A Hybrid Multistage Framework for Numerical Optimization based on ensembling Multiple Nature-inspired Algorithms

Doan Duy Tung
Hanoi University of Science and Technology
Hanoi, Vietnam
tung.dd224906@sis.hust.edu.vn

Nguyen Viet Tuan Kiet
Hanoi University of Science and Technology
Hanoi, Vietnam
kiet.nvt220032@sis.hust.edu.vn

Nguyen Thi Ha Chi
Hanoi University of Science and Technology
Hanoi, Vietnam
chi.nth220016@sis.hust.edu.vn

Bui Dinh Pham
Hanoi University of Science and Technology
Hanoi, Vietnam
pham.bd220039@sis.hust.edu.vn

Dao Van Tung
Hanoi University of Science and Technology
Hanoi, Vietnam
tung.dv204932@sis.hust.edu.vn

Huynh Thi Thanh Binh
Hanoi University of Science and Technology
Hanoi, Vietnam
binhht@soict.hust.edu.vn

ABSTRACT

In this work, we propose a novel hybrid optimization algorithm that combines state-of-the-art techniques from evolutionary computation and derivative-free optimization. The algorithm operates in two main stages to effectively balance exploration and exploitation of the search space. In the first stage, a population-based approach is employed using Differential Evolution (DE) and its variants, enhanced by local search methods such as L-BFGS-B and Multiple Trajectory Search - Local Search 1 (MTS-LS1). This stage also integrates Gaussian mutation to ensure diversity within the population. The second stage focuses on applying swarm intelligence, specifically the Salp Swarm Algorithm (SSA), to further refine the solutions and avoid local optima. This two-stage approach is designed to handle complex optimization problems with high precision and robustness. Extensive experiments on a variety of benchmark functions demonstrate the performance of our method in reaching the optimal point while minimizing number of Function Evaluation (FE). The results highlight the effectiveness of integrating local search and swarm intelligence in a hybrid framework, paving the way for advanced optimization techniques in real-world applications.

KEYWORDS

Evolutionary Computation, Swarm Intelligence, Differential Evolution, Global Optimization, Derivative-free Optimization, Nature-inspired Algorithm, Hybridization

Conference'17, July 2017, Washington, DC, USA
© 2024 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

ACM Reference Format:
Doan Duy Tung, Nguyen Viet Tuan Kiet, Nguyen Thi Ha Chi, Bui Dinh Pham, Dao Van Tung, and Huynh Thi Thanh Binh. 2024. A Hybrid Multistage Framework for Numerical Optimization based on ensembling Multiple Nature-inspired Algorithms . In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 TEAM INFORMATION:

- **Team Name:** MSOLab.
- **Team Leader:** Huynh Thi Thanh Binh.
- **Team Member:** Doan Duy Tung, Nguyen Viet Tuan Kiet, Nguyen Thi Ha Chi, Bui Dinh Pham, Dao Van Tung.
- **Primary Affiliation:** Hanoi University of Science and Technology.

2 INTRODUCTION

Optimization of black-box functions is a critical area in various fields such as engineering, finance, machine learning, and scientific research. These functions are typically characterized by their complex, non-linear, and non-convex nature, where the analytical form is unknown or inaccessible, and evaluations are expensive. The challenge in black-box optimization lies in efficiently exploring the search space to find optimal solutions while minimizing the number of function evaluations. Traditional gradient-based optimization techniques are not applicable due to the lack of derivative information. Therefore, derivative-free optimization methods have become essential for tackling such problems.

In recent years, evolutionary computation and derivative-free optimization have emerged as powerful paradigms for tackling complex optimization problems. Evolutionary algorithms, such as Differential Evolution (DE) [1], has been widely adopted and further enhanced through the incorporation of various strategies and modifications, such as Improved Multi-Operator Differential Evolution (IMODE) [2]. On the other hand, derivative-free optimization techniques, including local search methods like L-BFGS-B and Multiple Trajectory Search - Local Search 1 (MTS-LS1), excel in fine-tuning solutions by exploiting local information. However, these single methods often suffer from premature convergence and limited diversity within the population. Therefore, they fall short in providing reliable results for a wide range of test problem [3].

