Abstract

Based on the analysis of the core idea of seven selected swarm-based optimization algorithms (SA), we propose a Unified Framework $UNIOA$ to simpify SA understandable on the primitive math level, to prevent SA from meaningless repetitions and to help building a common environment for the SA field. The detailed process of constructing the $UNIOA$ is illustrated, including the Unified terminologies, the Unified procedure and the Unified framework $UNIOA$. Meanwhile, practical experiments are performed to verify the reliability of $UNIOA$. In addition, a practical auto-designer demo for SA is developed, which introduces a future application of $UNIOA$. Subsequently, we leave an open discussion on the possibility of extending $UNIOA$ into the whole of nature-inspired optimiztaion algorithms, including the SA and the evolutionary algorithms (EA).

Contents

Symbols

Table 1: Some symbols in this work.

Symbol	Meaning
$\mathcal{U}(a,b)$	A single random uniform distributed number in $[a,b)$
$rand$	A single random uniform distributed number in $[0,1)$. It is always randomly generated when utilized.
$Exp(a)$	A single random exponential distributed number with the scale $\lambda = a$
$\mathbf{Min}(\{a_i\}) \rightarrow a_i$	Minimization calculator that will generate one a_i that has the minimum fitness value amongst the set $\{a_i\}$.
$\mathbf{Sort}(\{a_i\}) \rightarrow \langle a_i \rangle$	Sort calculator that will generate an ordered sequence $\langle a_i \rangle$ of the ascending fitness values of a_i amongst the set $\{a_i\}$.
$\mathbf{Round}(a) \rightarrow a^{'}$	Rounding calculator that will round a value a to its nearest integer $a^{'}$.
$\mathbf{Dist}(a,b) \rightarrow D_{a,b}$	Distance calculator that will calculate the Euclidean distance by $D_{a,b} = \sqrt{(a-b)^2}$.
$\vert\ \vert$	Only denotes absolute value.[1]

[1] Some places use it as a calculation of distance that is defined as $\mathbf{Dist}(a,b) \rightarrow D_{a,b}$ in our work.

Chapter 1

Introduction

1.1 Motivation

The pace of designing optimization algorithms has never been stopped even since antiquity [1]. In the late stage of the 1900s, heuristic algorithms began to stand out with their impressive performance on more complicated modern optimization problems [25]. In these heuristic research fields, due to the great success of the simulated annealing optimization algorithm [25], the analogy connection between nature and optimization methods attracts increasing attention.

As the most successful analogy [25], the natural evolution process has inspired most metaphor-based optimization algorithms until now. From classical genetic algorithms, particle swarm optimization to modern 'animal'-inspired optimization algorithms, from pure algorithms to their various variants, from independent algorithms to combination algorithms, these algorithms that have come or are coming are welcome to solve problems and enrich the research community, however, one issue in this research field becomes increasingly obvious: **a great number of new nature-inspired optimization algorithms are just old heuristic algorithms in new clothes, which might already mislead the development of heuristic algorithms** [11, 12, 23, 25].

This problem frequently appears in population-based algorithms that are also well-known as swarm-based algorithms. One paper mentioned that these new modern nature-inspired algorithms are actually similar to swarm intelligence [11], in which they theoretically displayed the common components between the swarm intelligence and the nature-inspired algorithm. Some researchers also mentioned that several core components comprised most meta-heuristic algorithms [12], in which they provided a generalized flow chart to display these common components. Moreover, some papers pointed out that a unified representation for organizing these algorithms together, especially a mathematical representation, will help prevent a worse development of heuristic algorithms [26, 32].

Therefore, in this work, we aim to propose a unified framework that can combine swarm-based optimization algorithms together. This work will start with several selected swarm-based optimization algorithms. Before starting this work, we want to clarify:

- *Variants of algorithms are not considered in this work.*
  Variants are important improvements to the algorithms, however, these variants will be ignored when building up the unified framework in this work. We will only focus on the base algorithms.

- *Hybrid algorithms are not considered in this work.*
  We must accept that it is very challenging to discuss independent and hybrid algorithms together. Therefore, we will start with independent algorithms and ignore hybrid algorithms here.

To build up a unified framework, we get inspired by gene terminologies and operations in the Evolutionary algorithm group. Therefore, in this work, we firstly study the possibility of unifying different terminologies in selected algorithms (see Chapter 2), which is the same as the gene terminology that is the only way to express evolutionary algorithms. Next, we study the possibility of unifying different procedures in these selected algorithms (see Chapter 3), in which we discuss the possibility that these algorithms have the same model structure. Finally, we give our unified framework for several selected algorithms (see Chapter 4), in which selected algorithms can be re-written in this unified framework without losing their quality of optimization.

Furthermore, we will also prove the feasibility of this unified framework by benchmark experiments, and meanwhile compare the performance of these selected algorithms in our unified framework (see Chapter 5, Chapter 6). Lastly, we conclude our observations, point out our limitations in this work, and give future research suggestions (see Chapter 7).

1.2 Overview

As Figure 1.1 displayed, since unifying the entire swarm-based algorithms is impossible in one work, we will start with working on seven examples, including new modern and old classical examples, after proving the feasibility of the unified framework we will build, we will discuss the possibility that this framework is also able to be used in other swarm-based algorithms.

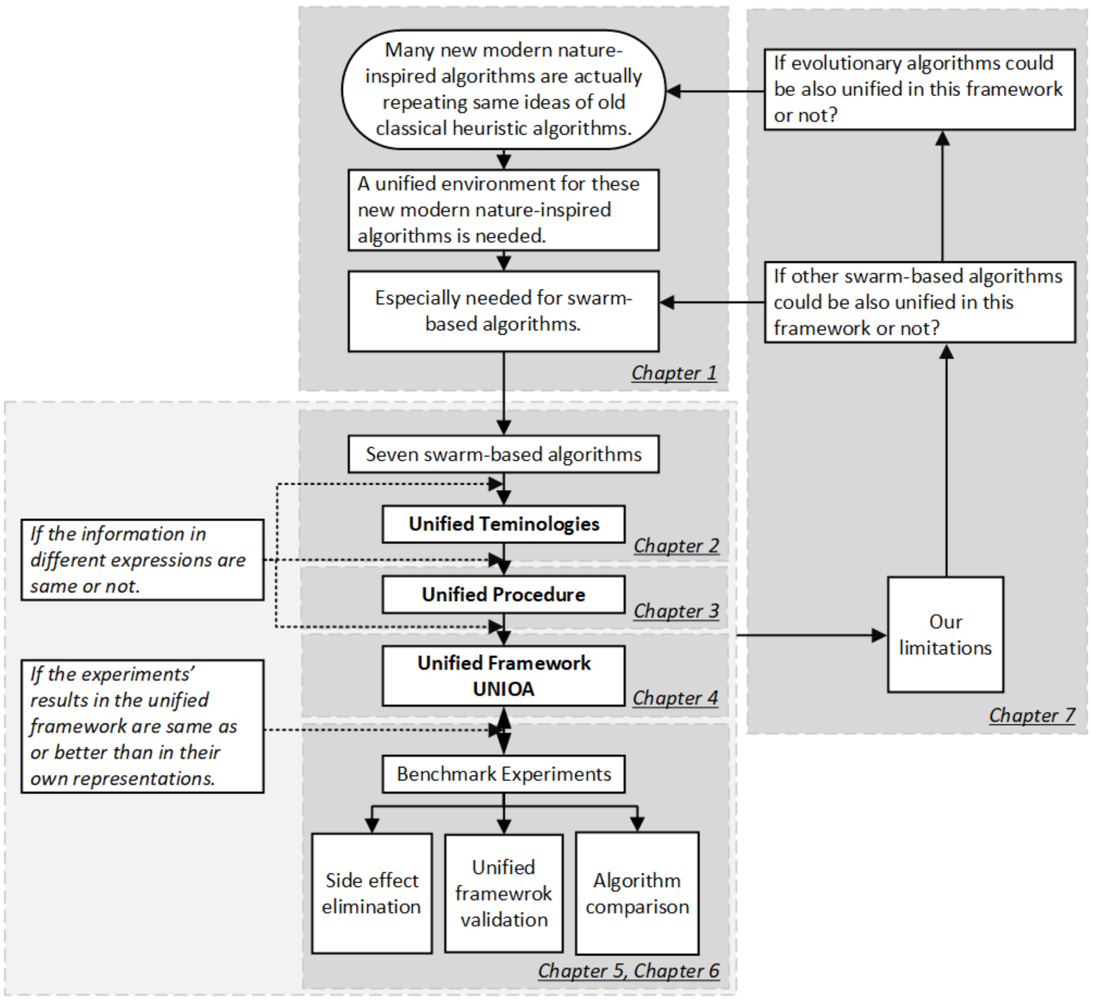
Many new modern nature-inspired algorithms are actually repeating same ideas of old classical heuristic algorithms.
A unified environment for these new modern nature-inspired algorithms is needed.
Especially needed for swarm-based algorithms.
Chapter 1
If evolutionary algorithms could be also unified in this framework or not?
If other swarm-based algorithms could be also unified in this framework or not?
Seven swarm-based algorithms
Unified Teminologies
Chapter 2
If the information in different expressions are same or not.
Unified Procedure
Chapter 3
Unified Framework UNIOA
Chapter 4
If the experiments' results in the unified framework are same as or better than in their own representations.
Our limitations
Chapter 7
Benchmark Experiments
Side effect elimination
Unified framewrok validation
Algorithm comparison
Chapter 5, Chapter 6

Figure 1.1: Overview.

These seven examples will be selected from the Bestiary of evolutionary computation[1]. They are :

- New modern swarm-based algorithms:
  Bat-inspired algorithm [31], Grasshopper optimization algorithm [24], Crow search

[1] https://github.com/fcampelo/EC-Bestiary

algorithm [2], Moth-flame optimization algorithm, Monarch butterfly optimization algorithm [28], Butterfly optimization algorithm [3].

- Old classical swarm-based algorithms:
  Particle swarm optimization [16]

We hope our unified framework is unified enough to cover both traditional and modern nature-inspired algorithms.

Chapter 2

Unified Terminologies

2.1 Motivation

This chapter discusses the commonalities among seven selected swarm-based algorithms on the level of their components. Specifically, in these algorithms, we find that various terminologies of components with their meanings are able to be categorized into same terminologies with same meanings.

Firstly, we want to introduce several common symbols in swarm-based algorithms. In the swarm-based algorithm research field[1], the given objective optimization problem is formulated as a math formula $f(\vec{x})$ in which all decision variables are represented as a vector $\vec{x}$. Moreover, the number of decision variables[2] is usually defined as the dimension of search space to the objective optimization problem. When seeking the optimal solution to the optimization problem, the swarm-based algorithm will first initialize an arbitrary set of $\vec{x}$ and then optimize this set by using iterative strategies. Same as in most studies, we also denote $\vec{x}$ as $\mathbf{x}$ and call it the possible objective solution. Inspired by this representation, we define that any term $\mathbf{a}$ in the bold style denotes that this item is also a vector with the same dimension as the objective solution $\mathbf{x}$. Furthermore, we want to clarify that if the bold style item $\mathbf{a}$ has a subscript i, it means this item has a one-to-one correspondence with each $\mathbf{x}_i$. However, one item without this subscript will act on the whole set of $\mathbf{x}_i$, whether this item is in bold style or not in bold style.

[1]In our work, we only consider single-objective continuous problem optimization.

[2]The number of decision variables is also the number of independent elements in $\vec{x}$.

Moreover, usually, the set of $\mathbf{x}_i$ is called a population, and the number of $\mathbf{x}_i$ in this set is called the size of this population. Each $\mathbf{x}_i$ is a possible optimal solution to the target optimization problem $f()$, therefore the value of $f(\mathbf{x}_i)$ is called the fitness value of $\mathbf{x}_i$.

In the following sections, Section 2.2 introduces these seven algorithms in detail but doesn't include how natures are simulated to create mathematical models. Readers can find such information in their published papers. In this work, the description of each algorithm is straightforward, only including the final model and its default hyper-parameter settings. Meanwhile, we also summarize their special points. Section 2.3 will give and discuss our unified terminologies based on these seven selected algorithms. A summary about this chapter is displayed in Section 2.4.

2.2 Specific Swarm-based Algorithms

2.2.1 Bat-inspired Algorithm (BA)

Description When simulating the echolocation behavior of micro-bats, for simplicity, the Bat-inspired algorithm only considered the velocity $\mathbf{v}_i$ of each bat $\mathbf{x}_i$, the echo frequency $freq$ emitted by each bat $\mathbf{x}_i$, the rate r of echo pulse, and the loudness A of echo [31]. The algorithm described here follows the published implementation code[3], and explanations with math formulas are from their published paper [31].

This algorithm starts from is (1) initializing a bat $\mathbf{x}_i$ population with size $n = 20$ in the d dimension environment and the initialization method is Eq.2.1 in which Lb/Ub is the lower/upper boundary of each element in $\mathbf{x}_i$. Meanwhile, (2) velocity $\mathbf{v}_i$ of each bat $\mathbf{x}_i$ is initialized as Eq.2.2.

$$\mathbf{x}_i(t=0) = \boldsymbol{\mathcal{U}}(Lb, Ub), i = 1, 2, \cdots, n \tag{2.1}$$

$$\mathbf{v}_i(t=0) = \boldsymbol{\mathcal{U}}(0, 0), i = 1, 2, \cdots, n \tag{2.2}$$

After (3) evaluating fitness of each $\mathbf{x}_i$, it (4) goes to find the best individual $\mathbf{x}_*$ that is the best $\mathbf{x}_i$ the whole population has found so far. Next is (5) the iterative optimization process in which the maximum number of iterations is $t\_max = 1000$. Under the iterative process, it firstly (6) updates A as Eq.2.3 in which $A^0 = 1$ is its initial value, and $\alpha = 0.97$ is a decreasing hyper-parameter. Next, (7) r is updated as Eq.2.4 in which $r_0 = 1$ is the initial value of pulse rate and $\gamma = 0.1$ is a decreasing hyper-parameter. In this algorithm,

[3] https://uk.mathworks.com/matlabcentral/fileexchange/74768-the-standard-bat-algorithm-ba?s_tid=prof_contriblnk

t is the iteration counter.

$$\begin{cases} A(t) = \alpha \times A^0 & , \quad t = 0 \\ A(t+1) = \alpha \times A(t) & , \quad \text{o.w} \end{cases} \tag{2.3}$$

$$r(t) = r_0 \times (1 - e^{-\gamma \times t}) \tag{2.4}$$

Next is (8) updating each $\mathbf{x}_i$ as Eq.2.5 in which $freq\_min = 0/freq\_max = 2$ is the lower/upper boundary of $freq_i$.

$$\begin{aligned} freq_i &= \mathcal{U}(freq\_min, freq\_max) \\ \mathbf{v}_i(t+1) &= \mathbf{v}_i(t) + (\mathbf{x}_i(t) - \mathbf{x}_*) \times freq_i \\ \mathbf{x}_i(t+1) &= \mathbf{x}_i(t) + \mathbf{v}_i(t+1) \end{aligned} \tag{2.5}$$

If (9) $rand < r(t)$, $\mathbf{x}_i$ will be updated differently as Eq.2.6 in which $\epsilon = 0.1$.

$$\mathbf{x}_i(t+1) = \mathbf{x}_*(t) + \epsilon \times \mathbf{rand} \times A(t+1) \tag{2.6}$$

After (10) fixing outliers in $\mathbf{x}_i(t+1)$ by replacing them with the boundary value, the BA model evaluates fitness of each $\mathbf{x}_i(t+1)$ again. Then (11) if the fitness of $\mathbf{x}_i(t+1)$ is better than $\mathbf{x}_i(t)$ or the situation is $rand > A$, $\mathbf{x}_i(t+1)$ will be accepted. Lastly (12) the $\mathbf{x}_*$ is updated.

Until now, one optimization round (from step (6) to step(12)) is finished, and it will iteratively execute until the stop condition meets.

Summary When modeling a Bat algorithm, there are two main processes: Initialization and Optimization. In the initialization process, besides initializing an initial population and all default parameters (including hyper-parameters and initial values), an initial velocity population is also initialized. Meanwhile, the initialization process also calculates the initial best bat. In the optimization process, two dynamically changing parameters are updated before updating the population. It is worth mentioning that the whole process of updating the population is asynchronous, including asynchronously evaluating the fitness of each bat, asynchronously judging whether the updated bat is improved, and asynchronously updating the current best bat.[4]

Moreover, we want to clarify that in their implemented codes, it is unreasonable to compare the previous best bat with a current unaccepted bat when updating the current

[4]The asynchronism here means the next individual will update only after the previous individual finishes updating, evaluating and all other optimization steps.

best bat. The definition of the current best bat shall be finding the best bat in the current population. The unaccepted bat is not in the current population anymore. Therefore, it is more reasonable to calculate the current best bat by comparing the previous best bat with the current bat who is at this same position in the current population.

Furthermore, we want to point out that in the Initialization process, the evaluation of the population happens immediately after initializing the initial population. However, in the Optimization process, the evaluation happens immediately after generating a temporary population. In other words, it happens in the middle of obtaining the temporary population and judging if accepting the generated individuals as the updated individuals, which means the evaluation does not happen on the final new updated population, but on a temporary updated population.

2.2.2 Grasshopper Optimization Algorithm (GOA)

Description The Grasshopper Optimization Algorithm was proposed in 2017 [24]. They designed this algorithm by studying grasshoppers' main swarming behaviors in larval and adulthood phases, such as slow movement and small steps in the larval phase, abrupt movement and long-range steps in the adulthood phase, and food source seeking in both phases. Their final model described here is from their published code [5], and explanations with math formulas are from their published paper [24].

This model starts at (1) initializing grasshoppers $\mathbf{x}_i$ with size $N = 100$ in the d dimension environment. The initialization method in their published code can be simply re-framed as Eq.2.7 in which $up/down$ is the upper/down boundary of $\mathbf{x}_i$ [6].

$$\mathbf{x}_i = \boldsymbol{\mathcal{U}}(down, up), i = 1, 2, \cdots, N \tag{2.7}$$

Next is (2) calculating the fitness of each individual $\mathbf{x}_i$ and (3) finding the best individual $\mathbf{T}$ that the whole population has found so far. The followings are iterative optimizing process in which the stop condition is current number of iteration l is smaller than the maximum number of iterations $max\_iteration = 100$ [7]. Under the iterative process, (4) the hyper-parameter c is firstly updated as Eq.2.8 in which $cMin = 0.00004/cMax = 1$

[5]https://seyedalimirjalili.com/goa

[6]Their published implementation code considered two boundary cases: if elements in $\mathbf{x}_i$ share the same boundary or if elements in $\mathbf{x}_i$ have different boundaries. According to the further experiments in the Chapter 5, we only consider the first case.

[7]Their published code thought the current iteration $l = 2$ when starting optimizing, however, we define the current iteration $l = 1$ when starting optimization in order to be same as other algorithms.

is the lower/upper boundary of c.

$$c = cMax - l \times \frac{cMax - cMin}{max\_iteration} \tag{2.8}$$

Then, the GOA model (5) normalizes the distances $\widetilde{d}$ between individuals into $[1, 4]$. The normalization method in their published code is Eq.2.9 in which d is the Euclidean distance [8] between individual pairs.

$$\widetilde{d} = 2 + d \text{ mod } 2 \tag{2.9}$$

Next is (6) updating $\mathbf{x}_i$ as Eq.2.10 that is a simplified version of their published original formula. Because we only consider one case that elements in $\mathbf{x}_i$ share the same boundary, we replace ub_d/lb_d by ub/lb. We also replace x_i^d by $\mathbf{x}_i$ and $\widehat{T_d}$ by $\mathbf{T}$ in the view of vector [9].

$$\mathbf{x}_i = c \times \left(\sum_{j=1, j \neq i}^{N} c \times \frac{ub - lb}{2} \times S\left(\widetilde{d}_{i,j}\right) \times \frac{\mathbf{x}_j - \mathbf{x}_i}{d_{ij}} \right) + \mathbf{T} \tag{2.10}$$

The method of updating Eq.2.10 has a function named S in which $\widetilde{d}_{i,j}$ is the input. The S function is formulated as Eq.2.11 in which $f = 0.5$ and $l = 1.5$ are two hyper-parameters [10].

$$S(\widetilde{d}_{i,j}) = f e^{\frac{-\widetilde{d}_{i,j}}{l}} - e^{-\widetilde{d}_{i,j}} \tag{2.11}$$

After updating each $\mathbf{x}_i$, (7) outliers in $\mathbf{x}_i$ are replaced by the boundary values before (8) calculating its fitness. The last step is (9) updating the best individual $\mathbf{T}$.

