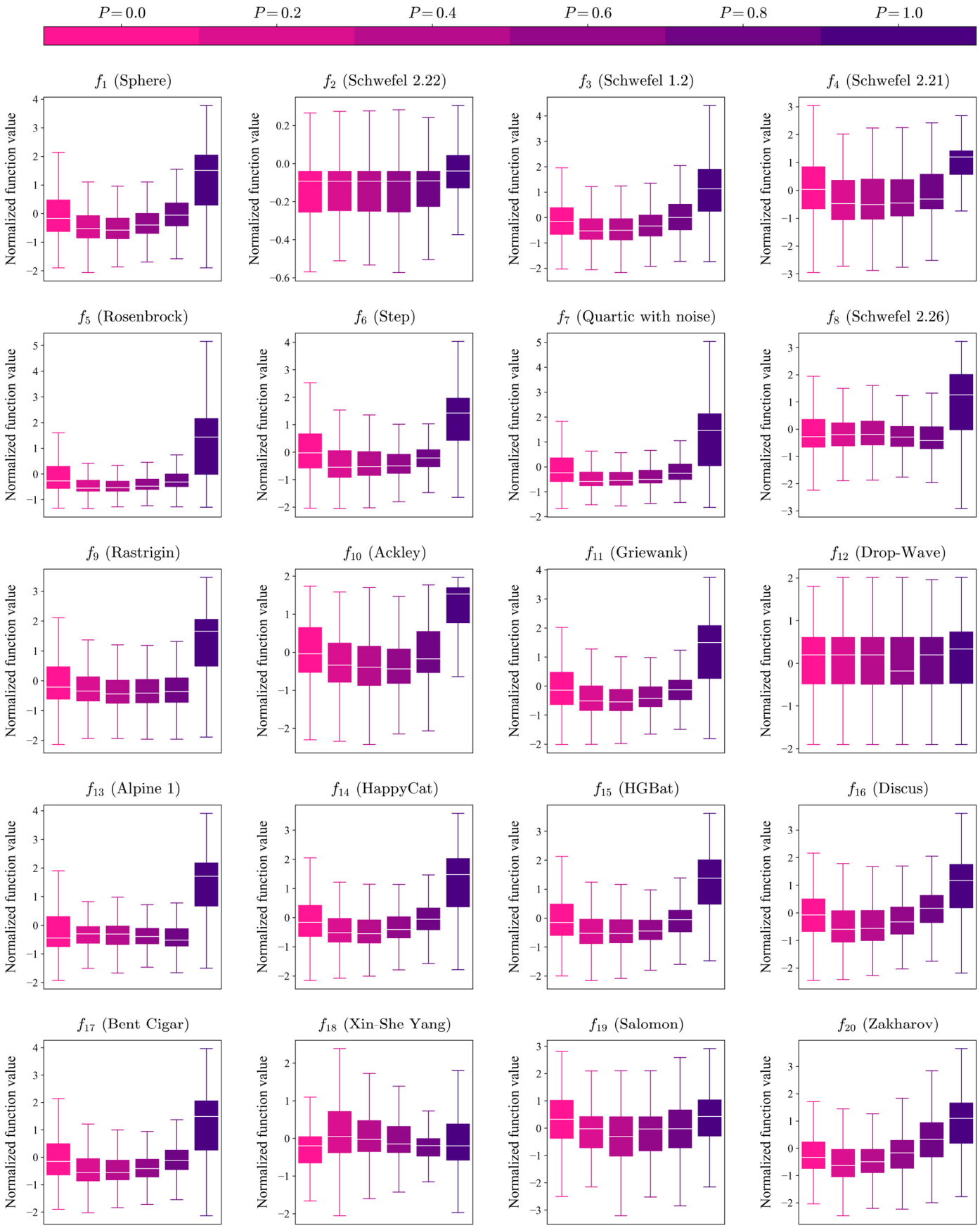

Figure 6. Distribution of normalized experimental results for each function with respect to the value of the hyperparameter P.

The optimal value for the hyperparameter P is most often $P = 0.2$ or $P = 0.3$, which is fully consistent with the distribution of best median results shown in Figure 5.

The convergence of the algorithm can be evaluated using the variance of the obtained results, which is visually represented in Figure 6 as the height of boxes (not including whiskers). As can be seen, the best convergence of the algorithm is most often demonstrated with optimal values of the hyperparameter P discussed earlier.

Figure 7 shows a histogram of the number of the best median results obtained using the algorithm with different numbers of islands, according to Table 4.

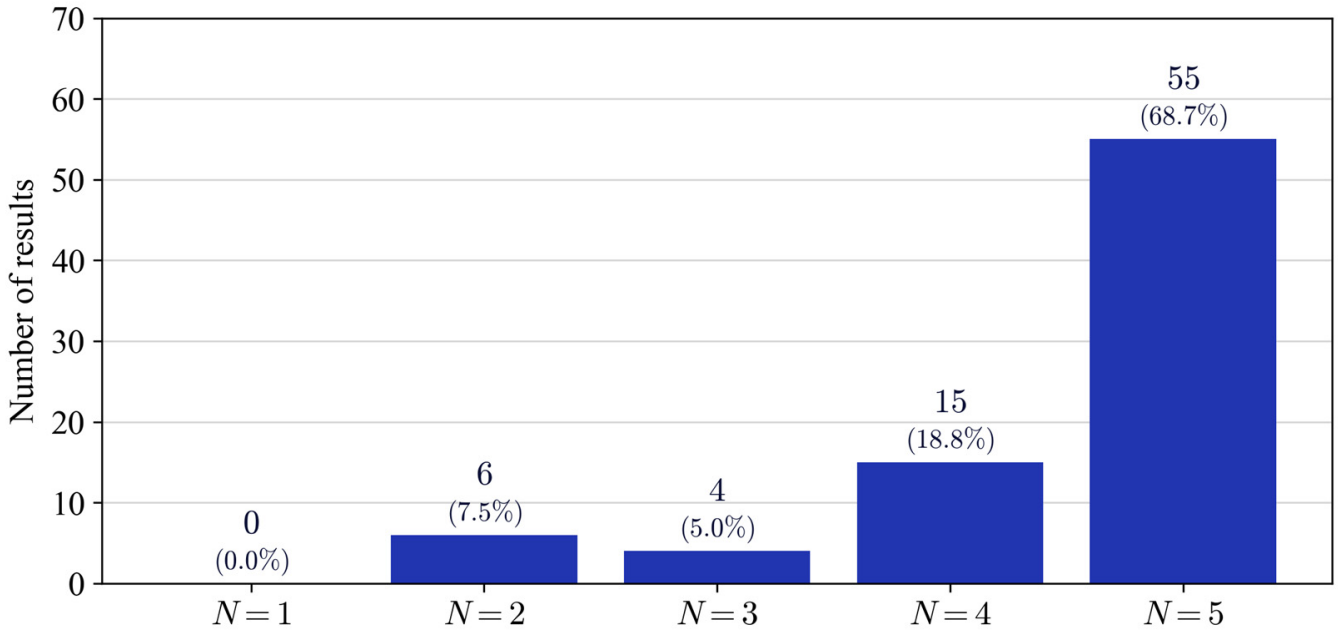

Figure 7. Histogram of the numbers of the best median results of all conducted experiments with different values of the hyperparameter N. The percentages of the total number of results are shown in brackets.

As it was mentioned earlier, there was not a single result with $N = 1$ among the best median results. At the same time, one can see a tendency of the quality of the results to increase with an increasing number of islands.

Similar to the analysis of all the results obtained for different values of the hyperparameter P performed earlier, in order to consider the dependence of the optimization quality on the number of islands, we also performed data normalization for all test functions f_q for $q = \overline{1, 20}$ for all dimensions $D = 5,\ 10,\ 50,\ 100$ using Formula (17). This time, however, all 980 values were considered for each of the 80 problems, including the data obtained using the single island algorithm.

Figure 8 shows the analysis of all results (after normalization) for each of the 20 test functions with respect to different values of hyperparameter N and population initialization methods. By analogy with Figure 6, the diagram shows that lower positions of the boxes along the Y-axis, particularly the box boundaries and the median lines within the boxes, corresponding to better optimization results, since each test function assumes a minimization problem.

From Figure 8, it can be seen that the original island-free ETFSS algorithm, which is equivalent to the SIM-ETFSS algorithm with a single island, performs worse in comparison to its islanded version in most cases.