Swarm intelligence algorithms, such as the Salp Swarm Algorithm (SSA), have also gained traction due to their simplicity and effectiveness in exploring the search space. SSA mimics the collective behavior of salps in the ocean, balancing exploration and exploitation through social interactions. Despite their individual successes, integrating evolutionary computation, local search methods, and swarm intelligence into a cohesive framework remains an ongoing research challenge.

This work proposes Multistage Hybrid Optimizer (MSHO) algorithm that combines state-of-the-art techniques from evolutionary computation and derivative-free optimization to effectively balance exploration and exploitation of the search space. The primary objective is to develop an algorithm capable of solving complex black-box optimization problems with high precision and robustness. The proposed algorithm operates in three main stages:

- **Initial Stage:** Apply Simulated Binary Crossover (SBX) to enhance population diversity.
- **Intermediate Stage:** Use Differential Evolution (DE) with superior operators to optimize real-valued functions.
- **Advanced Stage:** Integrate advanced local search techniques to escape local optima and improve solution quality.

Moreover, we also implement a **Standby Stage:** algorithm SSA with a dynamic population size immediately following the conclusion of the preceding search. This approach is employed to provide a final opportunity to attain a global solution.

This work is organized as follows: the problem and proposed algorithm is described in section 3 while section 4 provides experiment setup and results of our extensive experiments.

3 PROPOSED METHOD

3.1 Problem Formulation

We consider solving the 24 functions of the competition [3] as solving the global optimization problem with box constraints. The objective is to find the global minimum of the function within the given search space. The search space is defined by the lower bound $\mathbf{L} = (l_1, l_2, \ldots, l_n) \in \mathbb{R}^n$ and the upper bound $\mathbf{U} = (u_1, u_2, \ldots, u_n) \in \mathbb{R}^n$ of the solution vector $\mathbf{x} = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$ where n is the dimension and $l_i \leq x_i \leq u_i, \forall i = 1, 2, \ldots, n$. The optimization problem can be formulated as follows:

$$\mathbf{x}^* = \arg\min_{[\mathbf{L},\mathbf{U}]} f(\mathbf{x}) \tag{*}$$

The functions are designed to challenge various aspects of optimization algorithms, including convergence speed, precision, and robustness against local minima.

3.2 General Framework

In this section, we present the general framework of our proposed hybrid optimization algorithm. This algorithm integrates various state-of-the-art techniques from evolutionary computation and derivative-free optimization, providing a robust and efficient approach to solving complex optimization problems. The proposed method operates in multiple stages, each employing different strategies to explore and exploit the search space effectively. Detailed steps of the algorithm unfolds through a sequence of well-defined stages:

Algorithm 1: Multistage Hybrid Optimizer

```
Input:
- Maximum number of function evaluations $E_{\max}$
- Initial population size: $N$
- Number of DE evaluations: $DE\_evals$
- Number of operator: $M$
- Local search probability: $P_{LS}$
- L-BFGS-B gradient threshold: $\varepsilon$
- L-BFGS-B fitness difference threshold: $\Delta$
Output:
- optima value: $p^*$
- number of function evaluations to reach threshold: $FE$
Initialize population $|\mathcal{P}^0|$
$use\_onemore\_LBFGSB \leftarrow True$
while stopping criteria are not met do
    $G \leftarrow 0$
    $pool \leftarrow \emptyset$
    while $|pool| < |\mathcal{P}^G|$ do
        $q_1, q_2 = sample(\mathcal{P}^G, 2)$
        $o_1, o_2 = Crossover(q_1, q_2)$   ▷ SBX crossover is used
        $pool \leftarrow pool \cup \{o_1, o_2\}$
    $\mathcal{P}^G \leftarrow selection(\mathcal{P}^G \cup pool)$   ▷ $|\mathcal{P}^G|$ best individuals
    $subpops \leftarrow divideSubpopulations(\mathcal{P}^G)$
    $operators \leftarrow assignOperators(M)$
    for $i \leftarrow 1$ to $|subpops|$ do
        $subpop_i \leftarrow operator_i(subpop)$
    $\mathcal{P}^{\mathcal{G}} \leftarrow IMODE(\mathcal{P}^{\mathcal{G}}, DE\_evals)$   ▷ See subsection 3.3
    $op \leftarrow$ best individuals in $(\mathcal{P}^G)$
    $LS \leftarrow chooseBestLocalSearch()$   ▷ See subsection 3.4
    $op \leftarrow$localSearch($op$)   ▷ See algorithm 3
    if $use\_onemore\_LBFGSB = True$ then
        $rnd \leftarrow \mathcal{U}(1, |\mathcal{P}^G|)$
        $o \leftarrow \mathcal{P}^G_{rnd}$
        $o \leftarrow LBFGSB(o)$
        if $f(o) < f(op)$ then
            $op \leftarrow o$
        $use\_onemore\_LBFGSB \leftarrow$Update($\varepsilon, \Delta, \mathcal{P}^G, o$)   ▷ See algorithm 2
    $p^* \leftarrow op$
    $G \leftarrow G + 1$
    if $FE > 500000$ then
        $pa \leftarrow ASSA()$   ▷ See subsection 3.5
        $p^* \leftarrow \min(pa, p^*)$
return $p^*, E$
```

3.2.1 *Initial Stage: SBX Crossover.*

In the initial stage, the algorithm applies the Simulated Binary Crossover (SBX) [4] operator to the population to enhance diversity. Our experiments indicate that SBX

Algorithm 2: Update

```
Input:
- Gradient threshold: $\varepsilon$
- Fitness difference threshold: $\Delta$
- Population: $\mathcal{P}$
- Individual after applying L-BFGS-B: $o$
Output:
- Updating L-BFGS-B or not: $update$
$grad \leftarrow$ estimated gradient of $\mathcal{P}$ on $f$
$op \leftarrow$ best individuals in $\mathcal{P}$
$ratio \leftarrow \frac{o-op}{op}$
if $grad > \varepsilon \vee ratio > \Delta$ then
    $update \leftarrow False$
else
    $update \leftarrow True$
return $update$
```

is highly effective in introducing significant local-level mutations. These mutations, while not overly large or biased towards specific directions, facilitate gradual yet effective exploration, allowing individuals to adapt smoothly within the search space.

3.2.2 Intermediate Stage: DE with Superior Operators. Following the crossover, we incorporate a strategy that combines mixed superior operators from DE. These operators are particularly adept at optimizing real-valued functions, leveraging differentiation-like formulas without the need for derivatives. By increasing the oscillation level within the population, these algorithms address the complexities of the search environment, enhancing the algorithm's exploratory capabilities.

3.2.3 Advanced Stage: Local Search Techniques. Despite the effectiveness of DE, relying solely on it in large search spaces can lead to stagnation and difficulty in progressing towards optimal solutions. To mitigate this, we integrate advanced local search techniques aimed at escaping local optima and achieving higher-quality solutions. Inspired by the success of derivative-free optimization methods, we employ direction-set methods and quadratic matrix reduction algorithms. These methods provide intelligent and timely interventions, driving the search process towards more promising regions of the solution space.

This multi-stage approach demonstrates the strong potential and effective synergy between evolutionary computation and other optimization fields.

3.3 Population initialization and updating Mechanism

The population $\mathcal{P}$ is a set of $|\mathcal{P}|$ individuals, each representing the solution vectors. Let us denote the population at the generation G as $\mathcal{P}^G = \left\{\mathbf{x}_1^G, \mathbf{x}_2^G, \ldots, \mathbf{x}_{|\mathcal{P}^G|}^G\right\}$ where $\mathbf{x}_i^G \in \mathbb{R}^n$ is the i-th individual in the population at generation G. The population is updated at each generation based on the mutation, crossover and selection processes.