Until now, one optimization round (from step(4) to step(9)) is finished, and it will iteratively execute until the stop condition meets.

Summary When modeling a GOA algorithm, there are two main processes: Initialization process and Optimization process. In the Initialization process, the algorithm has only two main tasks: initializing the initial population and finding the global best individual in the initial population. In the Optimization process, the update strategy depends on neighbors' positions.

2.2.3 Crow Search Algorithm (CSA)

Description The crow search algorithm was designed [2] in 2016. The core metaphor is crows' behaviour of hiding excess food and retrieving this stored food when needed.

[8] It refers to their faster version code.

[9] Their published paper didn't point out whether $\widehat{\ }$ is a special operation or not.

[10] Please note, here the l is not the current iteration mentioned above.

The author believes that these behaviours, including stealing others' food by tailing after them and fighting with theft by moving to another place instead of their real hiding place, are similar to an optimization process [2]. Their final algorithm model described here is from their published implementation code [11], and explanations with math formulas are from their published paper [2].

The algorithm starts at (1) initializing the crow population of size $N = 20$ in the d dimension environment. The initialization method they used in their implementation code can be formulated as Eq.2.12 in which l/u is the lower/upper boundary.

$$\mathbf{x}_i = \boldsymbol{\mathcal{U}}(l, u), i = 1, 2, \cdots, N \tag{2.12}$$

The next is (2) calculating the fitness of each individual $\mathbf{x}_i$, and (3) initializing the memory $\mathbf{m}_i$ for each individual $\mathbf{x}_i$ as Eq.2.13.

$$\mathbf{m}_i = \mathbf{x}_i \tag{2.13}$$

The following steps are about the iterative optimization process in which the stop condition is the maximum number of iterations $tmax = 5000$. Under the iterative process, (4) each individual $\mathbf{x}_i$ is updated as Eq.2.14 in which $fl = 2$ and $AP = 0.1$ are two hyper-parameters.

$$\mathbf{x}_i(t+1) = \begin{cases} \mathbf{x}_i(t) + rand \times fl \times (\mathbf{m}_j(t) - \mathbf{x}_i(t)) & , \quad rand > AP \\ \boldsymbol{\mathcal{U}}(u, l) & , \quad \text{o.w} \end{cases} \tag{2.14}$$

After (5) calculating the fitness of the new individual $\mathbf{x}_i(t+1)$, if (6) the new individual $\mathbf{x}_i(t+1)$ is in the case of $l \leq \mathbf{x}_{i,d}(t+1) \leq u$, the new individual $\mathbf{x}_i(t+1)$ will be accepted, if not, the individual $\mathbf{x}_i(t)$ will keep unchanged and reject to update. Lastly, (7) the memory $\mathbf{m}_i(t+1)$ will be updated as Eq.2.15.

$$\mathbf{m}_i(t+1) = \begin{cases} \mathbf{x}_i(t+1) & , \quad f(\mathbf{x}_i(t+1)) < f(\mathbf{m}_i(t)) \\ \mathbf{m}_i(t) & , \quad \text{o.w} \end{cases} \tag{2.15}$$

Until now, one optimization round (step (4) to step (7)) is finished, and it will iteratively execute until the stop condition meets.

[11] https://nl.mathworks.com/matlabcentral/fileexchange/56127-crow-search-algorithm

Summary When modeling a CSA algorithm, there are two main processes: Initialization process and Optimization process. In the Initialization process, the algorithm has only two main tasks: initializing the initial population and the initial memory population that is the same as the initial population. In the Optimization process, the memory position of neighbors determines how far the individual moves.

When meeting outliers, the CSA algorithm chooses to give up the new position. In other words, as far as there is an outlier, the individual rejects to be updated.

2.2.4 Moth-flame Optimization Algorithm (MFO)

Description In 2015, the Moth-flame optimization algorithm simulated the process that moths will fly to the center of artificial light (that is also seen as flames) in the spiral path [21]. The core characteristic in MFO is that the author defined the positions of flames as the optimization orientation of moths, and each moth has its own dynamically changing flame. The final algorithm model described here is from their published implementation code [12], and explanations with math formulas are from their published paper [21].

This algorithm starts from (1) $N = 30$ initial moths population $Moth\_pos$ in a dim dimension environment. The initialization method in their published code can be formulated as Eq.2.16 in which $\mathbf{m}_i$ is each moth and ub/lb is the upper/lower boundary of elements in each moth [13].

$$\mathbf{m}_i = \boldsymbol{\mathcal{U}}(lb, ub), i = 1, 2, \cdots, N \tag{2.16}$$

Then the algorithm model starts iteratively optimizing each $\mathbf{m}_i$ in which the stop condition is there is a maximum number of iterations T. Under the iterative optimization process, first of all, (2) the number of flames is calculated as Eq.2.17 in which l is the current number of iteration. The $flame\_no$ will impact the updating positions of moths in the following steps.

$$flame\_no = \mathbf{Round}(N - l \times \frac{N-1}{T}) \tag{2.17}$$

After (3) replacing outliers in $\mathbf{x}_i$ by the boundary values, fitness of each moth is calculated. Next is (4) sorting the combination of the current population and the previous population [14] based on the ascending of their fitness, and the first $N = 30$ moths in the combination

[12]https://nl.mathworks.com/matlabcentral/fileexchange/52270-moth-flame-optimization-mfo-algorithm-toolbox?s_tid=srchtitle

[13]In this work, we only consider that the boundaries of all elements in each moth are same, although their published code also provided a method to deal with different boundaries.

[14]The previous population is ∅ (there is no previous population), when the current population is the initial population.

population are defined as $sorted\_population$. Then, a linearly decreasing parameter a is calculated by Eq.2.18 in which l is the current number of iteration.

$$a = -1 + l \times \frac{-1}{T} \tag{2.18}$$

Next, (5) each element j of each moth $\mathbf{m}_i$ will be updated as follows: if $i \leqslant flame\_no$, Eq.2.19; if not, Eq.2.20 in which $\mathbf{m}^{'}_{i,j}$ denotes each newly updated element j of each moth $\mathbf{m}_i$; $b = 1$ is a constant parameter used to define the shape of spiral [21]; $t = (a-1) \times rand + 1$ is a random number related to a.

$$\begin{aligned} &distance\_to\_flame = |sorted\_population(i,j) - \mathbf{m}_{i,j}| \\ &\mathbf{m}^{'}_{i,j} = distance\_to\_flame \times e^{b \times t} \times \cos(t \times 2 \times \pi) + sorted\_population(i,j) \end{aligned} \tag{2.19}$$

$$\begin{aligned} &distance\_to\_flame = |sorted\_population(i,j) - \mathbf{m}_{i,j}| \\ &\mathbf{m}^{'}_{i,j} = distance\_to\_flame \times e^{b \times t} \times \cos(t \times 2 \times \pi) + sorted\_population(flame\_no,j) \end{aligned} \tag{2.20}$$

Until now, one optimization round (from step(2) to step(5)) is finished, and it will iteratively execute until the stop condition meets.

Summary When modeling a Moth-flame algorithm, there are two processes: Initialization process and Optimization process. In the initialization process, this algorithm only initializes an initial population with all default parameters. In the optimization process, the main convergency mechanism is to keep moving to better solutions by narrowing better choices down. Here, the better choices are selected from the top several best solutions, and the method of narrowing down is to narrow the number of best solutions. The main convergency mechanism happens at Eq.2.20.

Compared with other algorithms, this algorithm has another two different places. The first one is that there is no evaluation operator in the initialization process. The second one is that fixing outliers and evaluating are happening at the beginning of the optimization process.

2.2.5 Monarch Butterfly Optimization (MBO)

Description The Monarch Butterfly Optimization simulated the migration behavior of monarch butterflies in 2015 [28]. The core metaphor is that monarch butterflies in two different habitats evolve differently. These two habitats are mimicked by dividing the whole population into two subpopulations. The author defined that the whole population will firstly be sorted based on their fitness before dividing them into two with a ratio [28]. Then, each habitat experiences its particular evolution approach (migration operator or adjusting operator [28]). We describe their final algorithm model implemented in their published code [15], and explanations with math expressions from their published paper [28].

This algorithm starts from (1) an initial population $Population$ with size $M = 50$, and the initialization method [16] in their published code can be formulated as Eq.2.21 in which $MinParValue/MaxParValue$ is the lower/upper boundary of elements in $\mathbf{x}_i$.

$$\mathbf{x}_i = \boldsymbol{\mathcal{U}}(MinParValue, MaxParValue), i = 1, 2, \cdots, M \tag{2.21}$$

After (2) evaluating each $\mathbf{x}_i$, the entire population is sorted from the best to the worst. Then (3) it starts iteratively optimizing each $\mathbf{x}_i$ in which the stop condition is the maximum number of iterations $Maxgen = 50$. Under the iterative process, (4) $Keep = 2$ best $\mathbf{x}_i$ are first kept in $chromKeep$ [17]. Next is (5) dividing the entire $Population$ into two sub-populations with a ratio $partition = \frac{5}{12}$. The two sub-populations ($Population1$, $Population2$) will be updated by two operators ('Migration operator', 'Adjusting operator') respectively. (6) In the $Population1$, if $rand \times period \leqslant partition$ in which $period = 1.2$, $\mathbf{x}_{i,k}^{t+1} = \mathbf{x}_{population1,k}^{t}$; else $\mathbf{x}_{i,k}^{t+1} = \mathbf{x}_{population2,k}^{t}$, in which k is the $k-$th element of $\mathbf{x}_i$, and $population1/population2$ is one arbitrary moth from previous $Population1/Population2$. (7) Next is fixing outliers in this newly generated $Population1$ by replacing them with the boundary value, before evaluating this newly generated $Population1$.

(8) In $Population2$, if $rand \leqslant partition$, $\mathbf{x}_{i,k}^{t+1} = \mathbf{x}_{best,k}^{t}$ in which k is the $k-$th element and $best$ is the best moth the entire population has found so far; else $\mathbf{x}_{i,k}^{t+1} = \mathbf{x}_{population2,k}^{t}$, in which k is the $k-$th element, and $population2$ is one arbitrary moth from previous $Population2$, furthermore, under this situation, if $rand > BAR$ in which $BAR = \frac{5}{12}$, $\mathbf{x}_{i,k}^{t+1}$ will be further updated as Eq.2.22 in which α is a weighting factor as Eq.2.23, and $\mathrm{d}\mathbf{x}$ is a walk step as Eq.2.24. (9) Then, outliers in the newly generated $Population2$ are fixed by replacing them with the boundary value, and next is evaluating this newly generated $Population2$.

$$\mathbf{x}_{i,k}^{t+1} = \mathbf{x}_{i,k}^{t+1} + \alpha \times (\mathrm{d}\mathbf{x}_k - 0.5) \tag{2.22}$$

[15] https://nl.mathworks.com/matlabcentral/fileexchange/101400-monarch-butterfly-optimization-mbo?s_tid=srchtitle

[16] The original paper considered the situation if boundaries of elements are same or not, but in this work, we only consider the situation that boundaries of elements are same

[17] This elitism strategy stated in their published code wasn't stated in their publish paper.

$$\alpha = \frac{S_{max}}{t^2} \text{ where } S_{max} = 1 \text{ and } t \text{ is the current iteration.} \tag{2.23}$$

$$\mathrm{d}\mathbf{x} = \mathbf{Levy}(\mathbf{x}_j^t) \tag{2.24}$$

where $\mathbf{Levy}$ function in their published code can be formulated as Eq.2.25.

$$\begin{aligned} &Stepsize \sim Exp(2 \times Maxgen) \\ &\mathrm{d}\mathbf{x}_k = \sum \underbrace{(\tan(\pi \times rand), \cdots, \tan(\pi \times rand))}_{Stepsize} \end{aligned} \tag{2.25}$$

Then (10) these two newly generated sub-populations are combined into one population. Next, (11) after the last $Keep = 2$ worst $\mathbf{x}_i$ will be replaced by $chromKeep$ that is mentioned in step (4), (12) the final new population will be sorted again.

Until now, one optimization round (from step (4) to step(12)) is finished, and it will iteratively execute until the stop condition meets.

Summary When modeling an MBO algorithm, there are two main processes: Initialization and Optimization. In the initialization process, besides initializing an initial population and all default parameters, the initial population is also sorted from best to worst. In the optimization process, the core idea is that the whole population is divided into two sub-populations, and they are updated differently. One sub-population is better than another one because of the previous sort operator. The elitism strategy is also a highlight of this model. It happens at the beginning and the end of the optimization process.

Furthermore, although in the initialization process, the evaluation of the population only happens once immediately after initializing the initial population, there are two evaluations in the optimization process. There are two evaluations because the whole population is updated in two methods for two sub-populations.

2.2.6 Butterfly Optimization Algorithm (BOA)

Description The Butterfly Optimization Algorithm was modeled in 2018 [3]. Butterflies' behaviors of searching for food and mates are mimicked to solve optimization problems. The algorithm model described here is from their published code [18], and explanations with math formulas are from their published paper [3].

[18]https://www.mathworks.com/matlabcentral/mlc-downloads/downloads/b4a529ac-c709-4752-8ae1-1d172b8968fc/67a434dc-8224-4f4e-a835-bc92c4630a73/previews/BOA.m/index.html

In this algorithm, they define a dim dimensional environment in which each individual is $\mathbf{x}_i$ and the fitness of $\mathbf{x}_i$ is $f(\mathbf{x}_i)$. (1) The algorithm first initializes a population $n = 50$ of $\mathbf{x}_i$ by Eq.2.26 in which Lb/Ub is lower/upper bound [19].

$$\mathbf{x}_i = \boldsymbol{\mathcal{U}}(Lb, Ub), i = 1, 2, \cdots, n \tag{2.26}$$

It also (2) setups three parameters: probability switch $p = 0.8$, $power\_exponent = 0.1$ and $sensory\_modality = 0.01$. After evaluating each $\mathbf{x}_i$, it (3) calculates the best individual g^* that the whole population had found so far.

Next steps are the iterative optimization process in which the stop condition is the current iteration t is smaller than the maximum number of iterations N_iter. Under the iterative process, for each $\mathbf{x}_i$, (4) its own fragrance FP_i is first calculated as Eq.2.27 in which $f(\mathbf{x}_i)$ is the fitness value of $\mathbf{x}_i$ [20].

$$FP_i(t) = sensory\_modality \times f(\mathbf{x}_i(t))^{power\_exponent} \tag{2.27}$$

Then (5) if $rand > p$, the $\mathbf{x}_i$ will be updated as Eq.2.28, (6) if not, the $\mathbf{x}_i$ will be updated as Eq.2.29 in which $\mathbf{x}_j$ and $\mathbf{x}_k$ are two arbitrary neighbors around $\mathbf{x}_i$.

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + FP_i(t) \times (rand^2 \times g^*(t) - \mathbf{x}_i(t)) \tag{2.28}$$

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + FP_i(t) \times (rand^2 \times \mathbf{x}_j(t) - \mathbf{x}_k(t)) \tag{2.29}$$

Then after (7) replacing the outliers by the boundary values, (8) only if the updated $\mathbf{x}_i(t+1)$ is better than the previous $\mathbf{x}_i(t)$, the update $\mathbf{x}_i(t+1)$ will be deliveried into the next optimization round. Next steps are (9) updating g^* and (10) updating $sensory\_modality$ using Eq.2.30.

$$sensory\_modality(t+1) = sensory\_modality(t) + \frac{0.025}{sensory\_modality(t) \times N\_iter} \tag{2.30}$$

Until now, one optimization round (step(4) to step(10)) is finished, it will iteratively execute until the stop condition meets.

[19] The initialization method isn't shown in their published paper or code, but according to their published code that only said there was an 'initialization' function receiving n, dim, Ub and Lb as inputs. We can reasonably assume their initialization method is Eq.2.26.

[20] Their published code calculated the fitness twice here. However, the published paper did not mention whether this is a necessary operator. Therefore, we ignore this duplicated evaluation when studying this algorithm.

Summary When modeling a BOA algorithm, there are two processes: Initialization process and Optimization process. In the Initialization process, there are four tasks: initialize the initial population and evaluate them, find the global best one individual, and set up several hyper-parameters. In the optimization process, the most interesting part is the fitness joins in optimizing the population. Moreover, the square value of the random number is also a special place.

Meanwhile, the original implemented code also has the same issue as the BA algorithm (see Subsection 2.2.1) when updating the global best one individual. We think it is unreasonable to compare the previous best individual with an unaccepted one when updating the current global best individual. The definition of the current best global individual shall be the best one that the current population has found so far. Here, the current population shall be the population that has been accepted and will be delivered into the next optimization round. Therefore, it is more reasonable to calculate the current global best one by comparing the previous best one with the current accepted best one at this same position in the current population.

2.2.7 Particle Swarm Optimization (PSO)

Description Particle Swarm Optimization was firstly introduced in 1995 [16]. The main idea comes from the movements of the animal swarm. By simulating their behaviors that the swarm dynamically moves to a 'roost' [16], the extremely simple algorithm, even with a small swarm size (15 to 30 agents), boasted impressive performance on solving continuous optimization functions. In the very first paper, the position of each agent in its swarm was controlled by two velocities $\mathbf{X}$ and $\mathbf{Y}$. Moreover, each agent can remember its best position and know the global best position in its swarm. Although the very first paper also provided the algorithm model with formulas, we preferred to reference clearer published pseudocode whose algorithm model is as same as the very first paper possible. The final PSO model described in this work is from a tutorial of this method [20], and the hyper-parameter settings are from an application of this method [5].

This algorithm starts (1) from an initial swarm $\mathbf{x}_i$ with the size $N = 25$. The initialization method is formulated as Eq.2.31 in which min/max is the lower/upper boundary of $\mathbf{x}_i$.

$$\mathbf{x}_i(t=0) = \boldsymbol{\mathcal{U}}(min, max), i = 1, 2, \cdots, N \tag{2.31}$$

Then (2) each velocity $\mathbf{v}_i$ is initialized as Eq.2.32 in which $lb_{\mathbf{v}}/ub_{\mathbf{v}}$ is the boundary of velocity. Next (3) each $pbest_i$ is calculated as Eq.2.33.

$$\mathbf{v}_i = \boldsymbol{\mathcal{U}}(lb_{\mathbf{v}}, ub_{\mathbf{v}}), i = 1, 2, \cdots, N \tag{2.32}$$

$$pbest_i = \mathbf{x}_i, i = 1, 2, \cdots, N \tag{2.33}$$

After (4) calculating the fitness of each agent $\mathbf{x}_i$, the algorithm obtains (5) the best agent $gbest$ that is the whole swarm has found so far in the initial swarm. The following steps are about (6) iterative optimization process with a stopping condition. Under the iterative process, the published pseudo-code first updates each velocity $\mathbf{v}_i$ by Eq.2.34 in which $w = 0.73$ is an adjusting parameter, $c_1 = 1.49$ and $c_2 = 1.49$ is respectively the cognitive and social coefficient [20].

$$\mathbf{v}_i(t+1) = w \times \mathbf{v}_i(t) + c_1 \times rand \times (pbest_i - \mathbf{x}_i) + c_2 \times rand \times (gbest - \mathbf{x}_i) \tag{2.34}$$

After updating (7) each agent as Eq.2.35, all elements in each new agent are checked if they are feasible. The last step in one optimization round is (8) updating $pbest$ and (9) $gbest$.