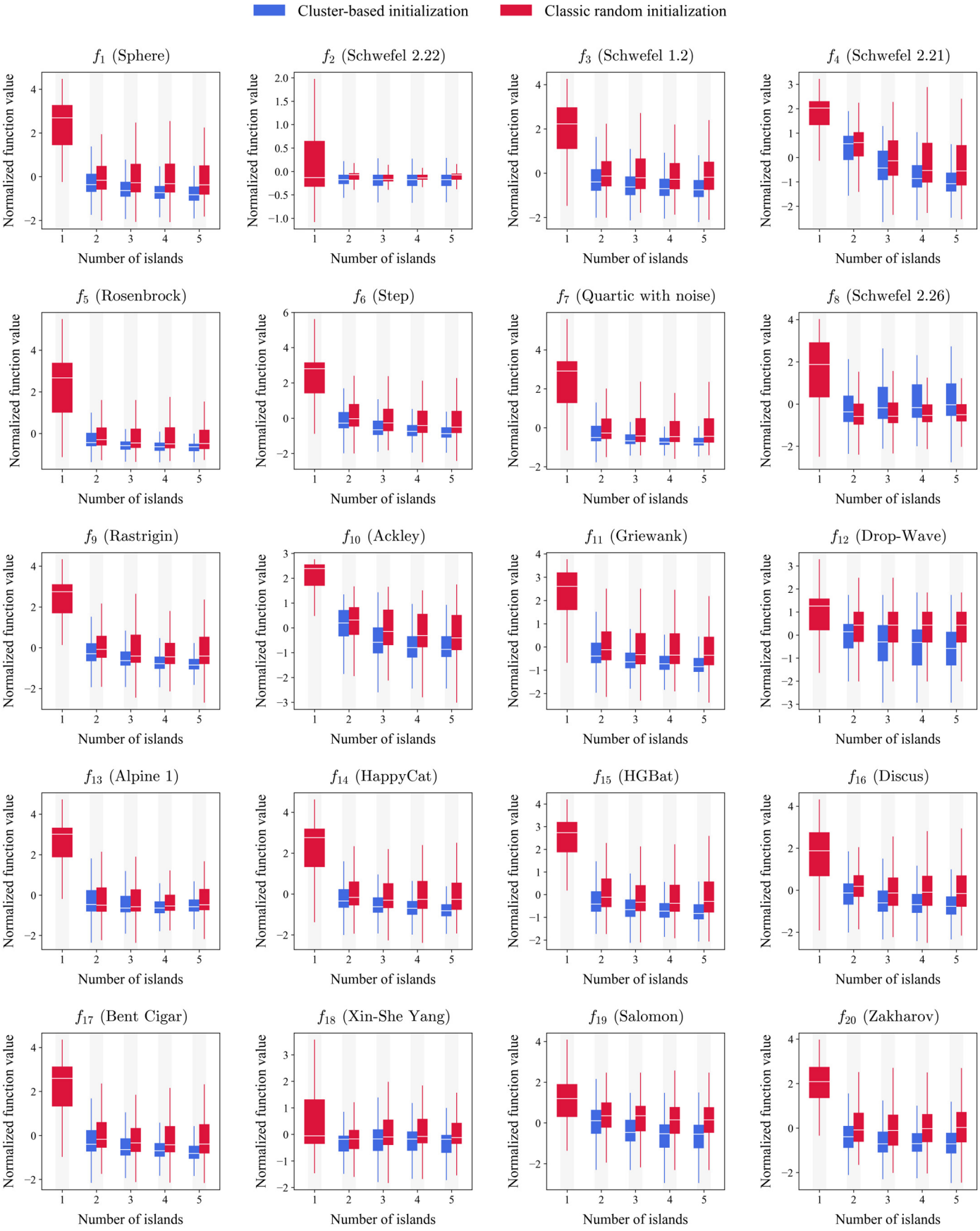

Figure 8. Distribution of normalized experimental results for each function with respect to the number of islands and initialization method.

However, the difference between the two is equally noticeable for all test functions. For example, in the case of f_{18} (Xin-She Yang), the medians of the results for different numbers of islands can be considered comparable. In order to explain this fact, we propose to examine the raw data for the f_{18} function on a logarithmic scale in more detail. Figure 9 shows the box plot for this function of dimension 100 for $P = 0.2$.

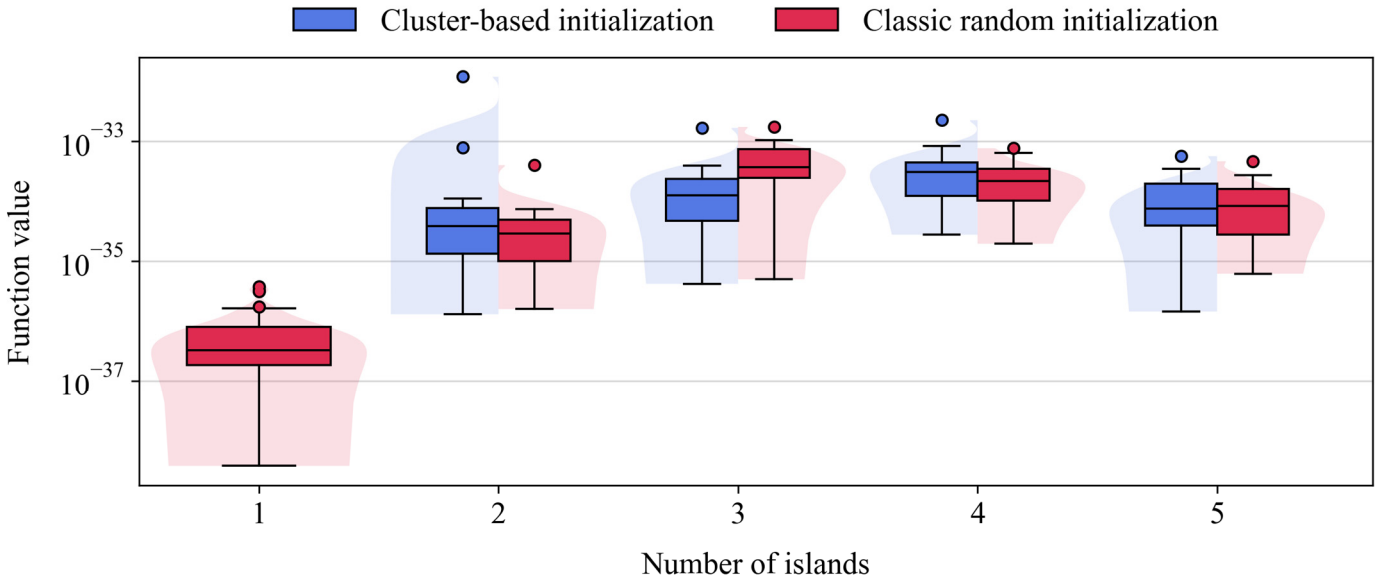

Figure 9. Comparison of the original version of ETFSS (one island) with the SIM-ETFSS algorithm (two or more islands) based on the optimization of the function f_{18} (Xin-She Yang) of dimension 100 for $P = 0.2$.

Figure 8 also shows the convergence of the algorithm when using different numbers of islands and different methods of population initialization. For this purpose, the variance of the results, represented by the height of the corresponding boxes on the plot, can be evaluated. In most cases, it is clearly seen that the results of the island-free ETFSS algorithm for almost all problems have a much larger spread than the results of the proposed island modification with a different numbers of islands. In addition, it is worth noting that the use of cluster-based initialization of populations (blue boxes in Figure 8) most often leads to better convergence of the algorithm compared to the use of classical random initialization (red boxes in Figure 8).

Figure 9 shows that the best values found by all algorithms when optimizing the function f_{18} are located close to zero, i.e., to the global extremum. One of the advantages of the island model is to increase the convergence speed of the algorithm, but in this case, given the initially narrow search area for the function f_{18}, as well as the smoothness and symmetry of the terrain in the vicinity of its global extremum, we can conclude that this problem is initially simple. Therefore, the key aspect of the result was not the convergence rate, but the population size, which is always larger for the ETFSS algorithm than for any SIM-ETFSS island for $N > 1$.

Another standout example in Figure 8 is the function f_2 (Schwefel 2.22). Similar to the previous case, we propose to take a closer look at the raw data (without normalization) for this function. Figure 10 shows a box plot for this function of dimension 10 for $P = 0.6$.

Unlike the previous example, Figure 10 shows that the results obtained during the optimization of the function f_2 are far from the global extremum. All versions of the algorithm mediocrely coped with the task, so their results are comparable. However, it is impossible not to mention the noticeable advantage of cluster-based initialization over the classical initialization approach.

According to Table 4, among the initialization methods, cluster-based population generation for each island in a separate region proved to be the best option in most cases. For a more detailed comparison, the one-tailed Mann–Whitney U test with an acceptable significance level $\alpha = 0.01$ was applied to each of the 1920 results of the experiment with the number of islands not equal to one (80 tasks for 2, 3, 4, and 5 islands with 6 variants of values for P), which serves as an indicator of the advantage of one method of initialization over another in the case of its presence. The results of its

application, grouped with respect to the dimensionality of the problem D, the number of islands N and the values of the hyperparameter P, are presented in Figure 11 in the form of bar diagrams.