At the beginning of the optimization phase, the population is initialized by randomly sampling the solution vectors from the search space. The solution vectors are generated as follows:

$$\mathbf{x}_i^0 = \mathbf{L} + (\mathbf{U} - \mathbf{L}) \odot \mathbf{r}_i \quad \forall i = 1, 2, \ldots, |\mathcal{P}^0| \tag{1}$$

where $\mathbf{r}_i$ is a vector of random numbers in the interval $[0, 1]$, and $\odot$ denotes the element-wise multiplication.

During the optimization process, the population size is linearly decreased from $|\mathcal{P}^0|$ to $|\mathcal{P}^{\text{final}}|$ over the generations, based on the number of function evaluations. The population size is updated as follows:

$$|\mathcal{P}^G| = |\mathcal{P}^0| - \left\lfloor \frac{E^G}{E_{\max}} \cdot \left(|\mathcal{P}^0| - |\mathcal{P}^{\text{final}}|\right) \right\rfloor \tag{2}$$

where $|\mathcal{P}^0|$ is the initial population size, $|\mathcal{P}^{\text{final}}|$ is the final population size, E^G is the number of function evaluations used in previous G generations, and $E_{\max}$ is the maximum number of function evaluations.

3.3.1 Mutation Operator. The mutation operator is used to explore the search space by perturbing the solution vectors. In this algorithm, we use three mutation operators to generate new individuals. The mutation operators are as follows:

- DE/current-to-ϕ best/1 without archive:

$$\mathbf{v}_i = \mathbf{x}_i + F_i \cdot (\mathbf{x}_\phi - \mathbf{x}_i) + F_i \cdot (\mathbf{x}_{r_1} - \mathbf{x}_{r_2}) \tag{3.1}$$

- DE/current-to-ϕ best/1 with archive:

$$\mathbf{v}_i = \mathbf{x}_i + F_i \cdot (\mathbf{x}_\phi - \mathbf{x}_i) + F_i \cdot (\mathbf{x}_{r_1} - \mathbf{x}_{r_3}) \tag{3.2}$$

- DE weighted-rand-to-ϕ best:

$$\mathbf{v}_i = F_i \cdot \mathbf{x}_{r_1} + (x_\phi - x_{r_1}) \tag{3.3}$$

where $\mathbf{x}_i$ is the current individual, $\mathbf{x}_\phi$ is chosen from the $\phi \cdot |\mathcal{P}^G|$ best individuals, $\mathbf{x}_{r_1}$ and $\mathbf{x}_{r_2}$ are randomly chosen individuals, $\mathbf{x}_{r_3}$ is a randomly chosen individual from the union of the current population and the archive, and F_i is the scaling factor of i-th individual. An archive is used to maintain diversity in the population. If an individual is worse than their offspring, it is added to the archive. To make space for the new individual, the worst individual in the archive is removed if the size of the archive exceeds a certain threshold.

Each mutation operator has its own bias towards exploration and exploitation. The DE/current-to-ϕ best/1 mutation operator is biased towards exploitation, as it generates the offspring by combining the current individual with the best individual. The DE weighted-rand-to-ϕ best mutation operator is biased towards exploration, as it generates the offspring by combining a randomly chosen individual with the best individual. The DE/current-to-ϕ best/1 with archive mutation operator is a combination of the other two mutation operators, as it generates the offspring by combining the current individual with the best individual and a randomly chosen individual from the archive.

It is necessary to have a smart usage of these mutation operators that can combine advantages to balance exploration and exploitation. Inspired by IMODE algorithm [2], we use a dynamic switching mechanism to select the mutation operator.

We divide the population into three subpopulations, each of which is optimized using a different mutation operator.