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t+1) \tag{2.35}$$

Until now, one optimization round (from step(6) to step(9)) is finished, and it will iteratively execute until the stop condition meets.

As a side note, the very first published paper did not directly point out the notation for search space dimension, the stop condition, and if outliers should be dealt with or not. However, as discussed in the Section 2.1, the dimension depends on the number of elements in $\mathbf{x}_i$, therefore, it is reasonable to accept that the PSO model also needs a concept of dimension. Meanwhile, the stop condition and the method to deal with the outliers also must exist. In this work, the method of fixing outliers is the most common way that replaces outliers by boundary values of $\mathbf{x}_i$.

Summary When modeling a PSO algorithm, there are two processes: Initialization process and Optimization process. In the Initialization process, four components need to be initialized: the initial population, their velocity population, their personal best position group, the global best one position in this population. In the Optimization process, individuals are optimized one by one, which means that after updating one individual, its fitness, its $pbest$, and the current $gbest$ are then updated.

Moreover, in the Initialization process, the evaluation happens immediately after generating the initial population. In the Optimization process, the evaluation happens after generating a temporary population and before updating $pbest$ and $gbest$.

2.3 Unified Terminologies

2.3.1 Unified Terminologies Catalog

Although these selected algorithms use different expressions, their core meanings are the same. For example, the memory position in CSA is the personal best position in PSO if we only consider the information carried by the memory position and the personal best position. Therefore, **the information carried by these various expressions is the only principle to categorize these original terminologies**. As shown in Table 2.1, we find **20 unified components can cover the entire components in these seven selected algorithms**. Specifically, only 20 kinds of information are used when modeling these seven algorithms. At the same time, **these 20 unified components can be further divided into two groups: compulsory components and selective components**, as shown in Figure 2.1.

In Table 2.1, ***Information 1 to 5, 13, 15, 17-20* are compulsory components**, because they must exist when modeling a swarm-based algorithm. In other words, these components can theoretically make up a most basic swarm-based algorithm, if we don't consider whether the algorithm performance is excellent. Meanwhile, we consider the ***Information 18* is also a compulsory component**, although not all algorithms use it in this work. Because firstly, selection process is an important process in natural evolution. Secondly, if one algorithm doesn't use selection method, it means that all individuals are delivered into the next generation round, which is also a kind of selection method. More details are as follows:

- *Information 1, 3, 4*: the objective function $f()$ must exist to determine where the algorithm will happen. When the $f()$ is determined, the dimension n and the boundary $[lb_{\mathbf{x}}, ub_{\mathbf{x}}]$ are also determined which is because they exist in the $f()$.
- *Information 2, 5, 13, 15*: when searching possible solutions to $f()$, possible solutions $\mathbf{x}_i$ with the number M of $\mathbf{x}_i$ also must exist, because it determines the size of the search pool in which the algorithm could find the final optimal solution to f. Furthermore, the method $Init_{\mathbf{x}}$ to initialize the initial possible solutions also must exist, because it determines where the algorithm starts to find the final optimal solution, which means there also must be a method $Opt_{\mathbf{x}}$ that can iteratively optimize the initial possible solutions.
- *Information 17, 18, 19, 20*: because these swarm-based optimization algorithms are a kind of iterative optimization heuristic algorithms in which sampling and repetition play an important role in finding optimal solutions [7], the method S related to

sampling and the stop condition T must exist. Moreover, these algorithms will solve the problem in a limited search space, there must also be a method C to deal with outliers outside the search space.

Therefore, we conclude that f, $\mathbf{x}_i$, M, T and methods $Init_{\mathbf{x}}$, $Opt_{\mathbf{x}}$, C, S must exist in a swarm-based optimization algorithm. As shown in the left side of Figure 2.1, these compulsory components exist in all of seven selected algorithms.

In Table 2.1, ***Information 6 to 12, 14, 16* are selective components**, because not every algorithm needs these information. We find these information plays a similar role in modeling swarm-based algorithms, and the role is that these information acts on compulsory components to improve the performance of the algorithm. **Considering the role of these information is to influence how far the newly generated individual will move, we define these selective components step-size with notation Δ.** Meanwhile, we consider the ***Information 11* is also a selective component**, because its main role is also to the movement of individuals, although it exists in every algorithm. Furthermore, we further categorize these selective components as as follows:

- *Information 11*: Static numerical w-relative step-size $\Delta : w, z^0, [lb_{\mathbf{y}}, ub_{\mathbf{y}}]$.
  Most static numerical step-size, such as w, is able to be understood as common hyper-parameters in most algorithms. They are numbers; are set before running algorithms and are unchanged during optimization. In Figure 2.1, we find there are two groups of static numerical step-size. The first group is w that is commonly seen as hyper-parameters. The second group is the initial value of dynamic step-size (see next item ▪) in which the dynamic numerical step-size z needs an initial value z^0, the $\mathbf{y}$-relative dynamic vector step-size $\mathbf{y}_i$ needs an initial value $[lb_{\mathbf{y}}, ub_{\mathbf{y}}]$.

- *Information 6, 7, 8, 9, 12*: Dynamic step-size $\Delta : z, \mathbf{x}_{i_p}, \mathbf{x}_g, \mathbf{x}_s, \mathbf{y}_i$.
  - *Information 12*: Dynamic numerical z-relative step-size $\Delta : z$.
    This kind of step-size is a single numeric that is changing over the iteration t during the optimization process.
  - Dynamic vector step-size $\Delta : \mathbf{x}_{i_p}, \mathbf{x}_g, \mathbf{x}_s, \mathbf{y}_i$.
    This kind of step-size is a vector with the same dimension as $\mathbf{x}$.
    - *Information 6, 7, 8*: $\mathbf{x}$-relative step-size $\Delta : \mathbf{x}_{i_p}, \mathbf{x}_g, \mathbf{x}_s$.
      This kind of step-size can be converted from the target $\mathbf{x}$. Specifically, there could be a straightforward formula between $\mathbf{x}$ and $\mathbf{x}$-relative step-size. For example, personal best position $\mathbf{x}_{i_p}$, global best position $\mathbf{x}_g$. The $\mathbf{x}_s$ denotes a special $\mathbf{x}$-relative step-size related to $\mathbf{x}$, for example, the sorted population in MFO.

- *Information 9*: $\mathbf{y}$-relative step-size $\Delta : \mathbf{y}_i$.
  This kind of step-size is not related to the target $\mathbf{x}_i$. In other words, it is impossible to represent this kind of step-size by translating other existing variables in a straightforward method. Normally, this kind of step-size has same size as the population of individuals. For example, velocity $\mathbf{v}_i$.

For other selective components *Information 10, 14, 16*, because the dynamic step-size $\Delta : \mathbf{x}_{i_p}, \mathbf{x}_g, \mathbf{x}_s, \mathbf{y}_i$ is changing during the optimization process, there must be an initialization status and a changing status. Therefore, we also define $Init_\Delta$ and Opt_Δ to achieve the initialization status and the changing status for dynamic step-size, in which the $Init_\Delta$ method can also include the set-up of static w-relative step-size $\Delta : w, z^0, [lb_\mathbf{y}, ub_\mathbf{y}]$. As shown in the right side of Figure 2.1, all of these step-size Δ have an initial status determined by $Init_\Delta$, and each dynamic step-size $\Delta : z, \mathbf{x}_{i_p}, \mathbf{x}_g, \mathbf{x}_s, \mathbf{y}_i$ will change by an optimization operator Opt_Δ.

2.3.2 Extension in Other Swarm-based Algorithms.

This section discusses the possibility of applying these unified terminologies in any other swarm-based algorithm. First of all, all of compulsory components (see *Information 1 to 5, 13, 15, 17-20* in Table 2.1) and several selective components (see *Information 6, 7, 11, 12* in Table 2.1) are easily detected in any one swarm-based algorithm, because they are straightforward. For example, $\mathbf{x}$ is the animal, w is the hyper-parameter and z is also the hyper-parameter but z is changing. Mostly any one swarm-based algorithm will use very clear noun or terminology to point out these information.

The place where it is most likely to confuse readers is **how to detect the dynamic vector step-size: Why we think one vector step-size is a $\mathbf{x}$-relative step-size not a $\mathbf{y}$-relative step-size?** This question can be answered according to the observations found in these seven algorithms.

In most of swarm-based algorithms, besides the animal $\mathbf{x}$, another noun is also always existing. This noun could be the velocity (in BA, PSO), the memory (in CSA) or the flame (in MFO). We are not allowed to directly assign them the $\mathbf{y}$-relative step-size, although this is the most straightforward way. **The reason is that the information carried by the noun is the only principle used to assign them different unified terminology name.** For example:

- The memory $\mathbf{m}_i$ in CSA is initialized by $\mathbf{x}_i$ itself as $\mathbf{m}_i(t=0) = \mathbf{x}_i(t=0)$, and then will be updated by $\mathbf{m}_i(t+1) = \mathbf{Min}\{\mathbf{m}_i(t), \mathbf{x}_i(t+1)\}$. Therefore in the first iteration, the update will happen as Eq.2.36. In other words, the $\mathbf{m}_i$ always can be represented by $\mathbf{x}_i$ all the time. Moreover, the update method to the $\mathbf{m}_i$ is same to find the $\mathbf{x}_{i_p}$, therefore, it is totally safe to replace $\mathbf{m}_i$ by $\mathbf{x}$-relative vector $\mathbf{x}_{i_p}$rather than the $\mathbf{y}$-relative vector $\mathbf{y}_i$.

$$\begin{aligned} \mathbf{m}_i(t=1) &= \mathbf{Min}\{\mathbf{m}_i(t=0), \mathbf{x}_i(t=1)\} \\ &= \mathbf{Min}\{\mathbf{x}_i(t=0), \mathbf{x}_i(t=1)\} \\ &= \mathbf{x}_i(t=0) \text{ or } \mathbf{x}_i(t=1) \end{aligned} \tag{2.36}$$

- The flame $\langle flame_i \rangle$ in MFO is initialized by $\langle \mathbf{x}_i \rangle$ itself as $\langle flame_i \rangle(t=0) = \langle \mathbf{x}_i \rangle(t=0)$, and then will be updated by $\langle flame_i \rangle(t+1) = \mathbf{Sort}(\{\mathbf{x}_i(t)\} \cup \{\mathbf{x}_i(t+1)\}), i = 1 \dots M$. Therefore, in the first iteration, the update will happen as Eq.2.37. In other words, the $\langle flame_i \rangle$ always can be represented by $\mathbf{x}$-relative vectors $\langle \mathbf{x}_i \rangle$ all the time, rather than the $\mathbf{y}$-relative vectors $\langle \mathbf{y}_i \rangle$.

$$\begin{aligned} \langle flame_i \rangle(t=1) &= \mathbf{Sort}(\langle flame_i \rangle(t=0) \cup \{\mathbf{x}_i(t=1)\}), i = 1 \dots M \\ &= \mathbf{Sort}(\langle \mathbf{x}_i \rangle(t=0) \cup \{\mathbf{x}_i(t=1)\}), i = 1 \dots M \end{aligned} \tag{2.37}$$

- The velocity in BA and PSO uses a different way to initialize itself, such as $\boldsymbol{\mathcal{U}}(lb, ub)$. Moreover, the most important point is there is no way to represent the velocity by $\mathbf{x}$-relative vector $\mathbf{x}_i$. Therefore, we prefer assigning $\mathbf{y}$-relative vector $\mathbf{y}_i$ to this kind of step-size.

After resolving the confusion between $\mathbf{x}$-relative vector step size and $\mathbf{y}$-relative vector step size, other *Information 8 to 10, 14, 16* also become clear to detect in any other swarm-based algorithms according to the previous observations.

2.4 Summary

In this chapter, we discuss the commonalities among terminologies in seven selected algorithms. We summarize their commonalities in Table 2.1 in which we conclude 20 unified components that can cover entire various components in these algorithms. Meanwhile, we discuss the classification of these 20 general components in Figure 2.1.

In Subsection 2.3.1, we detail the classification of these 20 general components. Every algorithm must have the target problem f, the search space n and $[lb_{\mathbf{x}}, ub_{\mathbf{x}}]$, the possible target solution $\mathbf{x}_i$, the population size M, the stop condition T, the initialization method $Init_{\mathbf{x}}$ to $\mathbf{x}_i$, the optimization method $Opt_{\mathbf{x}}$ to $\mathbf{x}_i$, the method C to deal with outliers and the method S to select which individuals could be delivered into the next generation.

However, selective components are more flexible, and these selective components are all related to determine how far the newly generated individual will move. Therefore, we name selective components step-size Δ. These selective components can be further categorized into static numerical w-relative step-size $\Delta : w, z^0, [lb_{\mathbf{y}}, ub_{\mathbf{y}}]$ and dynamic step-size $\Delta : z, \mathbf{x}_{i_p}, \mathbf{x}_g, \mathbf{x}_s, \mathbf{y}_i$. What is interesting is the dynamic step-size also has two categories: dynamic numeric z-relative step-size $\Delta : z$ and dynamic vector step-size $\Delta : \mathbf{x}_{i_p}, \mathbf{x}_g, \mathbf{x}_s$ that are $\mathbf{x}$-relative step-size and $\Delta : \mathbf{y}_i$ that is $\mathbf{y}$-relative step-size.

In Subsection 2.3.2, we discuss how to extend these unified terminologies into any other swarm-based algorithm. Furthermore, we resolve a confusion of the difference between $\mathbf{x}$-relative step-size and $\mathbf{y}$-relative step-size.

In conclusion, we provide a catalog of unified terminologies for seven selected algorithms.

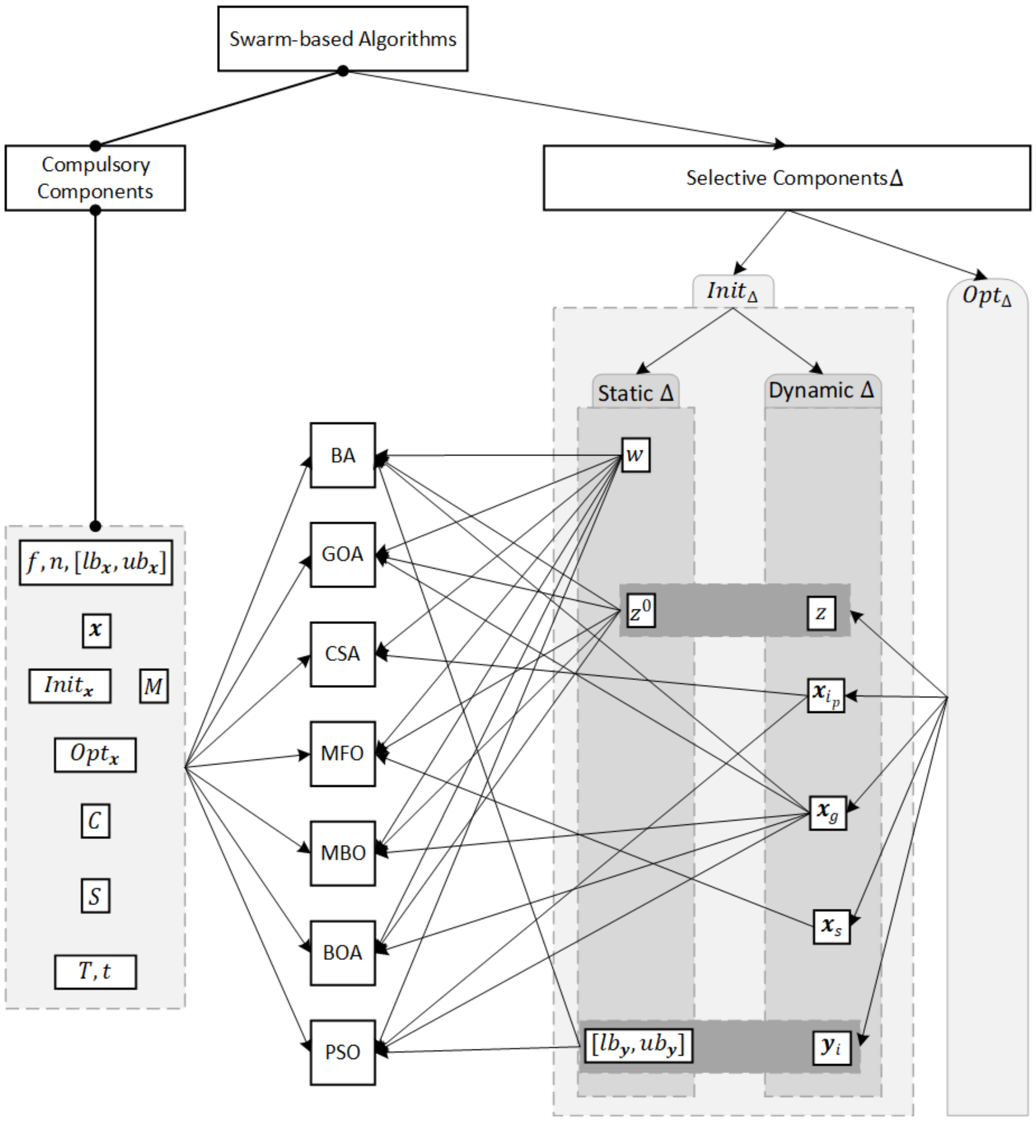

Figure 2.1: Entire compulsory components and some of selective components can comprise any of these seven algorithms.

Table 2.1: Unified Terminologies: all information appearing in these seven algorithms can be analyzed and then categorized into 20 components. The none in gray color means this information doesn't appear in this algorithm.

Information	Different Terminologies	Unified Terminology
1.The objective optimization problem.	The representations in seven algorithms are same.	$f()$: optimization function.
2.One possible objective solution.	•BA: each bat $\mathbf{x}_i$. •GOA: each grasshopper $\mathbf{x}_i$. •CSA: crow $\mathbf{x}_i$. •MFO: moth $\mathbf{m}_i$. •MBO: monarch butterfly $\mathbf{x}_i$. •BOA: butterfly $\mathbf{x}_i$. •PSO: particle $\mathbf{x}_i$.	$\mathbf{x}_i$: one objective solution.
3.The number of independent elements in $\mathbf{x}_i$.	•BA: d. •GOA: dim. •CSA: d. •MFO: dim. •MBO: k. •BOA: dim. •PSO: dimension.	n: dimension of the $f()$ search space.
4.The boundary of each independent element in $\mathbf{x}_i$[21].	•BA: Lb/Ub. •GOA: $down/up$. •CSA: l/u. •MFO: lb/ub. •MBO: $MinParValue/MaxParValue$. •BOA: Lb/Ub. •PSO: lb/ub.	$lb_{\mathbf{x}}/ub_{\mathbf{x}}$: the lower/upper boundary of all elements in $\mathbf{x}_i$.
5.The number of $\mathbf{x}_i$.	•$BA:$ n. •GOA: N. •CSA: N. •MFO: N. •MBO: M. •BOA: n. •PSO: N.	M: population size.
6.The best $\mathbf{x}_i$ that it itself has found so far. It will change during the optimization process.	•BA :none. •BA :none. •GOA: none. •CSA: memory $\mathbf{m}_i$. •MFO: none. •MBO: none. •BOA: none. •PSO: $pbest$.	$\mathbf{x}_{i_p}$: $\mathbf{x}$-relative step-size Δ.
7.The best $\mathbf{x}_i$ that the entire population has found so far. It will change during the optimization process.	•BA: $\mathbf{x}_*$. •GOA: $\mathbf{T}$. •CSA: none. •MFO: none. •MBO: $\mathbf{x}_{best}$. •BOA: g^*. •PSO: $gbest$.	$\mathbf{x}_g$: $\mathbf{x}$-relative step-size Δ.