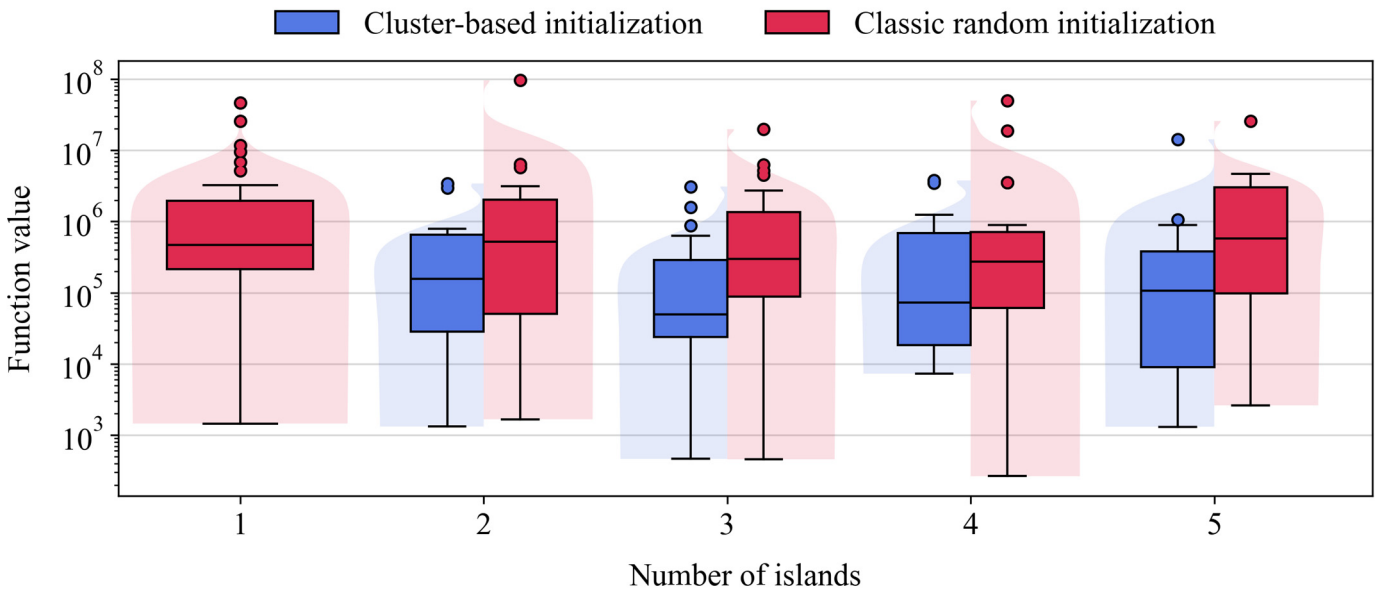

Figure 10. Comparison of the original version of ETFSS (one island) with the SIM-ETFSS algorithm (two or more islands) based on the optimization of the function f_2 (Schwefel 2.22) of dimension 10 for $P = 0.6$.

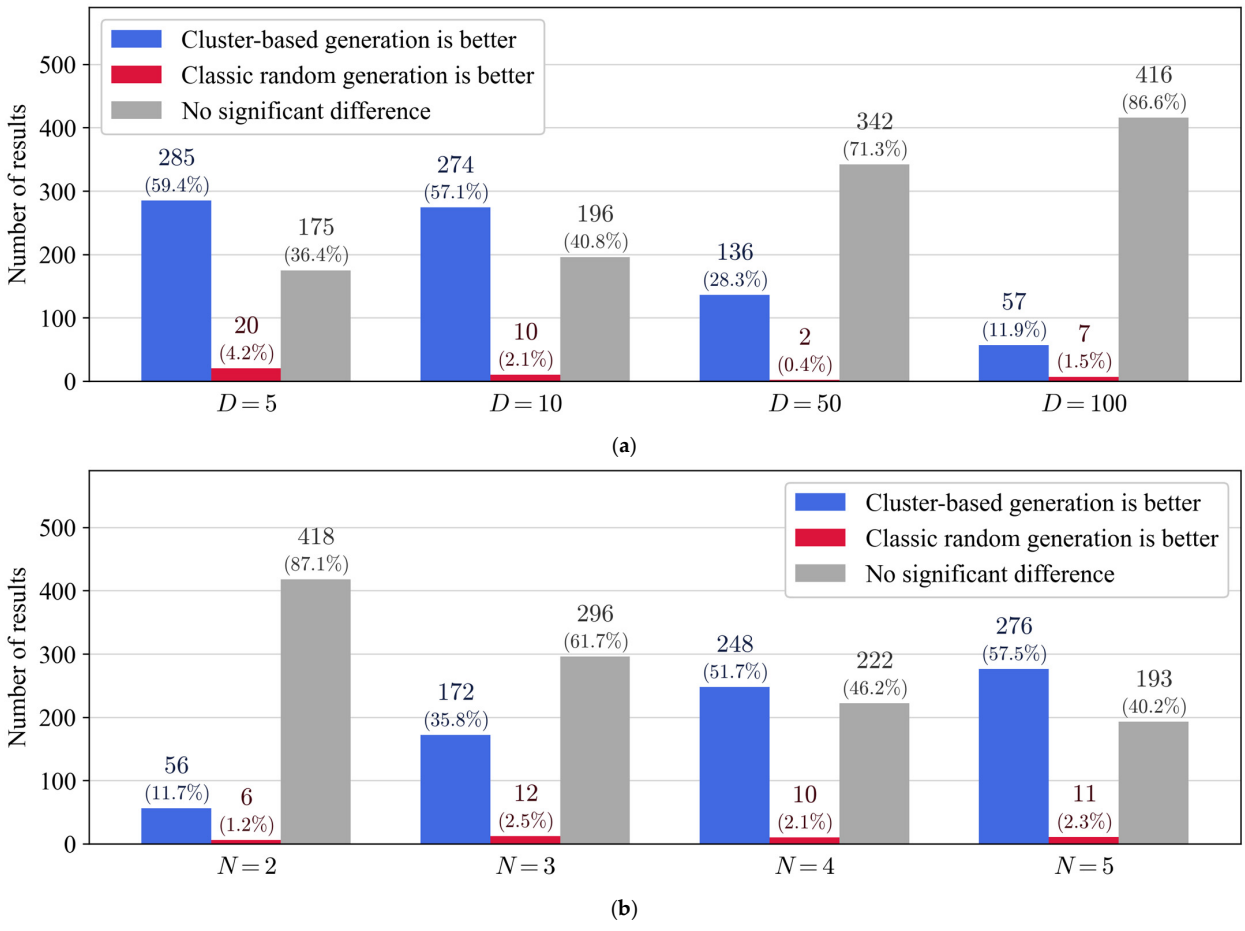

Figure 11. *Cont.*

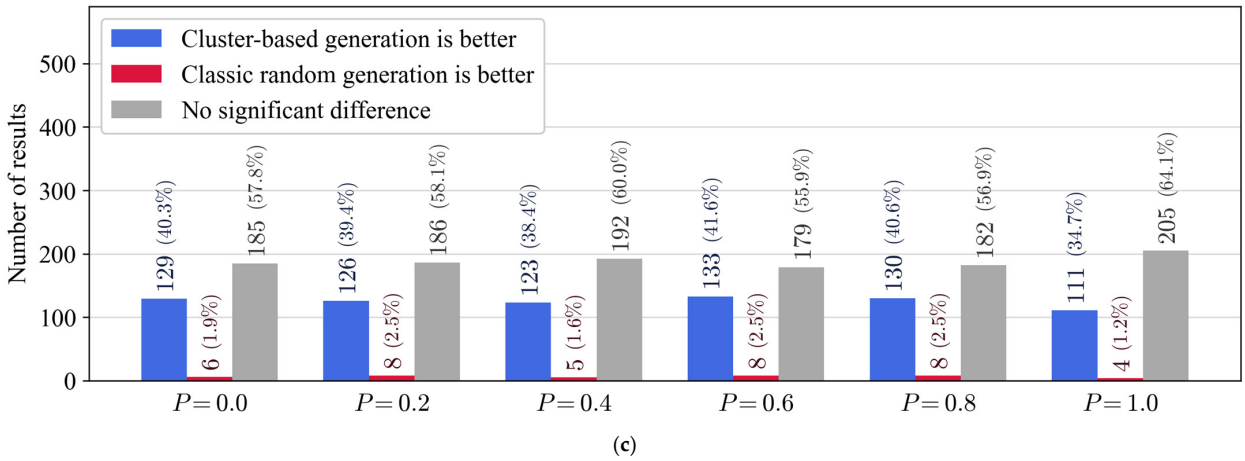

Figure 11. Comparison of population initialization methods based on calculating the results of the Mann–Whitney U test with respect to (**a**) different problem dimensions, (**b**) different numbers of islands and (**c**) different values of the hyperparameter P. The percentages of the total number of results are shown in brackets.

Figure 11a shows that as the dimensionality of the problem decreases, the influence of the choice of a particular method of initial population generation on the algorithm's performance decreases, as evidenced by the number of situations for which no statistically significant differences were observed. Nevertheless, it cannot be overlooked that for each of the dimensions, cluster-based generation was often better than the classical random approach. It can be assumed that the observed tendency for the differences between the two methods of initialization to decrease with increasing problem dimensionality is due to the so-called "curse of dimensionality", because the number of iterations required to solve the problem is linearly related to the problem dimensionality, which is not comparable to the real increase in the computational complexity required to solve the problem as the number of its dimensions increases.