The whole population $\mathcal{P} = \mathcal{P}_1 \cup \mathcal{P}_2 \cup \mathcal{P}_3$ such that $\mathcal{P}_1 \cap \mathcal{P}_2 \cap \mathcal{P}_3 = \emptyset$, where $\mathcal{P}_i$ is the subpopulation optimized using specific mutation operators called op_i.

The diversity obtained from a mutaion operator is calculated by the mean deviation of the population from the best individual, which is defined as follows:

$$D_{op_i} = \frac{1}{|\mathcal{P}_i|} \sum_{\mathbf{x} \in \mathcal{P}_i} \left\| \mathbf{x} - \mathbf{x}^{\text{best}_i} \right\| \quad \forall i = 1, 2, 3 \tag{4.1}$$

where $\mathbf{x}^{\text{best}_i}$ is the best individual in the subpopulation $\mathcal{P}_i$, and $\|\cdot\|$ is the Euclidean norm. The diversity ratio is defined as follows:

$$DR_{op_i} = \frac{D_{op_i}}{\sum_{i=1}^{3} D_{op_i}} \quad \forall i = 1, 2, 3 \tag{4.2}$$

Similarly, the performance ratio is defined as follows:

$$PR_{op_i} = \frac{f(\mathbf{x}_{op_i}^{\text{best}})}{\sum_{i=1}^{3} f(\mathbf{x}_{op_i}^{\text{best}})} \quad \forall i = 1, 2, 3 \tag{4.3}$$

Summarizing, the improvement ratio is defined as follows:

$$IR_{op_i} = (1 - PR_{op_i}) + DR_{op_i} \quad \forall i = 1, 2, 3 \tag{4.4}$$

Finally, the number of solutions generated by each mutation operator is defined as follows:

$$|\mathcal{P}_i| = \max\left\{0.1, \min\left\{0.9, \frac{IR_{op_i}}{\sum_{i=1}^{3} IR_{op_i}}\right\}\right\} \cdot |\mathcal{P}| \quad \forall i = 1, 2, 3 \tag{4.5}$$

3.3.2 Crossover Operator. After the mutation operator generates the offspring, the crossover operator is used to combine the parent and offspring to generate the trial vector. We choose the binomial crossover operator, which is defined as follows:

$$\mathbf{u}_{i,j} = \begin{cases} \mathbf{v}_{i,j} & \text{if rand} \le C_i \text{ or } j = j_{\text{rand}} \\ \mathbf{x}_{i,j} & \text{otherwise} \end{cases} \tag{5}$$

where $\mathbf{v}_i$ is the offspring generated by the mutation operator, $\mathbf{x}_i$ is the current individual, j is the dimension index, C_i is the crossover probability of the i-th individual, and j_{rand} is a random integer in the interval $[1, n]$. The crossover operator ensures that the offspring inherits the good features from the parent.

3.3.3 Selection Operator. The selection operator is used to select the individuals that will be included in the next generation. We use the greedy selection operator, which is defined as follows:

$$\mathbf{x}_i^{G+1} = \begin{cases} \mathbf{u}_i & \text{if } f(\mathbf{u}_i) \le f(\mathbf{x}_i^G) \\ \mathbf{x}_i^G & \text{otherwise} \end{cases} \tag{6}$$

where $\mathbf{u}_i$ is the trial vector generated by the crossover operator, and $\mathbf{x}_i$ is the current individual.

3.3.4 Parameter Adaptation. Each individual has its own scaling factor F_i and crossover probability C_i. These parameters are adapted during the optimization process based on the success of the individual. The scaling factor and crossover probability are updated as follows:

$$\begin{aligned} C_i &\sim \mathcal{N}(\mu_C, 0.1) \\ F_i &\sim \text{Cauchy}(\mu_F, 0.1) \end{aligned} \tag{7}$$

where μ_C and μ_F are the mean values of the crossover probability and scaling factor, respectively, $\mathcal{N}$ denotes the normal distribution, and Cauchy denotes the Cauchy distribution. They are truncated to the interval $[0, 1]$ to ensure that the parameters are within the valid range and 0.1 is the standard deviation of theses distributions. Our algorithm used the updating mechanism based on Success-History Based Parameter Adaptation (SHADE) algorithm [5].