[21]Some algorithms, such as GOA, MFO, consider the boundary of every element is different, however, we only consider the boundary of every element is same in this work.

8.Special vector step-size that is able to be represented by $\mathbf{x}_i$. It will change during the optimization process.	•BA: none. •GOA: none. •CSA: none. •MFO: sorted population. •MBO: none. •BOA: none. •PSO: none.	$\mathbf{x}_s$: $\mathbf{x}$-relative step size Δ.
9.Special vector step-size that is not able to be represented by $\mathbf{x}_i$. It will change during the optimization process.	•BA:velocity $\mathbf{v}_i$. •GOA: none. •CSA: none. •MFO: none. •MBO: none. •BOA: none. •PSO: velocity $\mathbf{v}_i$.	$\mathbf{y}_i$: $\mathbf{x}$-relative step-size Δ.
10.The boundary of each independents element in $\mathbf{y}_i$.	•BA: Eq.2.2. •GOA: none. •CSA: none. •MFO: none. •MBO: none. •BOA: none. •PSO: Eq.2.32.	$lb_{\mathbf{y}}/ub_{\mathbf{y}}$: the lower/upper boundary of all elements in $\mathbf{y}_i$.
11.step-size that is set before running algorithm. They are unchanged during the optimization process.	•BA: echo frequency interval $[freq\_min, freq\_max]$, 3 decreasing factors ϵ, α, γ. •GOA: attractive intensity f, attractive length l. •CSA: awareness probability AP, flight length fl. •MFO: spiral shape b. •MBO: $Keep$, $partition$, $period$, BAR, S_{max}. •BOA: switch probability p, $power\_exponent$, $sensory\_modality$. •PSO: adjusting parameter w, cognitive coefficient c_1, social coefficient c_2.	w: w-relative step-size Δ.
12.step-size whose initial values are set before running algorithm. They will change by math formulas during the optimization process.	•BA: loudness A, pulse rate $rate$. •GOA: coefficient c. •CSA: none. •MFO: decreasing weight tt, threshold $Fame\_no$. •MBO: weight α. •BOA: sensory_modality $sensory\_modality$. •PSO: none.	z: z-relative step-size Δ.
13.How to initialize $\mathbf{x}_i$ with size M in the n dimension environment.	•BA: Eq.2.1. •GOA: Eq.2.7. •CSA: Eq.2.12. •MFO: Eq.2.16. •MBO: Eq.2.21. •BOA: Eq.2.26. •PSO: Eq.2.31.	$Init_{\mathbf{x}}$: initialization method on $\mathbf{x}$.

14.How to initialize changing step-size.	•BA: Eq.2.2,step(4)(6)(7). •GOA: step(3),Eq.2.8. •CSA: Eq.2.13. •MFO: Eq.2.17,Eq.2.18,step(4). •MBO: step(6),Eq.2.23. •BOA: step(3),Eq.2.27. •PSO: Eq.2.32,Eq.2.33,step(5).	$Init_{\Delta}$: initialization method on Δ.
15.How to update $\mathbf{x}_i$.	•BA: Eq.2.5,Eq.2.6. •GOA: Eq.2.10. •CSA: Eq.2.14. •MFO: Eq.2.19,Eq.2.20. •MBO: step(5)(6),Eq.2.22. •BOA: Eq.2.28,Eq.2.29. •PSO: Eq.2.35.	$Opt_{\mathbf{x}}$: optimization method on $\mathbf{x}$.
16.How to update changing step-size.	•BA: Eq.2.3,Eq.2.4,Eq.2.5. •GOA: Eq.2.8,step(9). •CSA: Eq.2.15. •MFO: Eq.2.17,Eq.2.18, step(4). •MBO: Eq.2.23,step(5). •BOA: step(3),Eq.2.30.•PSO: Eq.2.34,sep(7)(8).	Opt_{Δ}: optimization method on Δ.
17.How to deal with outliers in $\mathbf{x}_i$.	•BA: step(10). •GOA: step(7). •CSA: step(6). •MFO: step(3). •MBO: step(7). •BOA: step(7). •PSO: common method.	C: outliers treatment in $\mathbf{x}_i$.
18.How to decide if a updated $\mathbf{x}_i$ would be accepted.	•BA: step(11). •GOA: none. •CSA: none. •MFO: none. •MBO: step(11). •BOA: step(8). •PSO: none.	S: selection method to select which individuals will be delivered into the next generation.
19.The iteration counter.	•BA: t. •GOA: l. •CSA: t. •MFO: t. •MBO: t. •BOA: t. •PSO: current iteration.	t: current iteration.
20.The maximum number of iterations.	•BA: t_max. •GOA: $max\_iteration$. •CSA: $tmax$. •MFO: T. •MBO: $Naxgen$. •BOA: N_iter. •PSO: stop condition.	T: stop condition.

Chapter 3

Unified Procedure

3.1 Introduction

This chapter discusses the commonality among these seven selected algorithms on the level of their procedures for achieving optimization. In other words, we discuss how these 20 unified components make up a valid algorithm. Specifically, we are interested in whether these seven selected algorithms can arrange these 20 components in the same order.

In Chapter 2, we have discussed that these algorithms are able to be composed of 20 unified components, and these components can be categorized into compulsory and selective components according to their information. In this chapter, we also discuss the category of these components, but the principle of categorizing becomes their functions. This is because the function of each component determines their positions when constructing a valid nature-inspired algorithm. As shown in Table 3.1, we find that **these 20 unified components can be categorized into eight tuples, and these eight tuples are enough to construct any one of these seven selected algorithms**. Therefore, these selected swarm-based optimization algorithms $NIOA$[1] can be defined in 8-tuple as Eq.3.1.

$$NIOA = (f, Init_{\mathbf{x}}, Opt_{\mathbf{x}}, C, T, S, Init_{\Delta}, Opt_{\Delta}) \tag{3.1}$$

where

[1]We expect our framework can be applied in all nature-inspired algorithms, therefore, the $NIOA$ is short for Nature-Inspired Optimization Algorithm

- *Tuple.1*: 'Evaluation' function f:
  'Evaluation' is needed to measure the quality of solutions that are found by the algorithm to the problem, when modeling a valid swarm-based algorithm.
- *Tuple.2*: 'Initialize population' function $Init_{\mathbf{x}}$:
  'Initialize population' provides a set of possible solutions. A swarm-based algorithm will improve this set and find the optimal one in this set.
- *Tuple.3*: 'Initialize step-size' function $Init_{\Delta}$:
  'Initialize step-size' helps the swarm-based algorithm to improve the quality of possible solutions efficiently. In Chapter 2, we have discussed that there are four kinds of step-size Δ: w-relative Δ, z-relative Δ, $\mathbf{x}$-relative Δ and $\mathbf{y}$-relative Δ.
- *Tuple.4*: 'Update population' function $Opt_{\mathbf{x}}$:
  'Update population' is the strategy that the algorithm used to improve the quality of possible solutions to the problem.
- *Tuple.5*: 'Update step-size' function Opt_{Δ}:
  If the step-size Δ used by the algorithm is dynamically changing, such as z-relative Δ, $\mathbf{x}$-relative Δ and $\mathbf{y}$-relative Δ, 'Update step-size' function is needed to implement this mechanism of changing.
- *Tuple.6*: 'Outliers treatment' function C:
  'Outliers treatment' deals with every outlier when the swarm-based algorithm updates the population.
- *Tuple.7*: 'Selection' function S:
  'Selection' determines how many updated individuals will be delivered into the next generation round, when the nature-inspired algorithm iteratively updates the population.
- *Tuple.8*: 'Stop strategy' function T:
  'Stop strategy' determines when the iterative optimization process stops.

Table 3.1: These 20 unified components can be categorized into eight tuples, according to their functions. Numbers in the 'Component' column represents the index of information disccused in Table 2.1

Tuple	Component		Function	
Tuple.1	1.f		evaluation	
Tuple.2	2.$\mathbf{x}_i$ 3.n 4.$lb_\mathbf{x}/ub_\mathbf{x}$ 5.M 13.$Init_\mathbf{x}$		initialize the initial population.	
Tuple.3	14.$Init_\Delta$	6.$\mathbf{x}_{i_p}$ 7.$\mathbf{x}_g$ 8.$\mathbf{x}_s$	initialize $\mathbf{x}$-relative step-size Δ.	initialize step-size Δ.
		9.$\mathbf{y}_i$ 10.$lb_\mathbf{y}/ub_\mathbf{y}$	initialize $\mathbf{y}$-relative step-size Δ.	
		11.w	initialize w-relative step-size Δ.	
		12.z	initialize z-relative step-size Δ.	
Tuple.4	15.$Opt_\mathbf{x}$		update the population.	
Tuple.5	16.Opt_Δ		update step-size Δ.	
Tuple.6	17.C		outliers treatment.	
Tuple.7	18.S		deliver selected individuals into next generation.	
Tuple.8	19.t 20.T		iteration counter, and the maximum number of generation	

In the following sections, Section 3.2 discusses whether these eight tuples can be ordered at the same position in these seven selected algorithms. The overview summary about this chapter is displayed in Section 3.3.

3.2 Unified Procedure

As observed in Table 3.2, we find that each algorithm only has two processes — Initialization process and Optimization process — separated by a stop strategy *Tuple.8* T.

In the Initialization process, *Tuple.2* $Init_{\mathbf{x}}$, *Tuple.1* f and *Tuple.3* $Init_{\Delta:w,z,\mathbf{y},\mathbf{x}}$ exist in every algorithm except the MFO. The MFO does not evaluate the initial population in the initialization process. However, in MFO's optimization process, the *Tuple.1* f happens before updating the population, which means the initial population also will be evaluated, but it will be evaluated in the optimization process. Therefore, considering the swarm-based algorithm is a kind an iterative optimization heuristic algorithm, the feature of iteration leads the Initialization process can have one *Tuple.1* f function, and then the *Tuple.1* f function in the optimization process has to be moved after updating the population.

After making this change, in the Optimization process, we can conclude that *Tuple.4* $Opt_{\mathbf{x}}$, *Tuple.6* C and *Tuple.1* f also exist in every algorithm, and the order of them must be that *Tuple.4* $Opt_{\mathbf{x}}$ is before *Tuple.6* C followed by *Tuple.1* f. Because outliers only might appear after updating the population. Furthermore, three places are also meaningful to be unified:

(1) The position of *Tuple.3* $Init_{\Delta:z}$ and *Tuple.5* $Opt_{\Delta:z}$ in BA, GOA, MFO and MBO. As shown in Table 3.2, the position of *Tuple.3* $Init_{\Delta:z}$ and *Tuple.5* $Opt_{\Delta:z}$ in these three algorithms are together assigned at the beginning of the optimization process. However, we can find that the generalized math equation of *Tuple.3* $Init_{\Delta:z}$ and *Tuple.5* $Opt_{\Delta:z}$ has two forms. For example,

- In BA, the generalized math equation G_1 of one z is $z(t+1) = G_1(z(t))$ which means the *Tuple.3* $Init_{\Delta:z}$ is assigning an initial value z^0 to $z(t=0)$ and can move from the beginning of optimization process to the initialization process. Meanwhile, the *Tuple.5* $Opt_{\Delta:z}$ has to move to the end of the optimization process. These two assignments are same as in the BOA, where *Tuple.3* $Init_{\Delta:z}$ will go to the initialization process, and *Tuple.5* $Opt_{\Delta:z}$ will go to the end of the optimization process.
- In BA, GOA, MFO and MBO, there is another generalized math equation G_2 of z is $z(t) = G_2(z(t), t)$ which means both *Tuple.3* $Init_{\Delta:z}$ and *Tuple.5* $Opt_{\Delta:z}$ are a function of iteration counter t. It means it doesn't matter where *Tuple.3* $Init_{\Delta:z}$ and *Tuple.5* $Opt_{\Delta:z}$ are, as far as the iteration counter t is input correctly. Therefore, we can move *Tuple.3* $Init_{\Delta:z}$ to the initialization process, and move *Tuple.5* $Opt_{\Delta:z}$ to the end of the optimization process.

(2) The position of *Tuple.3* $Init_{\Delta:\mathbf{x}}$ and *Tuple.5* $Opt_{\Delta:\mathbf{x}}$ in MFO.
The original position of *Tuple.3* $Init_{\Delta:\mathbf{x}}$ and *Tuple.5* $Opt_{\Delta:\mathbf{x}}$ are together at the beginning of the optimization process. After looking at the implement code in MFO, we can make sure the position of them can be separated. The *Tuple.3* $Init_{\Delta:\mathbf{x}}$ can be in the initialization process, and the *Tuple.5* $Opt_{\Delta:\mathbf{x}}$ can move to the end of the optimization process.

(3) The position of *Tuple.5* $Opt_{\Delta:\mathbf{y}}$ and *Tuple.5* $Opt_{\Delta:\mathbf{x}}$ in all of these selected algorithms.
Generally speaking, it would be more reasonable if these two tuples can be at the same position. However, updated $\mathbf{y}$-relative Δ is used in the current generation round, but updated $\mathbf{x}$-relative Δ is used in the next generation round. Therefore, the position of *Tuple.5* $Opt_{\Delta:\mathbf{y}}$ must be before *Tuple.4* $Opt_{\mathbf{x}}$, and the position of *Tuple.5* $Opt_{\Delta:\mathbf{x}}$ must be after *Tuple.1* f in the optimization.

Meanwhile, we also find that in the Optimization process, *Tuple.7* S does not exist all of these selected algorithms, such as GOA, CSA, MFO and PSO. However, considering the function of *Tuple.7* S is to select which individuals can be delivered into the next generation round which is a meaningful part in swam-based algorithms, we prefer assigning *Tuple.7* S to all of these selected algorithms. For these algorithms who does not use *Tuple.7* S, we define a kind of selection method in which all updated individuals will be delivered into the next generation round. (see Chapter 4 for more details about this special selection method).

Table 3.2: Original positions of these eight tuples in these seven algorithms

Process	BA	GOA	CSA	MFO	MBO	BOA	PSO
Initialization	*Tuple.2* $Init_{\mathbf{x}}$	*Tuple.2* $Init_{\mathbf{x}}$	*Tuple.2* $Init_{\mathbf{x}}$	*Tuple.2* $Init_{\mathbf{x}}$	*Tuple.2* $Init_{\mathbf{x}}$	*Tuple.2* $Init_{\mathbf{x}}$	*Tuple.2* $Init_{\mathbf{x}}$
	Tuple.1 f	*Tuple.1* f	*Tuple.1* f	*Tuple.3* $Init_{\Delta:w}$	*Tuple.1* f	*Tuple.1* f	*Tuple.1* f
	Tuple.3 $Init_{\Delta:w}$	*Tuple.3* $Init_{\Delta:w}$	*Tuple.3* $Init_{\Delta:w}$		*Tuple.3* $Init_{\Delta:w}$	*Tuple.3* $Init_{\Delta:z}$	*Tuple.3* $Init_{\Delta:w}$
	Tuple.3 $Init_{\Delta:\mathbf{y}}$	*Tuple.3* $Init_{\Delta:\mathbf{x}}$	*Tuple.3* $Init_{\Delta:\mathbf{x}}$		*Tuple.3* $Init_{\Delta:\mathbf{x}}$	*Tuple.3* $Init_{\Delta:\mathbf{x}}$	*Tuple.3* $Init_{\Delta:\mathbf{x}}$

continued . . .

. . . continued

Process	BA	GOA	CSA	MFO	MBO	BOA	PSO
	Tuple.3 $Init_{\Delta:\mathbf{x}}$						*Tuple.3* $Init_{\Delta:\mathbf{y}}$
Stop strategy *Tuple.8* T							
Optimization	*Tuple.3* $Init_{\Delta:z}$ and *Tuple.5* $Opt_{\Delta:z}$	*Tuple.3* $Init_{\Delta:z}$ and *Tuple.5* $Opt_{\Delta:z}$	*Tuple.4* $Opt_{\mathbf{x}}$	*Tuple.3* $Init_{\Delta:z}$ and *Tuple.5* $Opt_{\Delta:z}$	*Tuple.3* $Init_{\Delta:z}$ and *Tuple.5* $Opt_{\Delta:z}$	*Tuple.4* $Opt_{\mathbf{x}}$	*Tuple.5* $Opt_{\Delta:\mathbf{y}}$
	Tuple.5 $Opt_{\Delta:\mathbf{y}}$	*Tuple.4* $Opt_{\mathbf{x}}$	*Tuple.6* C	*Tuple.6* C	*Tuple.4* $Opt_{\mathbf{x}}$	*Tuple.6* C	*Tuple.4* $Opt_{\mathbf{x}}$
	Tuple.4 $Opt_{\mathbf{x}}$	*Tuple.6* C	*Tuple.1* f	*Tuple.1* f	*Tuple.1* f	*Tuple.1* f	*Tuple.6* C
	Tuple.6 C	*Tuple.1* f	*Tuple.5* $Opt_{\Delta:\mathbf{x}}$	*Tuple.3* $Init_{\Delta:\mathbf{x}}$ and *Tuple.5* $Opt_{\Delta:\mathbf{x}}$	*Tuple.4* $Opt_{\mathbf{x}}$	*Tuple.7* S	*Tuple.1* f
	Tuple.1 f	*Tuple.5* $Opt_{\Delta:\mathbf{x}}$		*Tuple.4* $Opt_{\mathbf{x}}$	*Tuple.1* f	*Tuple.5* $Opt_{\Delta:\mathbf{x}}$	*Tuple.5* $Opt_{\Delta:\mathbf{x}}$
	Tuple.7 S				*Tuple.7* S	*Tuple.5* $Opt_{\Delta:z}$	
	Tuple.5 $Opt_{\Delta:\mathbf{x}}$				*Tuple.5* $Opt_{\Delta:\mathbf{x}}$		

Based on these previous discussions, the unified procedure for these seven selected algorithms can be formulated as Algorithm.1. As the Algorithm.1 described, every algorithm will start at the Initialization process in which the population must be evaluated after being initialized, moreover, different algorithms initialize $Init_{\Delta}$ differently. For example, CSA only uses w-relative and $\mathbf{x}$-relative step-size Δ, but PSO uses one more $\mathbf{y}$-relative step-size Δ.

In the optimization process, if $\mathbf{y}$-relative step-size Δ exists in the initialization process, it will be firstly updated before updating the population. Evaluation happens after dealing with outliers in the new generated population. Selection method is possible existing in all of these seven algorithms, and it must exit after evaluation action. The last step in the optimization process is to update z-relative and $\mathbf{x}$-relative step-size Δ, if they exist in the initialization process.

Algorithm 1 Unified Procedure for seven selected algorithms (the unified positions of eight tuples in Table 3.1)

```
1:  $t \leftarrow 0$                                  ▷ iteration counter
2:  Tuple.2 $Init_{\mathbf{x}}$                        ▷ initialize population
3:  Tuple.1 $f$                                      ▷ evaluation
4:  Tuple.3 $Init_{\Delta:w,z,\mathbf{x},\mathbf{y}}$  ▷ initialize $w$-relative, $z$-relative, $\mathbf{x}$-relative and $\mathbf{y}$-relative step-size $\Delta$
5:  while Tuple.8 $T$ do                             ▷ stop strategy
6:      Tuple.5 $Opt_{\Delta:\mathbf{y}}$  ▷ update $\mathbf{y}$-relative step-size $\Delta$, if $\mathbf{y}$-relative step-size exsit in the initialization process
7:      Tuple.4 $Opt_{\mathbf{x}}$                     ▷ update population
8:      Tuple.6 $C$                                  ▷ outliers treatment
9:      Tuple.1 $f$                                  ▷ evaluation
10:     Tuple.7 $S$                                  ▷ selection
11:     Tuple.5 $Opt_{\Delta:z,\mathbf{x}}$            ▷ update $z$-relative and $\mathbf{x}$-relative step-size $\Delta$
12:     $t \leftarrow t + 1$
13: end while
```

3.3 Summary

In this chapter, we discuss the possibility to unify the various procedures among these seven selected algorithm. We firstly categorize the 20 unified components (see Chapter 2) again, but the principle becomes their functions in this chapter. As summarized in Table 3.1, all of these seven algorithms can be represented by eight tuples, and the unified representation is defined as Equation 3.1. Secondly, we detail each algorithm's original position of each tuple, and Table 3.2 displays all of these original positions. Then, we discuss the possibility of arranging these eight tuples in the same positions in these seven algorithms. Lastly, the final unified procedures is performed in Algorithm.1.