On the other hand, Figure 11b shows that with an increasing number of islands, the superiority of the cluster-based initialization of populations over classical random generation becomes more and more noticeable, which is confirmed by the decrease in cases with no statistically significant differences.

At the same time, it can be observed from Figure 11c that the advantages of different methods of initialization do not depend on the choice of the value of P, since the number of results of the Mann–Whitney U test is distributed uniformly over all values of the hyperparameter P.

The SIM-ETFSS performance can be further examined by analyzing island interactions throughout all iterations of the algorithm. It is particularly interesting to consider the different values of the hyperparameter P, which can be used to visually assess the dynamics of migrations.

Figure 12a shows an example of a migration map for the f_9 (Rastrigin) function for $P = 0.8$, where the Y-axis corresponds to a population of 60 agents uniformly distributed over three islands, and the X-axis corresponds to iterations of the algorithm starting from iteration zero, where the population was generated using cluster-based initialization. Figure 12b shows a plot of the change in the number of agents in the islands based on the iterations of the algorithm.

This example shows that during the optimization of the objective function, the islands actively compete with each other, either increasing their population several times or losing agents to more "successful" rivals. For example, the third island, shown in Figure 12 in blue, from the very beginning showed lower quality results compared to other islands, so it lost almost all of its agents during the first 20 iterations. But then, starting at about the 30th iteration, its performance improved and it absorbed most of the agents from the remaining islands, but after about another 20 iterations, it lost them again. The remaining two islands also actively competed with each other for the entire 100 iterations.

This example demonstrates that population development in the SIM-ETFSS algorithm is much like an evolutionary process in which the most "adaptable" islands that are close to the optimal value

of the function being optimized crowd out the others, increasing their own resources for finding and refining solutions.

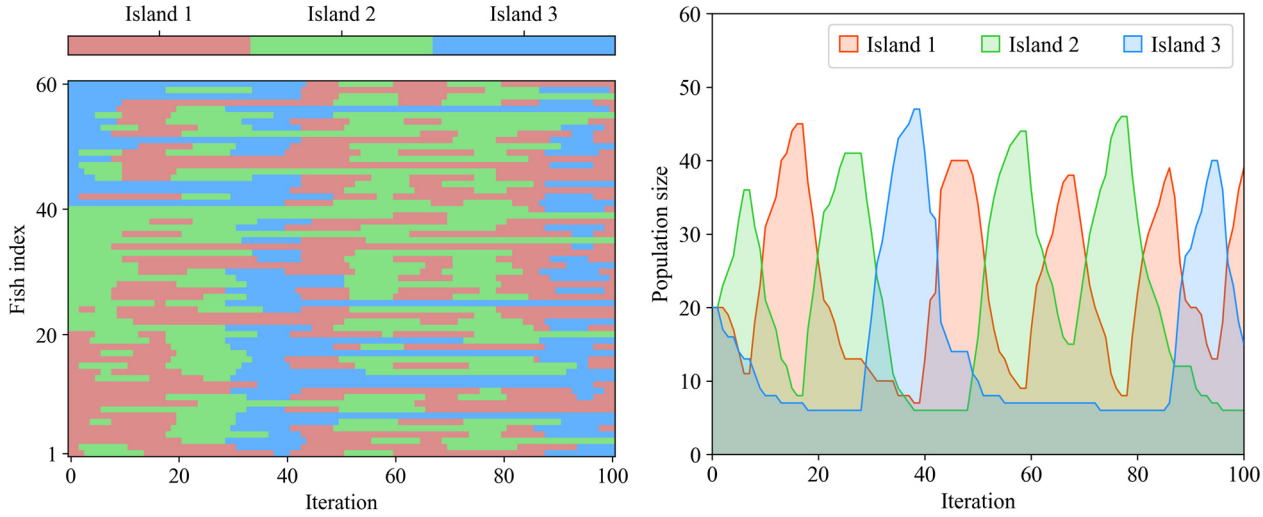

Figure 12. Example of the evolution of a population of 60 agents distributed over three islands over 100 iterations at $P = 0.8$: (**a**) migration map; (**b**) changes in agent numbers across islands during iterations.

Due to the existing constraint on the minimum number of agents belonging to a single island, no island can completely disappear, so competition can continue throughout the iterations of the algorithm. In Figure 12b, this fact is particularly evident in the lower part of the graph.

Similarly, it is proposed to consider the migration process for a lower value of the hyperparameter P. Figure 13 shows an example for the same test function f_9 (Rastrigin) but for $P = 0.2$.

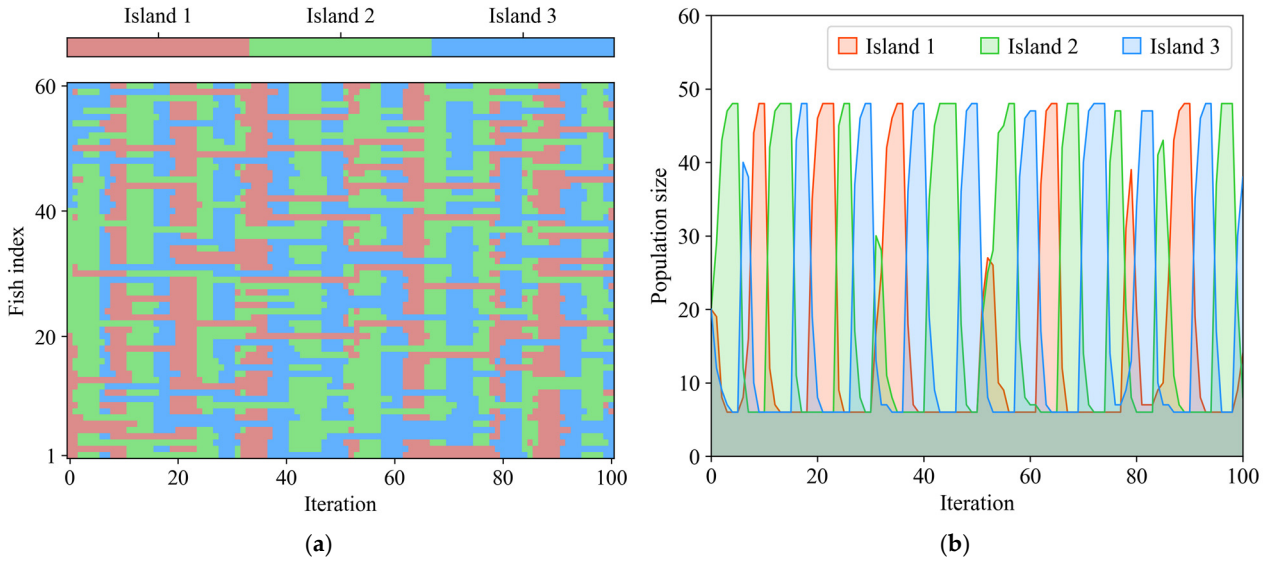

Figure 13. Example of the evolution of a population of 60 agents distributed over three islands over 100 iterations for $P = 0.2$: (**a**) migration map; (**b**) changes in agent numbers across islands during iterations.

Figure 13 shows that a low value of the hyperparameter P leads to an expectedly richer and more dynamic migration process, as can be seen from the frequency of changes in the island population during iterations of the algorithm.

4.2. Defect Detection

Since the best configuration of the SIM-ETFSS algorithm during conducted experiments turned out to be a model with five islands at $P = 0.2$, it was decided to use these values for hyperparameters to test the algorithm for the presence of the center-bias operator and the unevenness defect in its comparison with the original island-free model.

4.2.1. Testing for the Presence of a Center-Bias Operator

Table 5 shows the results of testing the original version of ETFSS, equivalent to SIM-ETFSS with a single island, with the SIM-ETFSS algorithm itself for the presence of a center-bias operator. All functions from Table 1 of dimension 30 were used for testing. The other values of the hyperparameters of the algorithm, except for the already mentioned $N = 5$ and $P = 0.2$, are the same as in Table 3.

Table 5. Results of testing the algorithm for the presence of a center-bias operator. The letter "C" denotes the cluster-based initialization of populations for each island in a separate region; the letter "R" denotes the classical approach to the initial population generation.