3.4 Local Search

Algorithm 3: localSearch

```
1  Input:
2  - Best individual from a population: $x_{best}$
3  Output:
4  - Individual after applying local search: $x^*$
5  if $Gen == 1$ then
6      MTSLS1.effective ← ∞
7      LBFGSB.effective ← ∞
8  Choose $LS_G$ as the local search with the highest
     effectiveness
9  $x_{LS} \leftarrow LS_G(x_{best}, LS\_evals)$
10 $LS_G.effective \leftarrow \frac{f(x_{best}) - f(x_{LS})}{|f(x_{best})|}$
11 $x_{best} \leftarrow x_{LS}$
12 return $x_{best}$
```

Inspired by [6], we use an iterative local search strategy for the best individual in the population. The local search phase is used to refine the solution vectors obtained from the mutation and crossover operators. We use the L-BFGS-B algorithm [7] and MTS-LS1 [8] to perform the local search. Similar to IMODE, in MSHO, local search is only applied to the best individual with probability P_{ls}. algorithm 1 illustrates the application of local search and the process of updating P_{ls}.

The L-BFGS-B algorithm is a quasi-Newton optimization algorithm that uses the limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) method to approximate the Hessian matrix. The L-BFGS-B algorithm is used to minimize the objective function subject to the box constraints.

While the MTS-LS1 algorithm is a robust local search technique designed to refine solutions in optimization problems. It operates by iteratively exploring the neighborhood of a given solution through controlled perturbations, aiming to find improved solutions by evaluating and selecting the best candidates from these neighborhoods. MTS-LS1 is particularly effective in fine-tuning solutions obtained from global search methods, making it an excellent complement to other optimization techniques. Its ability to dynamically adjust search parameters based on progress helps in avoiding local optima and ensuring thorough exploration of the solution space.

3.5 Standby Stage

We use algorithm SSA [9] with dynamic population size immediately after the previous search ends, hoping for one last chance to reach a good solution. The SSA algorithm is a nature-inspired optimization algorithm that simulates the social behavior of salps in the ocean. The SSA algorithm is used to explore the search space by iteratively updating the position of the salps.

The SSA is initialized with the solution vectors obtained from the local search phase. The algorithm iteratively updates the solution vectors by computing the fitness value and updating the position of the salps.

3.5.1 Swarn Initialization. We initialize the population of salps $\mathcal{S} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{|\mathcal{S}|}\}$ within the box constraints $[\mathbf{L}, \mathbf{U}]$ as follows:

$$\mathbf{x}_i = \mathbf{L} + (\mathbf{U} - \mathbf{L}) \odot \mathbf{r}_i \quad \forall i = 1, 2, \dots, |\mathcal{S}| \tag{8}$$

where $\mathbf{r}_i$ is a vector which each element is a random number in the interval $[0, 1]$ sampled from the uniform distribution on $[0, 1]$ and $\odot$ denotes the element-wise multiplication.

We sort the salps based on their fitness value, the leader salps are $\left\lfloor \frac{|\mathcal{S}|}{2} \right\rfloor$ best individuals in the population. The follower salps are the remaining individuals in the population.

3.5.2 Updating Position. We iteratively update the position of the salps by computing the fitness value and updating the position of the salps. The position of the i-th leader salp where $i = 1, 2, \dots, \left\lfloor \frac{|\mathcal{S}|}{2} \right\rfloor$ is updated as follows:

$$\mathbf{x}_i = \mathbf{x}^{\text{best}} + c_1 \cdot \text{sign}(c_3 - 0.5) \odot [c_2 \odot (\mathbf{U} - \mathbf{L}) + \mathbf{L}] \tag{9}$$

where $\mathbf{x}^{\text{best}}$ is the current best individual in the population, c_2, c_3 are the vectors of random numbers in the interval $[0, 1]$, $\text{sign}(\cdot)$ is the sign function.