In conclusion, all of these seven algorithms can be represented by eight unified tuples, and these eight tuples can be at the same position when modeling these seven algorithms.

Chapter 4

Unified Framework

4.1 Motivation

In this chapter, we aim to complete the purpose in this work — **building up a unified framework for selected algorithms** — by combing the Unified Terminologies (see Chapter 2) and the Unified Procedure (see Chapter 3) together. Specifically, **we focus on studying how these eight tuples (see Chapter 3) are represented in each algorithm by using unified terminologies (see Chapter 2), because the order of these eight tuples is already fixed in Chapter 3**.

Firstly, we want to introduce several special symbols used in this work. As we discussed in Chapter 2, the swarm-based algorithm happens on a set of individuals $\mathbf{x}_i$, therefore, we use $\mathbf{X}$ and $\mathbf{x}_i$ to distinguish the population and the individual. For example, the set of personal best individual $\mathbf{x}_{i_p}$ is $\mathbf{X}_p$, but there is only one global best individual $\mathbf{x}_g$. Meanwhile, the capital letter F is used to represent the set of the fitness of the whole population $\mathbf{X}$.[1]

Moreover, we use $\hat{\mathbf{X}}$ and $\mathbf{X}$ to distinguish the population updated before selection operation and the population updated after selection operation. We also want to clarify that the population updated after the selection operation is the finally updated population after one generation round, on the contrary, the population updated before the selection operation is the temporarily updated population.

[1]The capital letter in bold style denotes a set of vector. The capital letter in plain style denotes a set of numerics.

In the following sections, Section 4.2 introduces how these original representations of each algorithm is represented with only eight tuples. We also mention difficulties when re-framing the original model into these eight tuples. Next, in Section 4.3, we display the Unified Nature-Inspired Optimization Algorithm (UNIOA) and give discussions. The summary about this chapter is shown in Section 4.4.

4.2 Re-framed Nature-inspired Algorithms

4.2.1 Re-framed BA

The *Tuple.1* f can be represented as $f(\mathbf{x}_i)$. The *Tuple.2* $Init_{\mathbf{x}}$ can be extracted from Eq.2.1, and re-framed as Eq.4.1.

$$\mathbf{x}_i(t=0) = \boldsymbol{\mathcal{U}}(lb_{\mathbf{x}}, ub_{\mathbf{x}}), i = 1, 2, \dots, M \qquad 4.1$$

For the *Tuple.3* $Init_{\Delta}$, as discussed in Table 2.1, we find the velocity $\mathbf{v}_i$ is a kind of $\mathbf{y}$-relative step-size, and the global best individual $\mathbf{x}_*$ is a kind of $\mathbf{x}$-relative step-size. Meanwhile, the loudness A and the rate r are a kind of z-relative step-size. Therefore, we re-frame $\mathbf{v}_i$ (see step (2) in Subsection 2.2.1) into Eq.4.2. The $\mathbf{x}_*$ (see step (4) in Subsection 2.2.1) can be re-framed as Eq.4.3. The two z-relative step-size A and r can be respectively re-framed as Eq.4.4 and Eq.4.5.

$$\mathbf{y}_i(t=0) = \boldsymbol{\mathcal{U}}(lb_{\mathbf{y}}, ub_{\mathbf{y}}), i = 1 \dots M \qquad 4.2$$

$$\mathbf{x}_g(t=0) = \mathbf{Min}(\{\mathbf{x}_i(t=0)\}), i = 1 \dots M \qquad 4.3$$

$$z_1(t=0) = z_1^0 \times w_2 \qquad 4.4$$

$$z_2(t=0) = z_2^0 \times (1 - e^{-w_1 \times t}) \qquad 4.5$$

For the *Tuple.4* $Opt_{\mathbf{x}}$, we find step (8) is able to be re-framed into one equation as Eq.4.6.

$$\hat{\mathbf{x}}_i(t+1) = \begin{cases} \mathbf{x}_g(t) + w_3 \times \mathbf{rand} \times z_1(t), & rand < z_2(t) \\ \mathbf{x}_i(t) + \mathbf{y}_i(t+1), & \text{o.w} \end{cases} \qquad 4.6$$

For the *Tuple.5* Opt_{Δ} that corresponds to *Tuple.3* $Init_{\Delta}$, we find the update way of $\mathbf{v}_i$ (see one step in step (8) in Subsection 2.2.1) is able to be re-framed into as Eq.4.7. The update way of $\mathbf{x}_*$ (see step (9) in Subsection 2.2.1) is able to be re-framed as Eq.4.8. The update way of A (see Eq.2.3) and the update way of r (see Eq.2.4) can be respectively re-framed as Eq.4.9 and Eq.4.10.

$$\mathbf{y_i}(t+1) = \mathbf{y_i}(t) + \mathcal{U}(lb_{w_4}, ub_{w_4}) \times (\mathbf{x}_i(t) - \mathbf{x}_g(t)) \qquad 4.7$$

$$\mathbf{x}_g(t+1) = \mathbf{Min}(\mathbf{x}_g(t) \cup \{\mathbf{x}_i(t+1)\}), i = 1 \dots M \tag{4.8}$$

$$z_1(t+1) = z_1(t) \times w_2 \tag{4.9}$$

$$z_2(t+1) = z_2^0 \times (1 - e^{-w_1 \times (t+1)}) \tag{4.10}$$

For the *Tuple.6* C, the step (10) can be re-framed as Eq.4.11.

$$\mathbf{x}_{i,n}^{\text{fixed}}(t+1) = \begin{cases} ub_{\mathbf{x}} & , \quad \mathbf{x}_{i,n}(t+1) > ub_{\mathbf{x}} \\ \mathbf{x}_{i,n}(t+1) & , \quad \text{o.w} \\ lb_{\mathbf{x}} & , \quad \mathbf{x}_{i,n}(t+1) < lb_{\mathbf{x}} \end{cases} \tag{4.11}$$

For the *Tuple.7* S, the step (11) can be re-framed as Eq.4.12.

$$\mathbf{x}_i(t+1) = \begin{cases} \hat{\mathbf{x}}_i(t+1) & , \quad f(\hat{\mathbf{x}}_i(t+1)) < f(\mathbf{x}_i(t)) \text{ or } rand > z_1(t) \\ \mathbf{x}_i(t) & , \quad \text{o.w} \end{cases} \tag{4.12}$$

For the *Tuple.8* T which is a stop condition related to the maximum number of iteration.

Till now, all of other unknown parameters can be defined as w-relative step-size, and they belong to the *Tuple.3* $Init_{\Delta}$. It has common w-relative step-size $w_1 = 0.1$, $w_2 = 0.97$, $[lb_{w_4}, ub_{w_4}] = [0,2]$,$w_3 = 0.1$ and $[lb_{\mathbf{y}}, ub_{\mathbf{y}}] = [0,0]$. It also has special w-relative step-size that is related to the initial value of z-relative step-size $z_1^0 = 1$, $z_2^0 = 1$.

Therefore, the complete pseudo-code of re-framed BA model can be organized as Algorithm.2. When we re-frame the BA algorithm, the main difficulty is how to formulate the z-relative step-size. For example, the z_1 (that is the loudness A in Subsection 2.2.1) is located at the beginning of optimization process. Therefore, when we move it to the initialization process, the updated z_1 has to be moved after updating the population.

Algorithm 2 Re-framed BA

```
 1: $t \leftarrow 0$
 2: $\mathbf{X}(t) \leftarrow Init_{\mathbf{x}}(n, M, [lb_{\mathbf{x}}, ub_{\mathbf{x}}])$ as Eq.4.1          ▷ initialize initial population
 3: $F(t) \leftarrow f(\mathbf{X}(t))$                                                               ▷ evaluate
 4: $w, [lb_{\mathbf{y}}, ub_{\mathbf{y}}], z^0 \leftarrow Init_{\Delta:w}(w, [lb_{\mathbf{y}}, ub_{\mathbf{y}}], z^0)$   ▷ initialize $w$-relative step-size
 5: $\mathbf{Y}(t) \leftarrow Init_{\Delta:\mathbf{y}}(n, M, [lb_{\mathbf{y}}, ub_{\mathbf{y}}])$ as Eq.4.2     ▷ initialize $\mathbf{y}$-relative step-size
 6: $\mathbf{x}_g(t) \leftarrow Init_{\Delta:\mathbf{x}}(\mathbf{X}(t))$ as Eq.4.3                           ▷ initialize $\mathbf{x}$-relative step-size
 7: $z(t) \leftarrow Init_{\Delta:z}(t, z^0, w)$ as Eq.4.4, Eq.4.5                             ▷ initialize $z$-relative step-size
 8: while stop condition $T$ do
 9:     $\mathbf{Y}(t+1) \leftarrow Opt_{\Delta:\mathbf{y}}(\mathbf{Y}(t), \mathbf{x}_g, w)$ as Eq.4.7             ▷ update $\mathbf{y}$-relative step-size
10:     $\hat{\mathbf{X}}(t+1) \leftarrow Opt_{\mathbf{x}}(\mathbf{X}(t), \mathbf{Y}(t), \mathbf{x}_g(t), z(t), w)$ as Eq.4.6  ▷ generate temporarily
    updated population
11:     $\hat{\mathbf{X}}(t+1) \leftarrow C(\hat{\mathbf{X}}(t+1))$ as Eq.4.11                      ▷ treatment to outliers
12:     $F(t+1) \leftarrow f(\hat{\mathbf{X}}(t+1))$                                                 ▷ evaluate
13:     $\mathbf{X}(t+1) \leftarrow S(\mathbf{X}(t), \hat{\mathbf{X}}(t+1), z(t))$ as Eq.4.12       ▷ select and generate finally
    updated population
14:     $\mathbf{x}_g(t+1) \leftarrow Init_{\Delta:\mathbf{x}}(\mathbf{X}(t), \mathbf{X}(t+1))$ as Eq.4.8     ▷ update $\mathbf{x}$-relative step-size
15:     $z(t+1) \leftarrow Opt_{\Delta:z}(t+1, z(t), w)$ as Eq.4.9, Eq.4.10 ▷ update $z$-relative step-size
16:     $t \leftarrow t+1$
17: end while
```

4.2.2 Re-framed GOA

The *Tuple.1* f can be represented as $F(t) = f(\mathbf{X}(t))$. The *Tuple.2* $Init_{\mathbf{x}}$ can be extracted from Eq.2.7, and re-framed as Eq.4.13.

$$\mathbf{x}_i(t=0) = \boldsymbol{\mathcal{U}}(lb_{\mathbf{x}}, ub_{\mathbf{x}}), i = 1, 2, \ldots, M \tag{4.13}$$

For the *Tuple.3* $Init_{\Delta}$, as discussed in Table 2.1, we find the global best individual $\mathbf{T}$ is a kind of $\mathbf{x}$-relative step-size. Meanwhile, the hyper-parameter c is a kind of z-relative step-size. Therefore, we re-frame $\mathbf{T}$ (see step (3)) into Eq.4.14. The z-relative step-size c can be re-framed as Eq.4.15.

$$\mathbf{x}_g(t=0) = \mathbf{Min}(\{\mathbf{x}_i(t=0)\}), i = 1 \ldots M \tag{4.14}$$

$$z(t=0) = ub_z - t \times (\frac{ub_z - lb_z}{T}) \tag{4.15}$$

For the *Tuple.4* $Opt_{\mathbf{x}}$, we find Eq.2.9, Eq.2.10 and Eq.2.11 are able to be re-framed into one equation as Eq.4.16.

$$\begin{aligned}&\widetilde{D}_{i,j}(t) = 2 + \textbf{Dist}(\mathbf{x}_i(t), \mathbf{x}_j(t)) \,\textbf{mod}\, 2\\ &\hat{\mathbf{x}}_i(t+1) = z(t) \times (\sum_{j=1, j\neq i}^{M} z(t) \times \frac{ub_{\mathbf{x}} - lb_{\mathbf{x}}}{2} \times (w_1 \times e^{\frac{-\widetilde{D}_{i,j}(t)}{w_2}} - e^{-\widetilde{D}_{i,j}(t)}) \times \frac{\mathbf{x}_i(t) - \mathbf{x}_j(t)}{\textbf{Dist}(\mathbf{x}_i(t), \mathbf{x}_j(t))}) + \mathbf{x}_g\end{aligned} \quad 4.16$$

For the *Tuple.5* Opt_{Δ} that corresponds to *Tuple.3* $Init_{\Delta}$, we find the update way of $\mathbf{T}$ (see step (9)) is able to be re-framed into as Eq.4.17. The update way of z (see Eq.2.8) can be re-framed as Eq.4.18.

$$\mathbf{x}_g(t+1) = \mathbf{Min}(\mathbf{x}_g(t) \cup \{\mathbf{x}_i(t+1)\}), i = 1 \ldots M \quad 4.17$$

$$z(t+1) = ub_z - (t+1) \times (\frac{ub_z - lb_z}{T}) \quad 4.18$$

For the *Tuple.6* C, the step (7) can be re-framed as Eq.4.19.

$$\mathbf{x}_{i,n}^{\text{fixed}}(t+1) = \begin{cases} ub_x & , \mathbf{x}_{i,n}(t+1) > ub_{\mathbf{x}} \\ \mathbf{x}_{i,n}(t+1) & , \text{o.w} \\ lb_x & , \mathbf{x}_{i,n}(t+1) < lb_{\mathbf{x}} \end{cases} \quad 4.19$$

For the *Tuple.7* S, the GOA model doesn't have a process to select which individuals will be delivered into the next generation, therefore, we re-frame it as Eq.4.20 that means all newly generated individuals will be delivered into the next generation.

$$\mathbf{x}_i(t+1) = \hat{\mathbf{x}}_i(t+1) \quad 4.20$$

For the *Tuple.8* T which is a stop condition related to the maximum number of iteration.

Till now, all of other unknown parameters can be defined as w-relative step-size, and they belong to the *Tuple.3* $Init_{\Delta}$. It has common w-relative step-size $[ub_w, lb_w] = [0.00004, 1]$,$w_1 = 0.5$, and $w_2 = 1.5$.

Therefore, the complete pseudo-code of re-framed BA model can be organized as Algorithm.3.

Algorithm 3 Re-framed GOA

```
1: $t \leftarrow 0$
2: $\mathbf{X}(t) \leftarrow Init_{\mathbf{x}}(n, M, [lb_{\mathbf{x}}, ub_{\mathbf{x}}])$ as Eq.4.13    ▷ initialize initial population
3: $F(t) \leftarrow f(\mathbf{X}(t))$    ▷ evaluate
4: $w \leftarrow Init_{\Delta:w}(w)$    ▷ initialize $w$-relative step-size
5: $\mathbf{x}_g(t) \leftarrow Init_{\Delta:\mathbf{x}}(\mathbf{X}(t))$ as Eq.4.14    ▷ initialize $\mathbf{x}$-relative step-size
6: $z(t) \leftarrow Init_{\Delta:z}(z^0)$ as Eq.4.15    ▷ initialize $z$-relative step-size
7: while stop condition $T$ do
8:     $\hat{\mathbf{X}}(t+1) \leftarrow Opt_{\mathbf{x}}(\mathbf{X}(t), \mathbf{x}_g(t), z(t), w)$ as Eq.4.16    ▷ generate temporarily updated population
9:     $\hat{\mathbf{X}}(t+1) \leftarrow C(\hat{\mathbf{X}}(t+1))$ as Eq.4.19    ▷ treatment to outliers
10:    $F(t+1) \leftarrow f(\hat{\mathbf{X}}(t+1))$    ▷ evaluate
11:    $\mathbf{X}(t+1) \leftarrow S(\mathbf{X}(t), \hat{\mathbf{X}}(t+1))$ as Eq.4.20 ▷ select and generate finally updated population
12:    $\mathbf{x}_g(t+1) \leftarrow Init_{\Delta:\mathbf{x}}(\mathbf{X}(t), \mathbf{X}(t+1))$ as Eq.4.17    ▷ update $\mathbf{x}$-relative step-size
13:    $z(t+1) \leftarrow Opt_{\Delta:z}(z(t), t+1)$ as Eq.4.18    ▷ update $z$-relative step-size
14:    $t \leftarrow t+1$
15: end while
```

4.2.3 Re-framed CSA

The *Tuple.1* f can be represented as $F(t) = f(\mathbf{X}(t))$. The *Tuple.2* $Init_{\mathbf{x}}$ can be extracted from Eq.2.12, and re-framed as Eq.4.21.

$$\mathbf{x}_i(t=0) = \boldsymbol{\mathcal{U}}(lb_{\mathbf{x}}, ub_{\mathbf{x}}), i = 1, 2, \ldots, M \qquad 4.21$$

For the *Tuple.3* $Init_{\Delta}$, as discussed in Table 2.1, we find the personal best $\mathbf{m}_i$ is a kind of $\mathbf{x}$-relative step-size. Therefore, we re-frame $\mathbf{m}_i$ (see step (3)) into Eq.4.22.

$$\mathbf{x}_{i_p}(t=0) = \mathbf{x}_i(t=0), i = 1 \ldots M \qquad 4.22$$

For the *Tuple.4* $Opt_{\mathbf{x}}$, we find Eq.2.14 is able to be re-framed as Eq.4.23 in which $\mathbf{x}_{j_p}$ is any one neighbor around the $\mathbf{x}_{i_p}$.

$$\hat{\mathbf{x}}_i(t+1) = \begin{cases} \mathbf{x}_i(t) + rand \times w_2 \times (\mathbf{x}_{j_p}(t) - \mathbf{x}_i(t)) & , \quad r > w_1 \\ \boldsymbol{\mathcal{U}}(lb_{\mathbf{x}}, ub_{\mathbf{x}}) & , \quad \text{o.w} \end{cases} \qquad 4.23$$

For the *Tuple.5* Opt_{Δ} that corresponds to *Tuple.3* $Init_{\Delta}$, we find the update way of $\mathbf{m}_i$ (see Eq.2.15) is able to be re-framed into as Eq.4.24.

$$\mathbf{x}_{i_p}(t+1) = \mathbf{Min}(\{\mathbf{x}_{i_p}(t), \mathbf{x}_i(t+1)\}) \qquad 4.24$$

For the *Tuple.6* C, the step (6) can be re-framed as Eq.4.25.

$$\mathbf{x}_{i,n}^{\text{fixed}}(t+1) = \begin{cases} \mathbf{x}_{i,n}(t) & , \mathbf{x}_{i,n}(t+1) < lb_{\mathbf{x}} \text{ or } \mathbf{x}_{i,n}(t+1) > ub_{\mathbf{x}} \\ \mathbf{x}_{i,n}(t+1) & , \text{o.w} \end{cases} \tag{4.25}$$

For the *Tuple.7* S, the CSA model doesn't have a process to select which individuals will be delivered into the next generation, therefore, we re-frame it as Eq.4.26 that means all newly generated individuals will be delivered into the next generation.

$$\mathbf{x}_i(t+1) = \hat{\mathbf{x}}_i(t+1) \tag{4.26}$$

For the *Tuple.8* T which is a stop condition related to the maximum number of iteration.