Function	**Values of the Algorithm Hyperparameters**		
	N = 1, (R)	**N = 5, (C)**	**N = 5, (R)**
f_1	9.83×10^{-1}	1.08×10^{0}	1.02×10^{0}
f_2	4.16×10^{-1}	1.36×10^{0}	6.89×10^{0}
f_3	1.33×10^{0}	1.26×10^{0}	1.41×10^{0}
f_4	1.00×10^{0}	9.83×10^{-1}	1.01×10^{0}
f_5	1.07×10^{0}	1.18×10^{0}	1.08×10^{0}
f_6	1.02×10^{0}	1.03×10^{0}	1.01×10^{0}
f_7	1.03×10^{0}	1.03×10^{0}	1.15×10^{0}
f_8	1.09×10^{0}	9.41×10^{-1}	9.25×10^{-1}
f_9	1.01×10^{0}	1.01×10^{0}	1.05×10^{0}
f_{10}	1.02×10^{0}	1.01×10^{0}	1.01×10^{0}
f_{11}	1.00×10^{0}	1.07×10^{0}	1.13×10^{0}
f_{12}	1.00×10^{0}	1.00×10^{0}	1.00×10^{0}
f_{13}	1.25×10^{0}	1.20×10^{0}	1.10×10^{0}
f_{14}	9.59×10^{-1}	1.14×10^{0}	1.10×10^{0}
f_{15}	1.03×10^{0}	1.09×10^{0}	1.10×10^{0}
f_{16}	1.07×10^{0}	1.15×10^{0}	1.10×10^{0}
f_{17}	1.01×10^{0}	1.03×10^{0}	1.15×10^{0}
f_{18}	1.37×10^{0}	1.00×10^{0}	1.00×10^{0}
f_{19}	1.01×10^{0}	1.02×10^{0}	1.03×10^{0}
f_{20}	1.09×10^{0}	1.12×10^{0}	1.06×10^{0}
CBO_{score}	1.02×10^{0}	1.08×10^{0}	1.18×10^{0}

As mentioned earlier, CBO_{score} is a measure of the ratio of the quality of results obtained during optimization of test functions with a shifted global extremum to the quality of results obtained during optimization of initial functions with the optimal value in the center of the search area. In this case, the presence of the center-bias operator is registered if this measure exceeds the number 10 [25]. All the results presented in Table 5 are close to 1, which indicates the absence of the center-bias operator for both ETFSS and its island modification SIM-ETFSS presented in this paper.

4.2.2. Testing for the Presence of an Unevenness Defect

Table 6 presents the results of testing for the presence of the unevenness defect in ETFSS and SIM-ETFSS [32]. The functions from Table 1 of dimension 30 were used for testing. Other values of the hyperparameters of the algorithm, except for the already mentioned $N = 5$ and $P = 0.2$, are the same as in Table 3.

Table 6. The results of testing the algorithm for the unevenness defect. The letter "C" denotes the cluster-based initialization of populations for each island in a separate region; the letter "R" denotes the classical approach to initial population generation.

Function	Values of the Algorithm Hyperparameters		
	N = 1, (R)	**N = 5, (C)**	**N = 5, (R)**
f_1	4.42×10^7	2.21×10^7	1.96×10^7
f_2	2.14×10^{31}	8.70×10^{30}	8.42×10^{30}
f_3	3.52×10^2	7.79×10^4	5.72×10^4
f_4	1.60×10^1	3.08×10^3	3.01×10^3
f_5	4.20×10^6	1.42×10^6	1.41×10^6
f_6	1.79×10^4	3.13×10^{12}	3.19×10^{12}
f_7	3.29×10^4	1.17×10^5	1.83×10^5
f_8	5.36×10^{-1} *	4.21×10^{-1} *	4.37×10^{-1} *
f_9	4.60×10^1	3.93×10^4	4.01×10^4
f_{10}	5.79×10^0	2.40×10^3	2.28×10^3
f_{11}	6.36×10^2	5.10×10^4	4.81×10^4
f_{12}	1.51×10^1	7.01×10^0	1.53×10^1
f_{13}	3.39×10^2	3.84×10^3	3.36×10^3
f_{14}	4.30×10^1	3.55×10^1	3.37×10^1
f_{15}	2.05×10^3	1.30×10^3	1.22×10^3
f_{16}	9.32×10^2	1.39×10^4	1.55×10^4
f_{17}	1.51×10^7	3.26×10^7	4.17×10^7
f_{18}	8.29×10^{-8} *	8.42×10^{-8} *	8.39×10^{-8} *
f_{19}	1.63×10^1	2.40×10^1	2.45×10^1
f_{20}	3.10×10^2	8.78×10^5	1.05×10^6

* Cases with a statistically significant decrease in optimization quality when the search area is reduced.

The values for all test functions f_q, where $q = \overline{1,20}$, shown in Table 6, are the ratios of the median results $\mathrm{UD}_{score}^{(q)}$ obtained by searching for an extremum in the original search area to the median results obtained by a similar search but in a disproportionately (unevenly) reduced search area. As expected, the values for the function f_8 (Schwefel 2.26) turned out to be less than one, since the extremum of this function, which is on the periphery of the search space, may not enter the bounded region when it is reduced. For the rest of the functions a significant increase in the quality of results is observed when the search area is reduced, which indicates that the considered algorithms are resistant to the unevenness defect. The only exception is the function f_{18} (Xin-She Yang), on which all the tested algorithms demonstrated instability to the unevenness defect.

4.3. The Solution of the Real Data Problem Using the SIM-ETFSS Algorithm

As an example of a real data problem, the diagnosis of breast cancer based on the numerical features extracted from images of a fine needle aspirate of a breast mass is considered. The dataset is available in the UCI Machine Learning Repository [62] and contains 569 records (objects) described by 30 features. The classification problem of determining whether a breast mass is malignant is considered.

An Extreme Learning Machine (ELM) model [63] is proposed to solve this problem. The ELM model is a single-hidden layer feedforward neural network (SLFN) with an input layer of K_{in} neurons, a hidden layer of K_{hid} neurons and an output layer of K_{out} neurons. The forward propagation of the model for a set of $\hat{N}$ objects is carried out as follows:

$$Y_{pred} = \sigma\left(X_{in}W_{in} + \overline{\beta}\right)W_{out}, \tag{18}$$

where $Y_{pred} \in \mathbb{R}^{\hat{N} \times K_{out}}$ denotes the matrix of values predicted by the model; σ denotes the sigmoidal activation function [63]; $X_{in} \in \mathbb{R}^{\hat{N} \times K_{in}}$ denotes the matrix of input values; $W_{in} \in \mathbb{R}^{K_{in} \times K_{hid}}$ denotes the input layer weight matrix; $\overline{\beta} \in \mathbb{R}^{K_{hid}}$ denotes the bias vector for the hidden layer; $W_{out} \in \mathbb{R}^{K_{hid} \times K_{out}}$ denotes the output layer weight matrix.

Any nonlinear activation function can be used instead of σ, but the sigmoidal function is the most commonly used. The key feature of this model is the way it is trained. For this purpose, the values of input layer weights W_{in} and the values of the hidden layer biases $\overline{\beta}$ are initialized with random numbers, and the matrix W_{out} is calculated as follows:

$$W_{out} = \left(\sigma\left(X_{in}W_{in} + \overline{\beta}\right)\right)^{\dagger} Y_{out}, \tag{19}$$

where $(\cdot)^{\dagger}$ denotes the Moore–Penrose pseudoinverse matrix calculation; $Y_{out} \in \mathbb{R}^{\hat{N} \times K_{out}}$ denotes the matrix of correct answers for the X_{in}.

Since the pseudoinverse matrix calculation is used instead of a gradient-based methods, ELM trains very fast and produces high-quality results. To evaluate the quality of the model according to Algorithm 4, the value of the estimation metric is calculated using 10-fold cross-validation and the F_1-score.

Algorithm 4: ELM model estimation metric.

Input: W_{in}—input weights;
$\overline{\beta}$—hidden layer biases.

1. **for** each fold in 10-fold cross-validation **do**:
2. Create a new ELM model and initialize it with W_{in} and $\overline{\beta}$;
3. Scale the train subset of the fold so that the mean is 0 and the standard deviation is 1, and apply this transformation to the test subset of the fold;
4. Train the ELM model on a scaled train subset of the fold using Formula (19);
5. Perform a forward propagation on a scaled test subset of the fold using Formula (18);
6. Calculate the F_1-score value of the result of forward propagation;
7. **end for**
8. Calculate the mean value of all F_1-score values obtained by cross-validation.

The size of the hidden layer K_{hid} is a hyperparameter of the ELM model and directly affects the quality of the results it produces [63], so different values of K_{hid} from the set $\{10, 20, 30, \ldots, 180, 190, 200\}$ were considered. For each value, we calculate 1000 scores using Algorithm 4 with random values for W_{in} and $\overline{\beta}$ taken from the range $[-1, 1]$. According to the results shown in Figure 14, the best value for K_{hid} that maximizes the estimation metric is $K_{hid} = 110$ (red dashed line).