The parameter c_1 is the most important parameter in the salp swarm algorithm. It is used to control the exploration and exploitation of the algorithm and is updated using the following formula:

$$c_1 = 2 \cdot \exp\left(-\left(\frac{4E^G}{E_{\max}}\right)^2\right) \tag{10}$$

where E^G is the number of function evaluations used in the previous G generations, and $E_{\max}$ is the maximum number of function evaluations.

To udpate the position of the follower salp, we use the following formula based on the Newtonian mechanics:

$$\mathbf{x}_i^{G+1} = \frac{1}{2}\left(\mathbf{x}_i^G + \mathbf{x}_{i-1}^G\right) \tag{11}$$

where i is the index of the follower salp, $i \geq \left\lfloor \frac{|\mathcal{S}|}{2} \right\rfloor + 1$.

3.5.3 Dynamic Swarm Size. Also, the number of salps is updated based on the number of function evaluations. The number of salps is updated as follows:

$$|\mathcal{S}^G| = |\mathcal{S}^0| - \left\lfloor \frac{E^G}{E_{\max}} \cdot (|\mathcal{S}^0| - |\mathcal{S}^{\text{final}}|) \right\rfloor \tag{12}$$

where $|\mathcal{S}^0|$ is the initial number of salps, E^G is the number of function evaluations used in the previous G generations, and $E_{\max}$ is the maximum number of function evaluations. The number of leader salps and follower salps is updated based on this number of salps.

4 SETUP AND RESULT

4.1 Setup

This section provides the detail parameter setup of the proposed MSHO.

Parameter	Value
IMODE initial F	0.5
IMODE initial CR	0.5
IMODE p-best	0.1
IMODE archive rate	2.6
IMODE memory size	6
MTS-LS1 initial SR	-60
SSA initial population size	50
SSA minimum population size	20
MSHO initial population size	100
MSHO minimum population size	20
Function evaluations for IMODE per generation	1000
Function evaluations for local search per generation	1000
Gradient threshold	30
Fitness ratio	5
Mutation parameter	5

Table 1: Parameter values for algorithm

4.2 Result

REFERENCES

[1] Rainer Storn and Kenneth Price. "Differential Evolution - A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces". In: *Journal of Global Optimization* 11 (Jan. 1997), pp. 341–359. DOI: 10.1023/A:1008202821328.

[2] Karam M. Sallam et al. "Improved Multi-operator Differential Evolution Algorithm for Solving Unconstrained Problems". In: *2020 IEEE Congress on Evolutionary Computation (CEC)*. 2020, pp. 1–8. DOI: 10.1109/CEC48606.2020.9185577.

[3] Amir H. Gandomi et al. *GNBG-Generated Test Suite for Box-Constrained Numerical Global Optimization*. 2023. arXiv: 2312.07034 [math.OC]. URL: https://arxiv.org/abs/2312.07034.

[4] Kalyanmoy Deb and Ram Bhushan Agrawal. "Simulated Binary Crossover for Continuous Search Space". In: *Complex Syst.* 9 (1995). URL: https://api.semanticscholar.org/CorpusID:18860538.

[5] Ryoji Tanabe and Alex Fukunaga. "Success-history based parameter adaptation for Differential Evolution". In: *2013 IEEE Congress on Evolutionary Computation*. 2013, pp. 71–78. DOI: 10.1109/CEC.2013.6557555.

[6] Daniel Molina, Antonio LaTorre, and Francisco Herrera. "SHADE with Iterative Local Search for Large-Scale Global Optimization". In: *2018 IEEE Congress on Evolutionary Computation (CEC)*. 2018, pp. 1–8. DOI: 10.1109/CEC.2018.8477755.

[7] Ciyou Zhu et al. "Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization". In: *ACM Trans. Math. Softw.* 23.4 (Dec. 1997), pp. 550–560. ISSN: 0098-3500. DOI: 10.1145/279232.279236. URL: https://doi.org/10.1145/279232.279236.