Till now, all of other unknown parameters can be defined as w-relative step-size, and they belong to the *Tuple.3* $Init_{\Delta}$. It has common w-relative step-size $w_1 = 0.1$ and $w_2 = 2$.

Therefore, the complete pseudo-code of re-framed BA model can be organized as Algorithm.3.

Algorithm 4 Re-framed CSA

```
1: $t \leftarrow 0$
2: $\mathbf{X}(t) \leftarrow Init_{\mathbf{x}}(n, M, [lb_{\mathbf{x}}, ub_{\mathbf{x}}])$ as Eq.4.21    ▷ initialize initial population
3: $F(t) \leftarrow f(\mathbf{X}(t))$    ▷ evaluate
4: $w \leftarrow Init_{\Delta:w}(w)$    ▷ initialize $w$-relative step-size
5: $\mathbf{X}_p(t) \leftarrow Init_{\Delta:\mathbf{x}}(\mathbf{X}(t))$ as Eq.4.22    ▷ initialize $\mathbf{x}$-relative step-size
6: while stop condition $T$ do
7:     $\hat{\mathbf{X}}(t+1) \leftarrow Opt_{\mathbf{x}}(\mathbf{X}(t), \mathbf{X}_p(t), w)$ as Eq.4.23    ▷ generate temporarily updated population
8:     $\hat{\mathbf{X}}(t+1) \leftarrow C(\hat{\mathbf{X}}(t+1))$ as Eq.4.25    ▷ treatment to outliers
9:     $F(t+1) \leftarrow f(\hat{\mathbf{X}}(t+1))$    ▷ evaluate
10:    $\mathbf{X}(t+1) \leftarrow S(\mathbf{X}(t), \hat{\mathbf{X}}(t+1))$ as Eq.4.26    ▷ select and generate finally updated population
11:    $\mathbf{X}_p(t+1) \leftarrow Init_{\Delta:\mathbf{x}}(\mathbf{X}(t), \mathbf{X}(t+1))$ as Eq.4.24    ▷ update $\mathbf{x}$-relative step-size
12:    $t \leftarrow t+1$
13: end while
```

4.2.4 Re-framed MFO

The *Tuple.1* f can be represented as $F(t) = f(\mathbf{X}(t))$. The *Tuple.2* $Init_{\mathbf{x}}$ can be extracted from Eq.2.16, and re-framed as Eq.4.27.

$$\mathbf{x}_i(t=0) = \boldsymbol{\mathcal{U}}(lb_{\mathbf{x}}, ub_{\mathbf{x}}), i = 1, 2, \ldots, M \tag{4.27}$$

For the *Tuple.3* $Init_{\Delta}$, as discussed in Table 2.1, we find the $sort\_population$ is a kind of $\mathbf{x}$-relative step-size. Meanwhile, the $flame\_no$ and the linearly decreasing parameter a with another parameter t are a kind of z-relative step-size. Therefore, we re-frame $sort\_population$ (see step (4)) into Eq.4.28. The two z-relative step-size $flame\_no$ (see Eq.2.17) and a with t (see Eq.2.18, step (5)) can be respectively re-framed as Eq.4.28 and Eq.4.29.

$$\langle \mathbf{x}_i(t=0)\rangle = \mathbf{Sort}(\{\mathbf{x}_i(t=0)\}), i=1\dots M \tag{4.28}$$

$$z_2(t=0) = \mathbf{Round}(M - t \times \frac{M-1}{T}) \tag{4.29}$$

$$z_{1_i}(t=0) = rand \times (-2 - \frac{t}{T}) + 1 \tag{4.30}$$

For the *Tuple.4* $Opt_{\mathbf{x}}$, we find Eq.2.19 and Eq.2.20 are able to be re-framed as Eq.4.31.

$$\hat{\mathbf{x}}_i(t+1) = \begin{cases} (\mathbf{x}_{s_i}(t) - \mathbf{x}_i(t)) \times e^{w \times z_{1_i}(t)} \times \cos(2\pi \times z_{1_i}(t)) + \mathbf{x}_{s_i}(t), & i \le z_2(t) \\ (\mathbf{x}_{s_{z_2(t)}}(t) - \mathbf{x}_i(t)) \times e^{w \times z_{1_i}(t)} \times \cos(2\pi \times z_{1_i}(t)) + \mathbf{x}_{s_{z_2(t)}}(t), & \text{o.w.} \end{cases} \tag{4.31}$$

For the *Tuple.5* Opt_{Δ} that corresponds to *Tuple.3* $Init_{\Delta}$, we find the update way of $sort\_population$ into Eq.4.32. The two z-relative step-size $flame\_no$ and a with t can be respectively re-framed as Eq.4.33 and Eq.4.34.

$$\langle \mathbf{x}_i(t+1)\rangle = \mathbf{Sort}(\{\mathbf{x}_i(t)\} \cup \{\mathbf{x}_i(t+1)\}), i=1\dots M \tag{4.32}$$

$$z_2(t+1) = \mathbf{Round}(M - (t+1) \times \frac{M-1}{T}) \tag{4.33}$$

$$z_{1_i}(t+1) = rand \times (-2 - \frac{t+1}{T}) + 1 \tag{4.34}$$

For the *Tuple.6* C, the MFO model doesn't have a process to deal with outliers, but we think the method C is necessary. Therefore, we use the most common way here to deal with outliers and re-frame it as Eq.4.35.

$$\mathbf{x}_{i,n}^{\text{fixed}}(t+1) = \begin{cases} ub_x & , \mathbf{x}_{i,n}(t+1) > ub_{\mathbf{x}} \\ \mathbf{x}_{i,n}(t+1) & , \text{o.w} \\ lb_x & , \mathbf{x}_{i,n}(t+1) < lb_{\mathbf{x}} \end{cases} \tag{4.35}$$

For the *Tuple.7* S, the MFO model doesn't have a process to select which individuals will be delivered into the next generation, therefore, we re-frame it as Eq.4.36 that means all newly generated individuals will be delivered into the next generation.

$$\mathbf{x}_i(t+1) = \hat{\mathbf{x}}_i(t+1) \tag{4.36}$$

For the *Tuple.8* T which is a stop condition related to the maximum number of iteration.

Till now, all of other unknown parameters can be defined as w-relative step-size, and they belong to the *Tuple.3* $Init_{\Delta}$. It has common w-relative step-size $w = 1$.

Therefore, the complete pseudo-code of re-framed BA model can be organized as Algorithm.3.

Algorithm 5 Re-framed MFO with population size M; search space $n, [lb_{\mathbf{x}}, ub_{\mathbf{x}}]$; stop condition T; initialization method $Init_{\mathbf{x}}$, optimization method $Opt_{\mathbf{x}}$, treatment C of outliers, and selection S to objective solutions; initialization method $Init_{\Delta}$ and optimization method Opt_{Δ} to step-size Δ.

```
1:  $t \leftarrow 0$
2:  $\mathbf{X}(t) \leftarrow Init_{\mathbf{x}}(n, M, [lb_{\mathbf{x}}, ub_{\mathbf{x}}])$ as Eq.4.27    ▷ initialize initial population
3:  $F(t) \leftarrow f(\mathbf{X}(t))$    ▷ evaluate
4:  $w \leftarrow Init_{\Delta:w}(w)$    ▷ initialize $w$-relative step-size
5:  $\mathbf{X}_s(t) \leftarrow Init_{\Delta:\mathbf{x}_s}(\mathbf{X}(t))$ as Eq.4.28    ▷ initialize $\mathbf{x}$-relative step-size
6:  $z(t) \leftarrow Init_{\Delta:z}(t, T)$ as Eq.4.29, Eq.4.30    ▷ initialize $z$-relative step-size
7:  while stop condition $T$ do
8:      $\hat{\mathbf{X}}(t+1) \leftarrow Opt_{\mathbf{x}}(\mathbf{X}(t), \mathbf{X}_s(t), z(t), w)$ as Eq.4.31    ▷ generate temporarily updated population
9:      $\hat{\mathbf{X}}(t+1) \leftarrow C(\hat{\mathbf{X}}(t+1))$ as Eq.4.35    ▷ treatment to outliers
10:     $F(t+1) \leftarrow f(\hat{\mathbf{X}}_i(t+1))$    ▷ evaluate
11:     $\mathbf{X}(t+1) \leftarrow S(\mathbf{X}_i(t), \hat{\mathbf{X}}_i(t+1))$ as Eq.4.36    ▷ select and generate finally updated population
12:     $\mathbf{X}_s(t+1) \leftarrow Init_{\Delta:\mathbf{x}_s}(\mathbf{X}(t), \mathbf{X}(t+1))$ as Eq.4.32    ▷ update dynamic $\mathbf{x}$-relative vector step-size $\mathbf{x}_s$
13:     $z_1(t+1) \leftarrow Opt_{\Delta:z_1}(t+1, T)$ as Eq.4.33, Eq.4.34    ▷ update $z$-relative step-size
14:     $t \leftarrow t+1$
15: end while
```

4.2.5 Re-framed MBO

The *Tuple.1* f can be represented as $F(t) = f(\mathbf{X}(t))$. The *Tuple.2* $Init_{\mathbf{x}}$ can be extracted from Eq.2.21, and re-framed as Eq.4.37.

$$\mathbf{x}_i(t=0) = \boldsymbol{\mathcal{U}}(lb_{\mathbf{x}}, ub_{\mathbf{x}}), i = 1, 2, \dots, M \tag{4.37}$$

For the *Tuple.3* $Init_{\Delta}$, as discussed in Table 2.1, we find the global best individual $\mathbf{x}_{best}$ is a kind of $\mathbf{x}$-relative step-size. Meanwhile, the weighting factor α is a kind of z-relative step-size. Therefore, we re-frame $\mathbf{v}_i$ (see step (2)) into Eq.4.2. The $\mathbf{x}_{best}$ (see step (8)) can be re-framed as Eq.4.38. The z-relative step-size α can be re-framed as Eq.4.39.

$$\mathbf{x}_g(t=0) = \mathbf{Min}(\{\mathbf{x}_i(t=0)\}), i = 1 \dots M \tag{4.38}$$

$$z(t=0) = \frac{w_4}{t^2} \tag{4.39}$$

For the *Tuple.4* $Opt_{\mathbf{x}}$, we find its update way (from step (3) to step (8)) is able to be re-framed into one equation as Eq.4.40, in which $\textbf{Lévy}_{i,n} = \textbf{Lévy}(d,n,T)$ with $d \sim Exp(2 \times T)$.

$$\begin{aligned}
\langle \mathbf{x}_i(t) \rangle &= \textsf{Sort}(\{\mathbf{x}_i(t)\}), i = 1 \dots M \\
{}^{strong}\hat{\mathbf{x}}_{i,n}(t+1) &= \begin{cases} \mathbf{x}_{j,n}(t) \in \langle \mathbf{x}_i(t) \rangle, j \in [1, M'] & , \; r \times w_2 \leqslant w_1 \\ \mathbf{x}_{j,n}(t) \in \langle \mathbf{x}_i(t) \rangle, j \in (M', M] & , \; \text{o.w.} \end{cases} \\
M' &= \lceil w1 \times M \rceil \\
{}^{weak}\hat{\mathbf{x}}_{i,n}(t+1) &= \begin{cases} \mathbf{x}_{g,n}(t), r \geqslant w_1 \\ \begin{cases} \mathbf{x}_{j,n}(t) + z(t) \times \left(\textbf{Lévy}_{i,n} - 0.5\right), j \in (M', M], r > w_3 \\ \mathbf{x}_{j,n}(t) \in \langle \mathbf{x}_i(t) \rangle, j \in (M', M], \text{o.w.} \end{cases} & , \text{o.w.} \end{cases} \\
\{\hat{\mathbf{x}}_i(t+1)\} &= \{{}^{strong}\hat{\mathbf{x}}_i(t+1)\} \cup \{{}^{weak}\hat{\mathbf{x}}_i(t+1)\}
\end{aligned} \tag{4.40}$$

For the *Tuple.5* Opt_Δ that corresponds to *Tuple.3* $Init_\Delta$, we find the update way of $\mathbf{x}_{best}$ (see step (8)) is able to be re-framed into as Eq.4.41. The update way of α (see Eq.2.23) can be re-framed as Eq.4.42.

$$\mathbf{x}_g(t+1) = \mathbf{Min}(\mathbf{x}_g(t) \cup \{\mathbf{x}_i(t+1)\}), i = 1 \dots M \tag{4.41}$$

$$z(t+1) = \frac{w_4}{(t+1)^2} \tag{4.42}$$

For the *Tuple.6* C, the step (9) can be re-framed as Eq.4.43.

$$\mathbf{x}_{i,n}^{\text{fixed}}(t+1) = \begin{cases} ub_x & , \; \mathbf{x}_{i,n}(t+1) > ub_{\mathbf{x}} \\ \mathbf{x}_{i,n}(t+1) & , \; \text{o.w} \\ lb_x & , \; \mathbf{x}_{i,n}(t+1) < lb_{\mathbf{x}} \end{cases} \tag{4.43}$$

For the *Tuple.7* S, the step (11) can be re-framed as Eq.4.44.

$$\begin{aligned}
&\langle \hat{\mathbf{x}}_i(t+1) \rangle = \textsf{Sort}(\{\hat{\mathbf{x}}_i(t+1)\}), i = 1 \dots M \\
&\mathbf{x}_i(t+1) \in \{\langle \hat{\mathbf{x}}_i(t+1) \rangle, i = 1 \dots M - w_5\} \cup \{\langle \mathbf{x}_i(t) \rangle, i = 1 \dots w_5\}
\end{aligned} \tag{4.44}$$

For the *Tuple.8* T which is a stop condition related to the maximum number of iteration.

Till now, all of other unknown parameters can be defined as w-relative step-size, and they belong to the *Tuple.3* $Init_\Delta$. It has common w-relative step-size $w_1 = \frac{5}{12}$, $w_2 = 1.2$, $w_3 = \frac{5}{12}$, $w_4 = 1$ and $w_5 = 2$

Therefore, the complete pseudo-code of re-framed BA model can be organized as Algorithm.6.

Algorithm 6 Re-framed MBO

```
 1: $t \leftarrow 0$
 2: $\mathbf{X}(t) \leftarrow Init_{\mathbf{x}}(n, M, [lb_{\mathbf{x}}, ub_{\mathbf{x}}])$ as Eq.4.37    ▷ initialize initial population
 3: $F(t) \leftarrow f(\mathbf{X}(t))$    ▷ evaluate
 4: $w \leftarrow Init_{\Delta:w}(w)$    ▷ initialize $w$-relative step-size
 5: $\mathbf{x}_g(t) \leftarrow Init_{\Delta:\mathbf{x}}(\mathbf{X}(t))$ as Eq.4.38    ▷ initialize $\mathbf{x}$-relative step-size
 6: $z \leftarrow Init_{\Delta:z}(w)$ as Eq.4.39    ▷ initialize $z$-relative step-size
 7: while stop condition $T$ do
 8:     $\hat{\mathbf{X}}(t+1) \leftarrow Opt_{\mathbf{x}}(\mathbf{X}(t), \mathbf{x}_g(t), z(t), w)$ as Eq.4.40    ▷ generate temporarily updated population
 9:     $\hat{\mathbf{X}}(t+1) \leftarrow C(\hat{\mathbf{X}}(t+1))$ as Eq.4.43    ▷ treatment to outliers
10:     $F(t+1) \leftarrow f(\hat{\mathbf{X}}(t+1))$    ▷ evaluate
11:     $\mathbf{X}(t+1) \leftarrow S(\mathbf{X}(t), \hat{\mathbf{X}}(t+1))$ as Eq.4.44 ▷ select and generate finally updated population
12:     $\mathbf{x}_g(t+1) \leftarrow Init_{\Delta:\mathbf{x}}(\mathbf{X}(t), \mathbf{X}(t+1))$ as Eq.4.41    ▷ update $\mathbf{x}$-relative step-size
13:     $z(t+1) \leftarrow Opt_{\Delta:z}(z(t), t+1)$ as Eq.4.42    ▷ update $z$-relative step-size
14:     $t \leftarrow t+1$
15: end while
```

4.2.6 Re-framed BOA

The *Tuple.1* f can be represented as $F(t) = f(\mathbf{X}(t))$. The *Tuple.2* $Init_{\mathbf{x}}$ can be extracted from Eq.2.26, and re-framed as Eq.4.45.

$$\mathbf{x}_i(t=0) = \boldsymbol{\mathcal{U}}(lb_{\mathbf{x}}, ub_{\mathbf{x}}), i = 1, 2, \ldots, M \tag{4.45}$$

For the *Tuple.3* $Init_\Delta$, as discussed in Table 2.1, we find the global best individual g_* is a kind of $\mathbf{x}$-relative step-size. Meanwhile, the $sensory\_modality$ is a kind of z-relative step-size. Therefore, we re-frame the g_* (see step (3)) can be re-framed as Eq.4.46. The z-relative step-size $sensory\_modality$ can be re-framed as Eq.4.47.

$$\mathbf{x}_g(t=0) = \mathbf{Min}(\{\mathbf{x}_i(t=0)\}), i = 1 \ldots M \tag{4.46}$$

$$z(t=0) = z^0 \tag{4.47}$$

For the *Tuple.4* $Opt_{\mathbf{x}}$, we find step (4), step (5) and step (6) are able to be re-framed into one equation as Eq.4.48, in which $\mathbf{x}_j$ and $\mathbf{x}_k$ are any two neighbors around $\mathbf{x}_i$.

$$\hat{\mathbf{x}}_i(t+1) = \begin{cases} \mathbf{x}_i(t) + (r^2 \times \mathbf{x}_g(t) - \mathbf{x}_i(t)) \times z_1(t) \times f(\mathbf{x}_i(t))^{w_1} & , \ r > w_2 \\ \mathbf{x}_i(t) + (r^2 \times \mathbf{x}_j(t) - \mathbf{x}_k(t)) \times z_1(t) \times f(\mathbf{x}_i(t))^{w_1} & , \ \text{o.w} \end{cases} \tag{4.48}$$

For the *Tuple.5* Opt_{Δ} that corresponds to *Tuple.3* $Init_{\Delta}$, we find the update way of g_* (see step (9)) is able to be re-framed into as Eq.4.49. The update way of $sensory\_modality$ (see Eq.2.30) can be re-framed as Eq.4.50.

$$\mathbf{x}_g(t+1) = \mathbf{Min}(\mathbf{x}_g(t) \cup \{\mathbf{x}_i(t+1)\}), i = 1 \dots M \tag{4.49}$$

$$z(t+1) = z(t) + \frac{0.025}{z(t) \times T} \tag{4.50}$$

For the *Tuple.6* C, the step (7) can be re-framed as Eq.4.51.

$$\mathbf{x}_{i,n}^{\text{fixed}}(t+1) = \begin{cases} ub_x & , \ \mathbf{x}_{i,n}(t+1) > ub_{\mathbf{x}} \\ \mathbf{x}_{i,n}(t+1) & , \ \text{o.w} \\ lb_x & , \ \mathbf{x}_{i,n}(t+1) < lb_{\mathbf{x}} \end{cases} \tag{4.51}$$

For the *Tuple.7* S, the step (8) can be re-framed as Eq.4.52.

$$\mathbf{x}_i(t+1) = \begin{cases} \hat{\mathbf{x}}_i(t+1) & , \ f(\hat{\mathbf{x}}_i(t+1)) < f(\mathbf{x}_i(t)) \\ \mathbf{x}_i(t) & , \ \text{o.w} \end{cases} \tag{4.52}$$

For the *Tuple.8* T which is a stop condition related to the maximum number of iteration.

Till now, all of other unknown parameters can be defined as w-relative step-size, and they belong to the *Tuple.3* $Init_{\Delta}$. It has common w-relative step-size $w_1 = 0.1$ and $w_2 = 0.8$. It also has special w-relative step-size that is related to the initial value of z-relative step-size $z^0 = 0.01$.