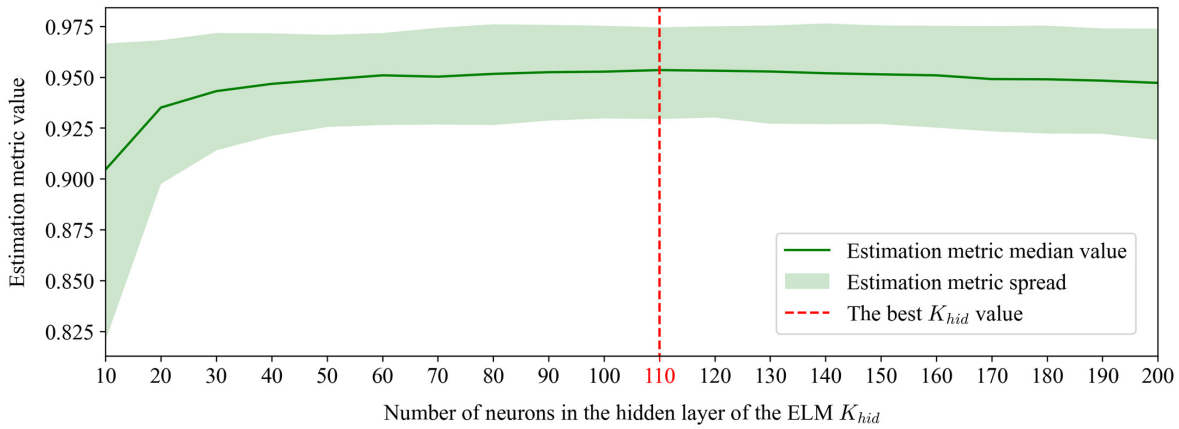

Figure 14. The results of considering different values of K_{hid}.

Thus, an ELM model with $K_{in} = 30$, $K_{hid} = 110$, $K_{out} = 1$ is proposed to solve the problem of the breast cancer diagnosis. The malignant class is labeled as 1, and the benign class is 0.

In order to improve the results produced by the ELM model, the SIM-ETFSS algorithm is proposed to optimize the values of input weights W_{in} and biases $\overline{\beta}$ similar to the work [42]. A small number of iterations and agents is enough to demonstrate the advantages of using the population-based algorithm. The chosen values of the SIM-ETFSS hyperparameters are presented in Table 7.

Table 7. SIM-ETFSS hyperparameter values.

Hyperparameter	Value
Number of islands N	3
Population size M	30
Number of iterations T	20
Probability P	0.2
Minimum island size L	2
Initial individual step length $step_{ind}^{(1)}$	0.5
Initial collective-volitive step length $step_{vol}^{(1)}$	0.25
Agent's maximum weight limitation w_{max}	5000
Initialization method	Cluster-based

The SIM-ETFSS algorithm is used to maximize the ELM estimation metric according to Algorithm 4. Each agent of the algorithm is a vector containing all values of W_{in} and $\overline{\beta}$, i.e., the dimensionality of the problem to be solved is $D = K_{in} \times K_{hid} + K_{hid} = 30 \times 110 + 110 = 3410$. The search area along each dimension is defined by the range $[-1, 1]$.

During $T = 20$ iterations, the SIM-ETFSS algorithm performs $M + T \times M \times 2 = 1230$ objective function evaluations (according to lines 4, 9, 20 of Algorithm 1), so for comparison, it was decided to set it against the random search algorithm, which estimates the ELM model 1230 times using Algorithm 4 with random values for W_{in} and $\overline{\beta}$ taken from the range $[-1, 1]$, and selects the best (maximum) result.

For the SIM-ETFSS algorithm and the random search algorithm, 20 independent runs each were performed, and the results are shown as box plots in Figure 15.

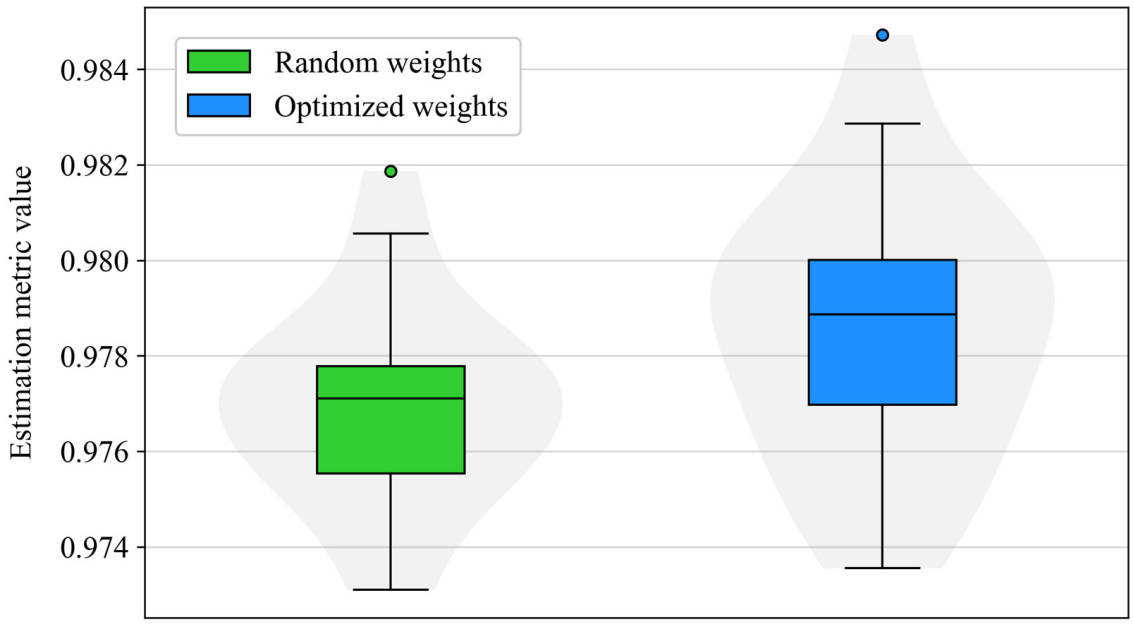

Figure 15. Comparison of SIM-ETFSS (blue) and random search model (green) based on the breast-cancer-diagnosis problem.

Figure 15 shows that even with a small number of iterations and agents, the use of the SIM-ETFSS algorithm is more advantageous than the random search algorithm, as it significantly improves the quality of the results produced. To further validate the obtained results, the Mann–Whitney U test with a significance level $\alpha = 0.05$ was applied, and as a result, the SIM-ETFSS algorithm was found to have statistically significant superiority over the random search algorithm.

5. Discussion

The improved Soft Island Model for the ETFSS algorithm presented in this paper implements a probabilistic approach for the implementation of agent migrations based on the gathering of statistics on the total island achievements in the context of optimizing the objective function at each t-th iteration of the algorithm. In this case, the topology of the migrations can be regulated by means of a hyperparameter P, as was shown when considering migration maps. To prevent the complete disappearance of islands, the hyperparameter L was provided during the optimization, specifying the minimum number of agents that should always be at one island.

In order to initialize the algorithm, a new approach to initial population generation was proposed, where each island is allocated its own independent space within a search area, within which its agents are generated. To compare this approach with the classical random initialization shown in Figure 2a, a statistical analysis method, such as the Mann–Whitney U test, was applied. It was used to calculate the number of situations for which there was an advantage of one method of initialization over the other among all the experiments conducted. The obtained results were grouped by different values of D, N, P and presented in the form of bar charts, which clearly showed not only the superiority of the cluster-based initialization proposed in this paper over the classical random approach, but also the correlation of the obtained results with some hyperparameters of the algorithm.

As part of the main experiments to analyze the performance of SIM-ETFSS compared to its island-free version ETFSS, different visualizations were proposed to show different aspects of the distribution statistics of the results. For the construction of some of the diagrams, data normalization was carried out to allow for a compact presentation of all the results for easy interpretation.

The SIM-ETFSS algorithm was also tested for known defects, such as the center-bias operator and the unevenness defect. Measures of central tendency (median, mean, geometric mean), as well as the previously mentioned Mann–Whitney U test, were used to detect them. As a result, it was concluded that neither the original ETFSS algorithm nor its island modification SIM-ETFSS proposed in this paper are subject to any of the mentioned defects.

As an example of applying SIM-ETFSS to a real data problem, a breast-cancer-diagnosis dataset was considered. It was found that the use of the presented algorithm can statistically significantly improve the performance of the ELM model and increase the classification quality.

According to the conducted statistical analysis, the SIM-ETFSS algorithm proposed in this paper has shown superiority over its island-free counterpart ETFSS, so it can be more effectively applied in solving optimization problems. Among the main advantages of the proposed algorithm are its high performance on large-scale problems, as well as the absence of a center-bias operator and the unevenness defect. Moreover, as demonstrated in the experiments, the hyperparameters of the SIM-ETFSS algorithm are highly interpretable, which simplifies the process of tuning the algorithm for specific problems. A limitation can be identified as the lack of convergence proofs, which is a common issue for all population-based optimization algorithms. However, when implementing our algorithm, we calculate estimates of the median and variance for the best values found based on the results of a number of independent runs of the algorithm and draw the corresponding conclusions. This is a common practice in the field of population-based optimization algorithms. Estimates for the median (or the mean) should be close to the corresponding values characterizing the known global extremum (in the case where optimization experiments are carried out for a known test function). If information about the global extremum is unknown, it is customary to focus on estimates for variances. The calculated estimates for variances should be minimal. It should be noted that there is a large set of tools used to estimate the convergence of population-optimization algorithms [31].