[8] Lin-Yu Tseng and Chun Chen. "Multiple Trajectory Search for Large Scale Global Optimization". In: July 2008, pp. 3052–3059. DOI: 10.1109/CEC.2008.4631210.

[9] Seyedali Mirjalili et al. "Salp Swarm Algorithm: A bio-inspired optimizer for engineering design problems". In: *Advances in Engineering Software* 114 (2017), pp. 163–191. ISSN: 0965-9978. DOI: https://doi.org/10.1016/j.advengsoft.2017.07.002. URL: https://www.sciencedirect.com/science/article/pii/S0965997816307736.

Functions	Error			Acceptance			Success Rate
	Min	Mean	Std	Min	Mean	Std	
1	**0.0**	**5.87e-14**	**9.95e-14**	**1352.0**	**1737.06**	**479.54**	**100.00**
2	0.0	8.24e-03	2.70e-02	88972.0	150165.23	135970.72	87.10
3	**0.0**	**3.12e-14**	**2.83e-14**	**30438.0**	**34239.90**	**2111.31**	**100.00**
4	**0.0**	**0.00e+00**	**0.00e+00**	**5500.0**	**82093.48**	**66344.92**	**100.00**
5	0.0	3.56e-02	1.09e-01	41678.0	236322.71	128937.51	90.32
6	**0.0**	**0.00e+00**	**0.00e+00**	**56586.0**	**177294.87**	**71508.93**	**100.00**
7	0.0	4.21e-05	1.44e-04	44147.0	136281.06	178212.07	80.65
8	**0.0**	**0.00e+00**	**0.00e+00**	**44396.0**	**49798.19**	**2430.88**	**100.00**
9	**1.14e-13**	**2.20e-13**	**2.79e-14**	**219973.0**	**228021.81**	**4898.06**	**100.00**
10	0.0	6.29e-08	2.37e-07	52819.0	113610.94	132683.49	90.32
11	**0.0**	**0.00e+00**	**0.00e+00**	**2743.0**	**92044.48**	**75496.23**	**100.00**
12	**0.0**	**0.00e+00**	**0.00e+00**	**2538.0**	**108067.29**	**88074.85**	**100.00**
13	**0.0**	**1.03e-13**	**1.83e-13**	**8152.0**	**93019.03**	**62747.00**	**100.00**
14	**0.0**	**0.00e+00**	**0.00e+00**	**59422.0**	**151520.00**	**88303.88**	**100.00**
15	0.0	4.12e-02	6.05e-02	63325.0	300412.55	160084.48	67.74
16	**0.0**	**2.93e-13**	**4.25e-13**	**1538.0**	**14186.71**	**15416.04**	**100.00**
17	**0.0**	**0.00e+00**	**0.00e+00**	**28309.00**	**109626.55**	**61532.53**	**100.00**
18	**0.0**	**0.00e+00**	**0.00e+00**	**8364.00**	**72606.90**	**47712.13**	**100.00**
19	**0.0**	**0.00e+00**	**0.00e+00**	**10680.00**	**64766.94**	**42257.60**	**100.00**
20	**0.0**	**0.00e+00**	**0.00e+00**	**48466.0**	**216381.29**	**262412.66**	**100.00**
21	**0.0**	**0.00e+00**	**0.00e+00**	**60701.0**	**160585.52**	**63904.91**	**100.00**
22	**0.0**	**1.10e-14**	**4.43e-14**	**28440.0**	**78989.26**	**51184.65**	**100.00**
23	**0.0**	**0.00e+00**	**0.00e+00**	**58333.0**	**190556.16**	**175488.05**	**100.00**
24	0.0	4.67e-01	5.95e-01	51281.0	720939.35	402872.25	54.84

Table 2: Table presenting the mean and standard deviation of absolute errors, required FEs to reach the acceptance threshold and the success rate, based on 31 runs for each problem instance.