Therefore, the complete pseudo-code of re-framed BA model can be organized as Algorithm.7.

Algorithm 7 Re-framed BOA

```
1: $t \leftarrow 0$
2: $\mathbf{X}(t) \leftarrow Init_{\mathbf{x}}(n, M, [lb_{\mathbf{x}}, ub_{\mathbf{x}}])$ as Eq.4.45    ▷ initialize initial population
3: $F(t) \leftarrow f(\mathbf{X}(t))$    ▷ evaluate
4: $w, z^0 \leftarrow Init_{\Delta:w}(w, z^0)$    ▷ initialize $w$-relative step-size
5: $\mathbf{x}_g(t) \leftarrow Init_{\Delta:\mathbf{x}}(\mathbf{X}(t))$ as Eq.4.46    ▷ initialize $\mathbf{x}$-relative step-size
6: $z(t) \leftarrow Init_{\Delta:z}(z^0)$ as Eq.4.47    ▷ initialize $z$-relative step-size
7: while stop condition $T$ do
8:     $\hat{\mathbf{X}}(t+1) \leftarrow Opt_{\mathbf{x}}(\mathbf{X}(t), \mathbf{x}_g(t), z(t), w)$ as Eq.4.48    ▷ generate temporarily updated population
9:     $\hat{\mathbf{X}}(t+1) \leftarrow C(\hat{\mathbf{X}}(t+1))$ as Eq.4.51    ▷ treatment to outliers
10:    $F(t+1) \leftarrow f(\hat{\mathbf{X}}(t+1))$    ▷ evaluate
11:    $\mathbf{X}(t+1) \leftarrow S(\mathbf{X}(t), \hat{\mathbf{X}}(t+1))$ as Eq.4.52 ▷ select and generate finally updated population
12:    $\mathbf{x}_g(t+1) \leftarrow Init_{\Delta:\mathbf{x}}(\mathbf{X}(t), \mathbf{X}(t+1))$ as Eq.4.49    ▷ update $\mathbf{x}$-relative step-size
13:    $z(t+1) \leftarrow Opt_{\Delta:z}(z(t), t+1)$ as Eq.4.50    ▷ update $z$-relative step-size
14:    $t \leftarrow t+1$
15: end while
```

4.2.7 Re-framed PSO

The *Tuple.1* f can be represented as $F(t) = f(\mathbf{X}(t))$. The *Tuple.2* $Init_{\mathbf{x}}$ can be extracted from Eq.2.31, and re-framed as Eq.4.53.

$$\mathbf{x}_i(t=0) = \boldsymbol{\mathcal{U}}(lb_{\mathbf{x}}, ub_{\mathbf{x}}), i = 1, 2, \ldots, M \tag{4.53}$$

For the *Tuple.3* $Init_{\Delta}$, as discussed in Table 2.1, we find the velocity $\mathbf{v}_i$ is a kind of $\mathbf{y}$-relative step-size. The personal best individual $pbest$ and the global best individual $gbest$ are a kind of $\mathbf{x}$-relative step-size. Therefore, we re-frame $\mathbf{v}_i$ (see step (2)) into Eq.4.54. The $pbest$ (see Eq.2.33) and the $gbest$ (see step (5)) can be respectively re-framed as Eq.4.55 and Eq.4.56.

$$\mathbf{y}_i(t=0) = \boldsymbol{\mathcal{U}}(lb_{\mathbf{y}}, ub_{\mathbf{y}}), i = 1 \ldots M \tag{4.54}$$

$$\mathbf{x}_{i_p}(t=0) = \mathbf{x}_i(t=0), i = 1 \ldots M \tag{4.55}$$

$$\mathbf{x}_g(t=0) = \mathbf{Min}(\{\mathbf{x}_i(t=0)\}), i = 1 \ldots M \tag{4.56}$$

For the *Tuple.4* $Opt_{\mathbf{x}}$, we find Eq.2.35 is able to be re-framed as Eq.4.57.

$$\hat{\mathbf{x}}_i(t+1) = \mathbf{x}_i(t) + \mathbf{y}_i(t+1) \tag{4.57}$$

For the *Tuple.5* Opt_{Δ} that corresponds to *Tuple.3* $Init_{\Delta}$, we find the update way of $\mathbf{v}_i$ (see Eq.2.34) is able to be re-framed into as Eq.4.58. The update way of $pbest$ (see step (9)) is able to be re-framed into as Eq.4.59. The update way of $gbest$ (see Eq.2.3) can be re-framed as Eq.4.60.

$$\mathbf{y}_i(t+1) = w_1 \times \mathbf{y}_i(t) + \mathcal{U}(0,w_2) \times (\mathbf{x}_{i_p}(t) - \mathbf{x}_i(t)) + \mathcal{U}(0,w_3) \times (\mathbf{x}_g(t) - \mathbf{x}_i(t)) \tag{4.58}$$

$$\mathbf{x}_{i_p}(t+1) = \textsf{\textbf{Min}}(\{\mathbf{x}_{i_p}(t), \mathbf{x}_i(t+1)\}) \tag{4.59}$$

$$\mathbf{x}_g(t+1) = \mathbf{Min}(\mathbf{x}_g(t) \cup \{\mathbf{x}_i(t+1)\}), i = 1 \dots M \tag{4.60}$$

For the *Tuple.6* C, the step (7) can be re-framed as Eq.4.61.

$$\mathbf{x}_{i,n}^{\text{fixed}}(t+1) = \begin{cases} ub_x & , \quad \mathbf{x}_{i,n}(t+1) > ub_{\mathbf{x}} \\ \mathbf{x}_{i,n}(t+1) & , \quad \text{o.w} \\ lb_x & , \quad \mathbf{x}_{i,n}(t+1) < lb_{\mathbf{x}} \end{cases} \tag{4.61}$$

For the *Tuple.7* S, the PSO model doesn't have a process to select which individuals will be delivered into the next generation, therefore, we re-frame it as Eq.4.62 that means all newly generated individuals will be delivered into the next generation.

$$\mathbf{x}_i(t+1) = \hat{\mathbf{x}}_i(t+1) \tag{4.62}$$

For the *Tuple.8* T which is a stop condition related to the maximum number of iteration.

Till now, all of other unknown parameters can be defined as w-relative step-size, and they belong to the *Tuple.3* $Init_{\Delta}$. It has common w-relative step-size $w_1 = 0.73$, $w_2 = 1.49$, $w_2 = 1.49$.

Therefore, the complete pseudo-code of re-framed BA model can be organized as Algorithm.8.

Algorithm 8 Re-framed PSO

```
1:  $t \leftarrow 0$
2:  $\mathbf{X}(t) \leftarrow Init_{\mathbf{x}}(n, M, [lb_{\mathbf{x}}, ub_{\mathbf{x}}])$ as Eq.4.53    ▷ initialize initial population
3:  $F(t) \leftarrow f(\mathbf{X}(t))$    ▷ evaluate
4:  $w, [lb_{\mathbf{y}}, ub_{\mathbf{y}}] \leftarrow Init_{\Delta:w}(w, [lb_{\mathbf{y}}, ub_{\mathbf{y}}])$    ▷ initialize $w$-relative step-size
5:  $\mathbf{Y}(t) \leftarrow Init_{\Delta:\mathbf{y}}(n, M, [lb_{\mathbf{y}}, ub_{\mathbf{y}}])$ as Eq.4.54    ▷ initialize $\mathbf{y}$-relative step-size
6:  $\mathbf{X}_p(t), \mathbf{x}_g(t) \leftarrow Init_{\Delta:\mathbf{x}}(\mathbf{X}(t))$ as Eq.4.55, Eq.4.56    ▷ initialize $\mathbf{x}$-relative step-size
7:  while stop condition $T$ do
8:      $\mathbf{Y}(t+1) \leftarrow Opt_{\Delta:\mathbf{y}}(\mathbf{Y}(t)$ as eqeq:pso5    ▷ update $\mathbf{y}$-relative step-size
9:      $\hat{\mathbf{X}}(t+1) \leftarrow Opt_{\mathbf{x}}(\mathbf{X}(t), \mathbf{Y}(t), \mathbf{x}_g(t), \mathbf{X}_p(t), w)$ as Eq.4.57    ▷ generate
    temporarily updated population
10:     $\hat{\mathbf{X}}(t+1) \leftarrow C(\hat{\mathbf{X}}(t+1))$ as Eq.4.60    ▷ treatment to outliers
11:     $F(t+1) \leftarrow f(\hat{\mathbf{X}}(t+1))$    ▷ evaluate
12:     $\mathbf{X}(t+1) \leftarrow S(\mathbf{X}(t), \hat{\mathbf{X}}(t+1))$ as Eq.4.61 ▷ select and generate finally updated
    population
13:     $\mathbf{X}_p(t+1), \mathbf{x}_g(t+1) \leftarrow Init_{\Delta:\mathbf{x}}(\mathbf{X}(t), \mathbf{X}(t+1))$ as Eq.4.60, Eq.4.61  ▷ update
    $\mathbf{x}$-relative step-size
14:     $t \leftarrow t+1$
15: end while
```

4.3 Unified Nature-Inspired Optimization Algorithm (UNIOA)

4.3.1 Unified Framework — UNIOA

In this section, we introduce the Unified framework for Nature-inspired Optimization Algorithm — UNIOA that is the end goal of this work. As discussed in Chapter 2, Chapter 3 and Section 4.2, we can already be sure that these seven selected algorithms can be re-framed from their original different terminologies in different procedures to our unified terminologies in eight same tuples whose positions can also be same.

We summarize our observations in Algorithm.9. In the Initialization Process, these seven algorithms can be completely same. For example, these algorithms need an initial population with the fitness of this population and also need several step-size Δ to trigger the optimization mechanism. Different algorithms can have different step-size Δ. In the Optimization Process, there are two options — the algorithm uses $\mathbf{y}$-relative step-size Δ or the algorithm doesn't use $\mathbf{y}$-relative step-size Δ. If the algorithm uses the $\mathbf{y}$-relative step-size Δ, the $\mathbf{y}$-relative Δ shall update before updating the population $\mathbf{X}$. If the algorithm doesn't use the $\mathbf{y}$-relative step-size Δ, all dynamic Δ can update together after the selection operation. Meanwhile, z-relative Δ has flexible positions when modeling a valid algorithm. For example, z-relative Δ can be used in S selection method in BA.

Algorithm 9 Unified Nature-Inspired Optimization Algorithm — UNIOA

```
1:  $t \leftarrow 0$
2:  $\mathbf{X}(t) \leftarrow Init_{\mathbf{x}}(n, M, [lb_{\mathbf{x}}, ub_{\mathbf{x}}])$    ▷ initialize initial pop
3:  $F(t) \leftarrow f(\mathbf{X}(t))$    ▷ evaluate
4:  $\Delta_{w,z,\mathbf{y},\mathbf{x}}(t) \leftarrow Init_{\Delta:w,z,\mathbf{y},\mathbf{x}}(\mathbf{X}(t), w, z^0, [lb_{\mathbf{y}}, ub_{\mathbf{y}}], t)$    ▷ initialize step-size
5:  while $T$ do
6:      if $\mathbf{y} \in \Delta(t=0)$ then
7:          $\mathbf{Y}(t+1) \leftarrow Opt_{\Delta:\mathbf{y}}(\mathbf{Y}(t), w)$    ▷ update $\mathbf{y}$-relative step-size
8:          $\hat{\mathbf{X}}(t+1) \leftarrow Opt_{\mathbf{x}}(\mathbf{X}(t), \mathbf{Y}(t+1), \Delta_{w,z,\mathbf{x}}(t))$    ▷ temporarily updated pop
9:          $\hat{\mathbf{X}}(t+1) \leftarrow C(\hat{\mathbf{X}}(t+1))$    ▷ outliers treatment
10:         $F(t+1) \leftarrow f(\hat{\mathbf{X}}(t+1))$    ▷ evaluate
11:         $\mathbf{X}(t+1) \leftarrow S(\mathbf{X}(t), \hat{\mathbf{X}}(t+1), \Delta_{w,z}(t))$▷ select and generate finally updated pop
12:         $\Delta_{z,\mathbf{x}}(t+1) \leftarrow Opt_{\Delta:z,\mathbf{x}}(\mathbf{X}(t), \mathbf{X}(t+1), z(t), t+1)$    ▷ update $z, \mathbf{x}$-relative step-size
13:     else
14:         $\hat{\mathbf{X}}(t+1) \leftarrow Opt_{\mathbf{x}}(\mathbf{X}(t), \Delta(t))$    ▷ temporarily updated pop
15:         $\hat{\mathbf{X}}(t+1) \leftarrow C(\hat{\mathbf{X}}(t+1))$    ▷ outliers treatment
16:         $F(t+1) \leftarrow f(\hat{\mathbf{X}}(t+1))$    ▷ evaluate
17:         $\mathbf{X}(t+1) \leftarrow S(\mathbf{X}(t), \hat{\mathbf{X}}(t+1), \Delta_{w,z}(t))$▷ select and generate finally updated pop
18:         $\Delta(t+1) \leftarrow Opt_{\Delta}(\mathbf{X}(t), \mathbf{X}(t+1), \Delta(t), t+1)$    ▷ update step-size
19:     end if
20:     $t \leftarrow t+1$
21: end while
```

We recall the 8-tuple function (see Eq.3.1) defined in Chapter 3, and extend it as Eq.4.63 with the new name — $UNIOA$.

$$UNIOA = (f, Init_{\mathbf{x}}, Opt_{\mathbf{x}}, C, T, S, Init_{\Delta}, Opt_{\Delta}) \qquad 4.63$$

Furthermore, according to observations in Section 4.2, these eight tuples have following different options:

(1) *Tuple.1* f:
The *Tuple.1* f exists in all these seven algorithms. Its function in these algorithms is also same as $f(\mathbf{X})$ that calculates the fitness of the whole population X.

(2) *Tuple.2* $Init_{\mathbf{x}}$:
The *Tuple.2* $Init_{\mathbf{x}}$ exists in all these seven algorithms. Its function is also same as $Init_{\mathbf{x}}$.

$$\mathbf{x}_i = \boldsymbol{\mathcal{U}}(lb_{\mathbf{x}}, ub_{\mathbf{x}}), i = 1, 2, \ldots, M \qquad Init_{\mathbf{x}}$$

(3) *Tuple.3* $Init_{\Delta}$:
The *Tuple.3* $Init_{\Delta}$ exists in all these seven algorithms. But not all kinds of step-size Δ are present in all algorithms at the same time. There are four kinds of step-size Δ: w-relative Δ, z-relative Δ, $\mathbf{x}$-relative Δ and $\mathbf{y}$-relative Δ.

(a) w-relative Δ:
This kind of step-size doesn't change during the optimization process. There are three kinds of w-relative Δ: $w, z^0, [lb_{\mathbf{y}}, ub_{\mathbf{y}}]$. The w is commonly used in most algorithms and its common name is hyper-parameter. The z^0 and $[lb_{\mathbf{y}}, ub_{\mathbf{y}}]$ are respectively initial values of z-relative Δ and $\mathbf{y}$-relative Δ.

(b) z-relative Δ:
The z is a kind of hyper-parameter that will be changing over the iteration t during the optimization process. Its initialization method could be varying (see Index.4 in Table 4.1).

(c) $\mathbf{x}$-relative Δ:

- $\mathbf{x}_{i_p}$ is the best $\mathbf{x}_i$ that each $\mathbf{x}_i$ has found so far. Its initialization method is fixed as $Init_{\Delta:\mathbf{x}_{i_p}}$.

$$\mathbf{x}_{i_p}(t) = \mathbf{x}_i(t), i = 1 \ldots M \qquad Init_{\Delta:\mathbf{x}_{i_p}}$$

- $\mathbf{x}_g$ is the best $\mathbf{x}_i$ that the whole population has found so far. Its initialization method is fixed as $Init_{\Delta:\mathbf{x}_g}$.

$$\mathbf{x}_g(t) = \mathbf{Min}(\{\mathbf{x}_i(t)\}), i = 1 \ldots M \qquad Init_{\Delta:\mathbf{x}_g}$$

- $\mathbf{x}_s$ denotes one kind of special $\mathbf{x}$-relative Δ that has a strong connection with the $\mathbf{x}_i$. It depends on different algorithms, so it is varying. For example, Eq.4.28 in MFO.

(d) $\mathbf{y}$-relative Δ:
In this work, there is only one kind of initialization method as $Init_{\Delta:\mathbf{y}}$. In this work, its initialization method is fixed, however, it could be customized in the future.

$$\mathbf{y}_i(t) = \boldsymbol{\mathcal{U}}(lb_{\mathbf{y}}, ub_{\mathbf{y}}), i = 1 \dots M \qquad Init_{\Delta:\mathbf{y}}$$

(4) *Tuple.4* $Opt_{\mathbf{x}}$:
The *Tuple.4* $Opt_{\mathbf{x}}$ exists in all these seven algorithms. Its method is varying (see Index.7 in Table 4.1).

(5) *Tuple.5* Opt_{Δ}:
The *Tuple.5* Opt_{Δ} that corresponds to *Tuple.3* $Init_{\Delta}$ exists in all these seven algorithms. *Tuple.5* Opt_{Δ} can locate at two positions: Index.6 or Index.11 in Table 4.1.

(a) $\mathbf{y}$-relative Δ at Index.6 position:
The update method to $\mathbf{y}$-relative Δ is varying in different algorithms (see Index.6 in Table 4.1)

(b) z-relative and $\mathbf{x}$-relative Δ at Index.11 position:
The update method of z-relative Δ is varying in different algorithms (see Index.11 in Table 4.1). However, the update method of two $\mathbf{x}$-relative Δ is fixed as $Opt_{\Delta:\mathbf{x}_{ip}}$ and $Opt_{\Delta:\mathbf{x}_g}$. The update method of $\mathbf{x}$-relative Δ can also be customized, for example, the $Opt_{\Delta:\mathbf{x}_s}$: Eq.4.28 in MFO.

$$\mathbf{x}_{i_p}(t+1) = \mathbf{Min}(\{\mathbf{x}_{i_p}(t), \mathbf{x}_i(t+1)\}) \qquad Opt_{\Delta:\mathbf{x}_{i_p}}$$

$$\mathbf{x}_g(t+1) = \mathbf{Min}(\mathbf{x}_g(t) \cup \{\mathbf{x}_i(t+1)\}), i = 1 \dots M \qquad Opt_{\Delta:\mathbf{x}_g}$$

(6) *Tuple.6* C:
The *Tuple.6* C exists in all algorithms. There are two methods to deal with outliers in this work. For example, $C1$ is applied in BA, GOA, MFO, MBO, BOA and PSO, $C2$ is only applied in CSA.[2]

$$\mathbf{x}_{i,n}^{\text{fixed}}(t+1) = \begin{cases} ub_x & , \ \mathbf{x}_{i,n}(t+1) > ub_{\mathbf{x}} \\ \mathbf{x}_{i,n}(t+1) & , \ \text{o.w} \\ lb_x & , \ \mathbf{x}_{i,n}(t+1) < lb_{\mathbf{x}} \end{cases} \qquad \text{C1}$$

[2]Some algorithms didn't mention C in their published paper or code, however, it is acceptable that outliers are invalid and must be dealt with.

$$\mathbf{x}_{i,n}^{\text{fixed}}(t+1) = \begin{cases} \mathbf{x}_{i,n}(t) & , \mathbf{x}_{i,n}(t+1) < lb_{\mathbf{x}} \text{ or } \mathbf{x}_{i,n}(t+1) > ub_{\mathbf{x}} \\ \mathbf{x}_{i,n}(t+1) & , \text{o.w} \end{cases} \quad \text{C2}$$

(7) *Tuple.7* S.
The *Tuple.7* S exists in all algorithms. There are four selection methods in this work. For example, $S1$ is applied in GOA, CSA, MFO and PSO, $S2$ is applied in BOA, $S3$ is applied in BA, $S4$ is applied in MBO.

$$\mathbf{x}_i(t+1) = \hat{\mathbf{x}}_i(t+1) \quad \text{S1}$$

$$\mathbf{x}_i(t+1) = \begin{cases} \hat{\mathbf{x}}_i(t+1) & , f(\hat{\mathbf{x}}_i(t+1)) < f(\mathbf{x}_i(t)) \\ \mathbf{x}_i(t) & , \text{o.w} \end{cases} \quad \text{S2}$$

$$\mathbf{x}_i(t+1) = \begin{cases} \hat{\mathbf{x}}_i(t+1) & , f(\hat{\mathbf{x}}_i(t+1)) < f(\mathbf{x}_i(t)) \text{ or } rand > z_2(t) \\ \mathbf{x}_i(t) & , \text{o.w} \end{cases} \quad \text{S3}$$

$$\begin{aligned} &\langle \hat{\mathbf{x}}_i(t+1) \rangle = \mathbf{Sort}(\{\hat{\mathbf{x}}_i(t+1)\}), i = 1 \ldots M \\ &\mathbf{x}_i(t+1) \in \{\langle \hat{\mathbf{x}}_i(t+1) \rangle, i = 1 \ldots M - w_5\} \cup \{\langle \mathbf{x}_i(t) \rangle, i = 1 \ldots w_5\} \end{aligned} \quad \text{S4}$$

(8) *Tuple.8* T.
The T determines if the iterative optimization process would stop or not. It must exist.