One of the significant contributions of this work is the high performance of the presented algorithm based on high-dimensional problems. This makes it possible to use it to choose parameter values for neural networks and other machine learning models instead of classical gradient-based approaches. Since population-based algorithms consider the entire available search space without

prioritizing certain dimensions, the algorithm proposed in this paper is not subject to such well-known problems as an explosive and vanishing gradient, which can be especially useful for training multilayer neural networks.

For example, ref. [64] discusses the use of population-based optimization algorithms to train a recurrent neural network (RNN) to solve the time series-based volatility forecasting problem. The authors note that the use of gradient-based methods negatively affects the model's ability to account for long-term dependencies. The application of population-based algorithms, according to their results, leads to improved model performance in the considered task.

As mentioned earlier, the need for optimization in one form or another arises in a wide range of problems from different spheres of activity, so the practical contribution of this study is to expand the range of methods for solving different problems.

Further research can be aimed at improving the SIM-ETFSS algorithm proposed in this paper, as well as at a comparative analysis of other island models as applied to ETFSS. The model presented in this paper assumes that migrations happen at each iteration of the optimization algorithm. However, it makes sense to consider the possibility of the implementation of periodic migrations, which will be carried out only once after some predetermined number of iterations (period). For this purpose, it is necessary to develop a mechanism for accumulating information about the quality of the work performed by each island, so that it could be used to calculate probability values for each agent to either migrate or stay on its island. The accumulation of information can be implemented, for example, by computing the cumulative change in the weights of all agents over the period. In addition, more sophisticated approaches can be considered, such as computing a weighted average adjusted so that changes in agents' weights in the last iterations of the period contribute more (or, conversely, less) to the overall island score.

In order to increase competition between islands, it also makes sense to consider using different combinations of hyperparameter values for agents from different islands. This will allow, for example, to increase the performance of some islands relative to others when passing certain places on the surface of the objective function or upon reaching a certain number of iterations, taking into account that the maximum step lengths for individual and collective-volitive movements of the agents of the SIM-ETFSS algorithm change over time.

The cluster-based initialization of populations presented in this paper was implemented using a pseudo-random number generator based on chaotic tent-maps, naturally developing the concept of the original ETFSS algorithm. However, this approach does not guarantee a high degree of coverage of the entire solution space, especially for a large number of islands, for which clusters may be much smaller than the original search area. It is worthwhile to pay attention to this feature in future research and consider other methods of pseudorandom number generation that provide better coverage of the search space, for example, the Latin hypercube [65] or Sobol' [66] methods.

Since the presented SIM-ETFSS algorithm has shown good results when optimizing classical test functions, it makes sense to test it on more complex problems, e.g., CEC [53,54].

Author Contributions: Conceptualization, guidance, supervision, validation L.A.D., software, resources, visualization, testing, V.E.Z.; original draft preparation, L.A.D. and V.E.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data used in the example application of the algorithm is openly available in [62].

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Hotta, S.; Kiyasu, S.; Miyahara, S. Pattern recognition using average patterns of categorical k-nearest neighbors. In Proceedings of the 17th International Conference on Pattern Recognition, Cambridge, UK, 26 August 2004; IEEE: New York, NY, USA, 2004; Volume 4, pp. 412–415.
2. Dalal, N.; Triggs, B. Histograms of Oriented Gradients for Human Detection. In Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), San Diego, CA, USA, 20–25 June 2005; IEEE: New York, NY, USA, 2005; Volume 1, pp. 886–893.
3. Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; et al. ImageNet Large Scale Visual Recognition Challenge. *Int. J. Comput. Vis.* **2014**, *115*, 211–252. [CrossRef]
4. Meng, Q. Application of machine learning in medicine. *Appl. Comput. Eng.* **2024**, *33*, 207–212. [CrossRef]
5. Srinivasaiah, B. The Power of Personalized Healthcare: Harnessing the Potential of Machine Learning in Precision Medicine. *Int. J. Sci. Res.* **2024**, *13*, 426–429. [CrossRef]
6. Gramovich, I.V.; Musatov, D.Y.; Petrusevich, D.A. Implementation of bagging in time series forecasting. *Russ. Technol. J.* **2024**, *12*, 101–110. [CrossRef]
7. Aïmeur, E.; Amri, S.; Brassard, G. Fake news, Disinformation and Misinformation in Social Media: A Review. *Soc. Netw. Anal. Min.* **2023**, *13*, 30. [CrossRef] [PubMed]
8. Wang, K.; Abid, M.A.; Rasheed, A.; Crossa, J.; Hearne, S.; Li, H. DNNGP, a deep neural network-based method for genomic prediction using multi-omics data in plants. *Mol. Plant* **2023**, *16*, 279–293. [CrossRef]
9. Wang, Y.; Moradi, R.; Haghighi, M.H.Z.; Rastegarnia, F. Introduction of machine learning for astronomy (hands-on workshop). *Astron. Astrophys. Trans.* **2022**, 337–346. [CrossRef]
10. Joshi, P.B. Navigating with chemometrics and machine learning in chemistry. *Artif. Intell. Rev.* **2023**, *56*, 9089–9114. [CrossRef]
11. Singh, S.P.; Yadav, D.K.; Chamran, M.K.; Perera, D.G. Intelligent mutation based evolutionary optimization algorithm for genomics and precision medicine. *Funct. Integr. Genom.* **2024**, *24*, 128. [CrossRef]
12. Zhang, J. Optimization design of highway route based on deep learning. *Front. Future Transp.* **2024**, *5*, 1430509. [CrossRef]
13. Boronnikov, A.S.; Tsyngalev, P.S.; Ilyin, V.G.; Demenkova, T.A. Evaluation of connection pool PgBouncer efficiency for optimizing relational database computing resources. *Russ. Technol. J.* **2024**, *12*, 7–24. [CrossRef]
14. Dasgupta, S.; Sen, J. A Comparative Study of Hyperparameter Tuning Methods. *arXiv* **2024**, arXiv:2408.16425.
15. Wu, J.; Chen, X.Y.; Zhang, H.; Xiong, L.D.; Lei, H.; Deng, S.H. Hyperparameter Optimization for Machine Learning Models Based on Bayesian Optimization. *J. Electron. Sci. Technol.* **2019**, *17*, 26–40.
16. Hertel, L.; Collado, J.; Sadowski, P.; Ott, J.; Baldi, P. Sherpa: Robust hyperparameter optimization for machine learning. *SoftwareX* **2020**, *12*, 100591. [CrossRef]
17. Demidova, L.A.; Gorchakov, A.V. A Study of Biology-inspired Algorithms Applied to Long Short-Term Memory Network Training for Time Series Forecasting. In Proceedings of the 2021 3rd International Conference on Control Systems, Mathematical Modeling, Automation and Energy Efficiency (SUMMA), Lipetsk, Russia, 10–12 November 2021; IEEE: New York, NY, USA, 2021; pp. 473–478.
18. Romadloni, M.I.H.; Saputro, D.R.S. Backpropagation neural network weight training using particle swarm optimization. In *Proceedings of the 6th National Conference on Mathematics and Mathematics Education*; AIP Publishing: Semarang, Indonesia, 2022; Volume 2577, p. 020056.
19. Xu, C.; Zhang, J. A Survey of Quasi-Newton Equations and Quasi-Newton Methods for Optimization. *Ann. Oper. Res.* **2001**, *103*, 213–234. [CrossRef]
20. Cagan, J.; Shimada, K.; Yin, S. A survey of computational approaches to three-dimensional layout problems. *Comput. Aided Des.* **2002**, *34*, 597–611. [CrossRef]
21. Singh, A.; Sharma, S.; Singh, J. Nature-Inspired Algorithms for Wireless Sensor Networks: A Comprehensive Survey. *arXiv* **2021**, arXiv:2101.10453. [CrossRef]
22. Li, X.; Hua, S.; Liu, Q.; Li, Y. A partition-based convergence framework for population-based optimization algorithms. *Inf. Sci.* **2023**, *627*, 169–188. [CrossRef]
23. Wolpert, D.; Macready, W. No free lunch theorems for optimization. *IEEE Trans. Evol. Comput.* **1997**, *1*, 67–82. [CrossRef]
24. Demidova, L.A.; Zhuravlev, V.E. Novel Four-stage Comprehensive Analysis Approach for Population-based Optimization Algorithms. In Proceedings of the 2023 5th International Conference on Control Systems, Mathematical Modeling, Automation and Energy Efficiency (SUMMA), Lipetsk, Russia, 8–10 November 2023; IEEE: New York, NY, USA, 2023; pp. 263–268.
25. Kudela, J. The Evolutionary Computation Methods No One Should Use. *arXiv* **2023**, arXiv:2301.01984.
26. Velasco, L.; Guerrero, H.; Hospitaler, A. A Literature Review and Critical Analysis of Metaheuristics Recently Developed. *Arch. Comput. Methods Eng.* **2023**, *31*, 125–146. [CrossRef]
27. Demidova, L.A.; Gorchakov, A.V. A Study of Chaotic Maps Producing Symmetric Distributions in the Fish School Search Optimization Algorithm with Exponential Step Decay. *Symmetry* **2020**, *12*, 784. [CrossRef]

28. Akhmedova, S.; Stanovov, V.; Semenkin, E. Soft Island Model for Population-Based Optimization Algorithms. In *Advances in Swarm Intelligence*; Springer International Publishing: Berlin/Heidelberg, Germany, 2018; pp. 68–77.
29. Bilal; Pant, M.; Zaheer, H.; Garcia-hernandez, L.; Abraham, A. Differential Evolution: A review of more than two decades of research. *Eng. Appl. Artif. Intell.* **2020**, *90*, 103479. [CrossRef]
30. Stanovov, V.; Semenkin, E. Surrogate-Assisted Automatic Parameter Adaptation Design for Differential Evolution. *Mathematics* **2023**, *11*, 2937. [CrossRef]
31. Halim, A.H.; Ismail, I.; Das, S. Performance assessment of the metaheuristic optimization algorithms: An exhaustive review. *Artif. Intell. Rev.* **2021**, *54*, 2323–2409. [CrossRef]
32. Zhuravlev, V.E. Study of the Influence of the Search Area Shape on the Performance of Population-based Optimization Algorithms. *IT Standard* **2024**, *2*, 24–31. (In Russian)
33. Karaboga, D.; Basturk, B. On the performance of artificial bee colony (ABC) algorithm. *Appl. Soft Comput.* **2008**, *8*, 687–697. [CrossRef]
34. Chen, B.; Zhao, L.; Lu, J.H. Wind power forecast using RBF network and culture algorithm. In Proceedings of the 2009 International Conference on Sustainable Power Generation and Supply, Nanjing, China, 6–7 April 2009; IEEE: New York, NY, USA, 2009; pp. 1–6.
35. Storn, R.; Price, K. Differential Evolution—A Simple and Efficient Heuristic for global Optimization over Continuous Spaces. *J. Glob. Optim.* **1997**, *11*, 341–359. [CrossRef]
36. Whitley, D. A genetic algorithm tutorial. *Stat. Comput.* **1994**, *4*, 65–85. [CrossRef]
37. Mitchell, M.; Holland, J.H.; Forrest, S. When will a Genetic Algorithm Outperform Hill Climbing. In *Neural Information Processing Systems*; Morgan Kaufmann Publishers, Inc.: San Francisco, CA, USA, 1993.
38. Moscato, P. *On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms*; Technical Report C3P 826, Caltech Con-Current Computation Program 158–79; California Institute of Technology: Pasadena, CA, USA, 1989.
39. Kennedy, J.; Eberhart, R. Particle swarm optimization. In Proceedings of the ICNN'95—International Conference on Neural Networks, Perth, WA, Australia, 27 November–1 December 1995; IEEE: New York, NY, USA, 2002; Volume 4, pp. 1942–1948.
40. Mirjalili, S. The Ant Lion Optimizer. *Adv. Eng. Softw.* **2015**, *83*, 80–98. [CrossRef]
41. Bastos Filho, C.J.A.; de Lima Neto, F.B.; Lins, A.J.; Nascimento, A.I.; Lima, M.P. A novel search algorithm based on fish school behavior. In Proceedings of the 2008 IEEE International Conference on Systems, Man and Cybernetics, Singapore, 12–15 October 2008; IEEE: New York, NY, USA, 2009; pp. 2646–2651.
42. Demidova, L.A.; Gorchakov, A.V. Biology-inspired optimization algorithms applied to intelligent input weights selection of an extreme learning machine in regression problems. In *VII International Conference "Safety Problems of Civil Engineering Critical Infrastructures" (SPCECI2021)*; AIP Publishing: Krasnoyarsk, Russia, 2023; Volume 2700, p. 030003.
43. Demidova, L.A.; Gorchakov, A.V. Classification of Program Texts Represented as Markov Chains with Biology-Inspired Algorithms-Enhanced Extreme Learning Machines. *Algorithms* **2022**, *15*, 329. [CrossRef]
44. Duarte, G.R.; de Castro Lemonge, A.C.; da Fonseca, L.G.; de lima, B.S.L.P. An Island Model based on Stigmergy to solve optimization problems. *Nat. Comput.* **2020**, *20*, 413–441. [CrossRef]
45. Tismer, A.; Riedelbauch, S. Optimization of a diffuser augmented kinetic turbine with the island model. *IOP Conf. Ser. Earth Environ. Sci.* **2024**, *1411*, 012054. [CrossRef]
46. Wolpert, D.; Macready, W. Coevolutionary Free Lunches. *IEEE Trans. Evol. Comput.* **2005**, *9*, 721–735. [CrossRef]
47. Ali, O.; Abbas, Q.; Mahmood, K.; Bautista thompson, E.; Arambarri, J.; Ashraf, I. Competitive Coevolution-Based Improved Phasor Particle Swarm Optimization Algorithm for Solving Continuous Problems. *Mathematics* **2023**, *11*, 4406. [CrossRef]
48. Sun, Y.; Xu, P.; Zhang, Z.; Zhu, T.; Luo, W. Brain Storm Optimization Integrated with Cooperative Coevolution for Large-Scale Constrained Optimization. In *Advances in Swarm Intelligence*; Springer Nature: Cham, Switzerland, 2023; pp. 356–368.
49. Ghasemi, M.; Akbari, E.; Rahimnejad, A.; Razavi, S.E.; Ghavidel, S.; Li, L. Phasor particle swarm optimization: A simple and efficient variant of PSO. *Soft Comput.* **2018**, *23*, 9701–9718. [CrossRef]
50. Shi, Y. Brain Storm Optimization Algorithm. In *Advances in Swarm Intelligence*; Springer: Berlin/Heidelberg, Germany, 2011; Volume 6728, pp. 303–309.
51. Stanovov, V.; Kazakovtsev, L.; Semenkin, E. Hyper-Heuristic Approach for Tuning Parameter Adaptation in Differential Evolution. *Axioms* **2024**, *13*, 59. [CrossRef]
52. Stanovov, V.; Semenkin, E. Adaptation of the Scaling Factor Based on the Success Rate in Differential Evolution. *Mathematics* **2024**, *12*, 516. [CrossRef]
53. Awad, N.; Ali, M.; Liang, J.; Qu, B.; Suganthan, P. *Problem Definitions and Evaluation Criteria for the CEC 2017 Special Session and Competition on Single Objective Bound Constrained Real-Parameter Numerical Optimization*; Nanyang Technological University: Singapore, 2016; Technical Report.

54. Kumar, A.; Price, K.; Mohamed, A.K.; Suganthan, P.N. *Problem Definitions and Evaluation Criteria for the CEC 2022 Special Session and Competition on Single Objective Bound Constrained Numerical Optimization*; Nanyang Technological University: Singapore, 2021; Technical Report.
55. Ruciński, M.; Izzo, D.; Biscani, F. On the impact of the migration topology on the Island Model. *Parallel Comput.* **2010**, *36*, 555–571. [CrossRef]
56. Frahnow, C.; Kötzing, T. Ring Migration Topology Helps Bypassing Local Optima. *arXiv* **2018**, arXiv:1806.01128.
57. Homayounfar, H.; Areibi, S.; Wang, F. An advanced island based ga for optimization problems. *Dyn. Contin. Discret. Impuls. Syst. Ser. B Appl. Algorithms* **2003**, 46–51.
58. Kwak, J.W.; Jhon, C.S. Torus Ring: Improving performance of interconnection network by modifying hierarchical ring. *Parallel Comput.* **2007**, *33*, 2–20. [CrossRef]
59. Fernández, F.; Tomassini, M.; Vanneschi, L. An Empirical Study of Multipopulation Genetic Programming. *Genet. Program. Evolvable Mach.* **2003**, *4*, 21–51. [CrossRef]
60. Plevris, V.; Solorzano, G. A Collection of 30 Multidimensional Functions for Global Optimization Benchmarking. *Data* **2022**, *7*, 46. [CrossRef]
61. Jamil, M.; Yang, X.S. A literature survey of benchmark functions for global optimisation problems. *Int. J. Math. Model. Numer. Optim.* **2013**, *4*, 150. [CrossRef]
62. Wolberg, W.; Mangasarian, O.; Street, N.; Street, W. Breast Cancer Wisconsin (Diagnostic)—UCI Machine Learning Repository. Available online: https://doi.org/10.24432/C5DW2B (accessed on 10 January 2025).
63. Huang, G.; Zhu, Q.; Siew, C. Extreme learning machine: Theory and applications. *Neurocomputing* **2006**, *70*, 489–501. [CrossRef]
64. Lim, B.; Zohren, S.; Roberts, S. Population-based Global Optimisation Methods for Learning Long-term Dependencies with RNNs. *arXiv* **2019**, arXiv:1905.09691.
65. Mckay, M.D.; Beckman, R.J.; Conover, W.J. A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code. *Technometrics* **2000**, *42*, 55–61. [CrossRef]
66. Sobol', I. On the distribution of points in a cube and the approximate evaluation of integrals. *USSR Comput. Math. Math. Phys.* **1967**, *7*, 86–112. [CrossRef]