We also summarize these eight tuples in Table 4.1, in which each algorithm has its own representations of each tuple, but these seven algorithms can also have same representations in some tuples.

4.3.2 UNIOA Package

The observations in Subsection 4.3.1 are visualized in Figure 4.1, in which seven selected algorithms can be re-written in one same framework. In this unfied framework — UNIOA, tuples for making up algorithms are fixed, and the positions of these tuples are also fixed. We also find these algorithms perform differently mainly because of the different designs of these tuples.

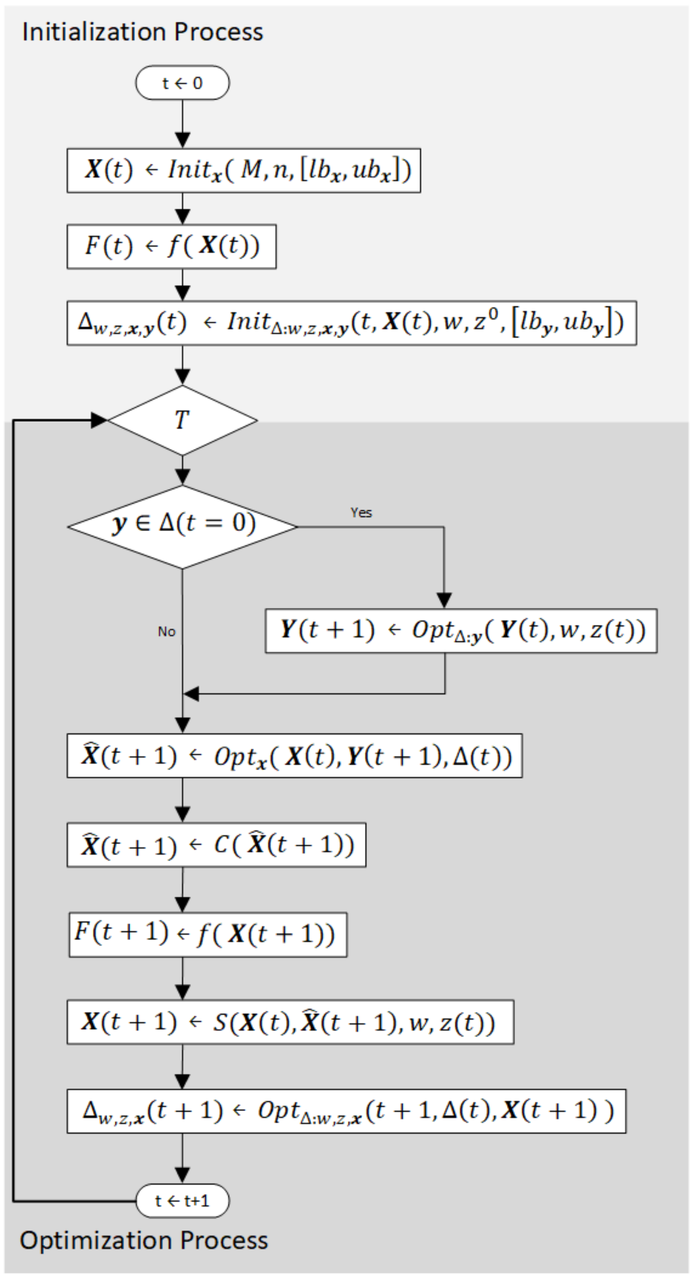
Initialization Process
t ← 0
$\boldsymbol{X}(t) \leftarrow Init_{\boldsymbol{x}}(M, n, [lb_{\boldsymbol{x}}, ub_{\boldsymbol{x}}])$
$F(t) \leftarrow f(\boldsymbol{X}(t))$
$\Delta_{w,z,\boldsymbol{x},\boldsymbol{y}}(t) \leftarrow Init_{\Delta:w,z,\boldsymbol{x},\boldsymbol{y}}(t, \boldsymbol{X}(t), w, z^0, [lb_{\boldsymbol{y}}, ub_{\boldsymbol{y}}])$
T
$\boldsymbol{y} \in \Delta(t=0)$
Yes
No
$\boldsymbol{Y}(t+1) \leftarrow Opt_{\Delta:\boldsymbol{y}}(\boldsymbol{Y}(t), w, z(t))$
$\widehat{\boldsymbol{X}}(t+1) \leftarrow Opt_{\boldsymbol{x}}(\boldsymbol{X}(t), \boldsymbol{Y}(t+1), \Delta(t))$
$\widehat{\boldsymbol{X}}(t+1) \leftarrow C(\widehat{\boldsymbol{X}}(t+1))$
$F(t+1) \leftarrow f(\boldsymbol{X}(t+1))$
$\boldsymbol{X}(t+1) \leftarrow S(\boldsymbol{X}(t), \widehat{\boldsymbol{X}}(t+1), w, z(t))$
$\Delta_{w,z,\boldsymbol{x}}(t+1) \leftarrow Opt_{\Delta:w,z,\boldsymbol{x}}(t+1, \Delta(t), \boldsymbol{X}(t+1))$
t ← t+1
Optimization Process

Figure 4.1: Unified framework for Nature-Inspired Optimization Algorithm —UNIOA, only for seven selected algorithms

Therefore, as shown in Table 4.2, we discuss the possibility of each tuple being customized in the future. In a complete optimization algorithm, *Tuple.1* f, *Tuple.2* $Init_{\mathbf{x}}$, *Tuple.3* $Init_{\Delta:\mathbf{x}_{ip},\mathbf{x}_g}$ and *Tuple.5* $Init_{\Delta:\mathbf{x}_{ip},\mathbf{x}_g}$ are not allowed to be customized. *Tuple.1* f and *Tuple.2* $Init_{\mathbf{x}}$ are coming from the objective optimization problem. The way of initializing and updating two special $\mathbf{x}$-relative step-size Δ (*Tuple.3* $Init_{\Delta:\mathbf{x}_{ip},\mathbf{x}_g}$ and *Tuple.5* $Init_{\Delta:\mathbf{x}_{ip},\mathbf{x}_g}$) is also fixed in general.[3] For other tuples (*Tuple.3* $Init_{\Delta:z,\mathbf{y},\mathbf{x}_s}$, *Tuple.4* $Opt_{\mathbf{x}}$, *Tuple.5* $Init_{\Delta:z,\mathbf{y},\mathbf{x}_s}$, *Tuple.6* C and *Tuple.7* S), they can be freely designed by users.

The general utilization of this observation is to design an auto-designer for algorithms. For example, there are many open Python libraries [4, 6, 9, 13, 22, 27] for developing evolutionary and genetic algorithms (EA). Various 'crossover,' 'mutation,' and 'selection' are freely combined together to build up new algorithms. However, such application might not be suitable in $UNIOA$ for swarm-based algorithms (SI).

The most significant difference between the unified framework for EA and the unified framework for SI is whether customers can freely combine different options of different tuples to build up a new algorithm. In the EA group, it is possible to freely combine different options from different tuples because every tuple operation happens to the entire population. For example, in the most popular unified framework for EA, the entire population is fed into each tuple one by one. The output of each tuple is still the entire population. However, in the SI group, this design idea is impossible. For example, in the Table 4.2, the *Tuple.3* $Init_{\Delta}$ has six options, and the *Tuple.4* $Opt_{\mathbf{x}}$ has seven options. However, it doesn't mean any option of *Tuple.3* $Init_{\Delta}$ can be combined with any option of *Tuple.4* $Opt_{\mathbf{x}}$ to make up a new algorithm. The reason is *Tuple.3* $Init_{\Delta}$ and *Tuple.4* $Opt_{\mathbf{x}}$ are one-to-one correspondence.[4]

Therefore, although in the EA framework, customers are allowed to build up a new algorithm by freely selecting different options from existing methods, customers don't have much freedom in the UNIOA framework. However, the UNIOA gives customers many possibilities to design a totally new algorithm by only considering mathematical equations. For example, users shall start designing *Tuple.4* $Opt_{\mathbf{x}}$ that determines how many and what kinds of step-size Δ will appear in this algorithm. *Tuple.6* C and *Tuple.7* S are then customized to help achieve the *Tuple.4* $Opt_{\mathbf{x}}$.

[3]Step-size $\Delta : \mathbf{x}_{ip}, \mathbf{x}_g$ might also be customized when people have different understanding about what is the personal best solution and what is the global best solution.

[4]This difference can also be understood as: SI has more complicated optimization mechanisms compared to EA. For example, gene operations in EA are limited, but social activities for designing SI in the real world are very various.

The idea of auto-designing SI in UNIOA is basically achieved in UNIOA [5] in which users can design their iterative optimization algorithms only with primitive math knowledge and without any biological knowledge. Moreover, in UNIOA, the customized algorithm can measure its performance or compare its performance with the other seven existing algorithms (studied in this work) with the help of the IOHprofiler environment.

[5]https://github.com/Huilin-Li/UNIOA

Table 4.1: Representations of eight tuples in each algorithm.

Progress	Index	Tuple		BA	GOA	CSA	MFO	MBO	BOA	PSO
Initialization	1	t=0								
	2	*Tuple.2* $Init_{\mathbf{x}}$		$Init_{\mathbf{x}}$						
	3	*Tuple.1* f		$f(\mathbf{x}(t))$						
	4	*Tuple.3* $Init_{\Delta}$	$\mathbf{y}_i$	$Init_{\Delta:\mathbf{y}}$	×	×	×	×	×	$Init_{\Delta:\mathbf{y}}$
			$\mathbf{x}_{i_p}$	×	×	$Init_{\Delta:\mathbf{x}_{i_p}}$	×	×	×	$Init_{\Delta:\mathbf{x}_{i_p}}$
			$\mathbf{x}_g$	$Init_{\Delta:\mathbf{x}_g}$		×	×	$Init_{\Delta:\mathbf{x}_g}$		
			$\mathbf{x}_s$	×	×	×	Eq.4.28	×	×	×
			z	Eq.4.3, Eq.4.4	Eq.4.14	×	Eq.4.29, Eq.4.29	Eq.4.38	Eq.4.46	×
			w	✓	✓	✓	✓	✓	✓	✓
Optimization	5	*Tuple.8* Stop condition T								
	6	*Tuple.5*$Opt_{\Delta:\mathbf{y}}$	$\mathbf{y}_i(t+1)$	Eq.4.5	×	×	×	×	×	Eq.4.54
	7	*Tuple.4* $Opt_{\mathbf{x}}$		Eq.4.6	Eq.4.15	Eq.4.22	Eq.4.31	Eq.4.39	Eq.4.47	Eq.4.55
	8	*Tuple.6* C		$C1$		$C2$	$C1$			
	9	*Tuple.1* f		$f(\hat{\mathbf{x}}_i(t+1))$						
	10	*Tuple.7* S		$S3$	$S1$			$S4$	$S2$	$S1$
	11	*Tuple.5* $Opt_{\Delta:z,\mathbf{x}}$	$z(t+1)$	Eq.4.7, Eq.4.8	Eq.4.16	×	Eq.4.33, Eq.4.34	Eq.4.41	Eq.4.48	×
			$\mathbf{x}_{i_p}(t+1)$	×	×	$Opt_{\Delta:\mathbf{x}_{i_p}}$	×	×	×	$Opt_{\Delta:\mathbf{x}_{i_p}}$
			$\mathbf{x}_g(t+1)$	$Opt_{\Delta:\mathbf{x}_g}$		×	×	$Init_{\Delta:\mathbf{x}_g}$		
			$\mathbf{x}_s(t+1)$	×	×	×	Eq.4.32	×	×	×
	12	t=t+1								

Table 4.2: In UNIOA, It is possible to customize *Tuple.*, but it is impossible to customize

	Index	Tuple		Existing Method	Customization
Initialization	1	t=0			
	2	*Tuple.2* $Init_{\mathbf{x}}$		$Init_{\mathbf{x}}$	no
	3	*Tuple.1* f		$f(\mathbf{x}_i(t)), i=1,2,\ldots,M$	no
	4	*Tuple.3* $Init_{\Delta}$	$\mathbf{y}_i$	$Init_{\Delta:\mathbf{y}}$	yes
			$\mathbf{x}_{i_p}$	$Init_{\Delta:\mathbf{x}_{i_p}}$	no
			$\mathbf{x}_g$	$Init_{\Delta:\mathbf{x}_g}$	no
			$\mathbf{x}_s$	Eq.4.28(MFO)	yes
			z	Eq.4.3, Eq.4.4(BA). Eq.4.14(GOA). Eq.4.29, Eq.4.30(MFO). Eq.4.38(MBO). Eq.4.46(BOA)	yes
			w	commonly known as hyper-parameters	yes
Optimization	5	*Tuple.8* Stop condition T			yes
	6	*Tuple.5* Opt_{Δ}	$\mathbf{y}_i(t+1)$	Eq.4.5(BA). Eq.4.54(PSO)	yes
	7	*Tuple.4* $Opt_{\mathbf{x}}$		Eq.4.6(BA). Eq.4.15(GOA). Eq.4.31(MFO). Eq.4.39(MBO). Eq.4.47(BOA). Eq.4.55(PSO) Eq.4.22(CSA)	yes
	8	*Tuple.6* C		C1, C2	yes
	9	*Tuple.1* f		$f(\hat{\mathbf{x}}_i(t+1))$	no
	9	$S \rightarrow \mathbf{x}_i(t+1)$		$S1$, $S2$, $S3$, $S4$	yes
	11	*Tuple.5* Opt_{Δ}	$z(t+1)$	Eq.4.7, Eq.4.8(BA). Eq.4.16(GOA). Eq.4.33, Eq.4.34(MFO). Eq.4.41(MBO). Eq.4.48(BOA)	yes
			$\mathbf{x}_{i_p}(t+1)$	$Opt_{\Delta:\mathbf{x}_{i_p}}$	no
			$\mathbf{x}_g(t+1)$	$Opt_{\Delta:\mathbf{x}_g}$	no
			$\mathbf{x}_s(t+1)$	Eq.4.32(MFO)	yes
	12	t=t+1			

4.4 Summary

In this chapter, we give our Unified Framework — UNIOA that is the end goal in this work. We firstly re-write these seven algorithms in UNIOA in Section 4.2, in which each algorithm is re-written in UNIOA with eight tuples whose positions are also fixed. Meanwhile, we attempt to use primitive math knowledge to represent each tuple, which means basic math knowledge is enough to understand these swarm-based algorithms. Re-framed pseudo-code of each algorithm is also displayed in this section. The detailed representations of each tuple in each algorithm is shown in Table 4.1.

In Section 4.3, we discuss UNIOA deeply. The pseudo-code of the UNIOA is shown as Algorithm.9, in which we conclude the cooperation among these eight tuples. In Subsection 4.3.2, the visualization Figure 4.1 inspires the anto-designer UNIOA Package for swarm-based algorithms. Furthermore, we discuss the difference between the auto-designer for EA and the auto-designer UNIOA for swarm-based algorithms.

In conclusion, the Unified Framework UNIOA is constructed well in this chapter. based on seven selected algorithms.

Chapter 5

Experimental Setup of Benchmark Study

5.1 Motivation for experiments

As discussed in previous chapters, we find these seven selected swarm-based algorithms can be rewritten in terms of the unified framework $UNIOA$ (see Chapter 4). However, **to demonstrate the reliability of the unified framework $UNIOA$, theoretical evidence alone is not enough, and practical evidence is also essential**. Therefore, this chapter introduces our experiment plans to provide practical evidence.

Our experiments have two purposes. Firstly, we want to know if the unified framework we designed for these seven algorithms can adequately replace their original framework. In other words, for each of these seven algorithms, we want to know whether the performance of this algorithm written in the unified framework and the performance of this algorithm written in its original framework are same on solving a same set of questions. Secondly, we are also interested in how well they perform on solving a set of standardized benchmark problems when they are written in the unified framework. Therefore, our work has two kinds of comparisons:

- a comparison between two samples (mainly for the first purpose)
- a comparison among multiple (> 2) samples (mainly for the second purpose)

When the comparison happens between two samples, we prefer the Wilcoxon signed-rank test with a confidence level of 95%[1] to provide more convincing discussions. **The null hypothesis H_0 in this kind of comparison is the difference between two samples is on average zero on a random problem**.

Here, a benchmark environment — IOHprofiler [8] — is needed. It can allow us to implement algorithms under the same modules and also allow us to access enough optimization problems for measuring comparisons. Furthermore, considering these seven algorithms are created for solving continuous optimization problems [14–18, 30, 31], the BBOB problem set [10] is preferred as the benchmark problems.

IOHprofiler [8, 29] consists of an experimental part and a post-processing part. The experimental part is used to generate data that contains the number of function evaluations, current function (transformed and original) fitness and best-so-far function (transformed and original) fitness. Here, the transformed function is isomorphically transformed from the basic function. The transformed function exists in IOHprofiler in the form of instances[2]. Specifically, **in the experimental part**, when we manually code each algorithm in both framework, we need to customize the stop condition, the dimension of problems, the number of instances of each problem, and the number of runs of each instance .

The post-processing part will visualize the algorithm performance by analyzing the generated data. Specifically, **in the post-processing part**, although the IOHanalyzer[3] provides many measurements of the performance of algorithms, we only use two of them:

- ERT plot
- $ECDF$ measurement

Moreover, we only focus on the number of function evaluations at various given target values ('Fix-Target' section), although the IOHanalyzer also allow us to observe algorithms in the view of the target value at various given function evaluations ('Fixed-Budget' section).

The ERT plot visualizes how many evaluations (y-axis) each algorithm will take on solving a problem to reach a given target value (x-axis). As shown in Eq.5.1 [29], when an algorithm A is solving a problem f in the n dimensional environment, we customize 10^{-8} as the target value v and $n \times 10^4$ as the maximum number of evaluations B[4]. We

[1] The reason why we use this statistical method is experiments in Chapter 6 show samples in our cases are not normally distributed.

[2] More details about instances are in [8].

[3] https://iohanalyzer.liacs.nl/

[4] Its another name is allocated budget [29]

also customize that A will repeat solving f r times. Here, r is the product of the number of instances and the number of runs of each instance. $T(A,f,n,B,v,i)$ is the number of evaluations needed by A on solving f at i-th time.

$$ERT(A,f,n,B,v) = \frac{\sum_{i=1}^{r} \min\{T(A,f,n,B,v,i),B\}}{\sum_{i=1}^{r} \mathbf{1}(T(A,f,n,B,v,i) < \infty)} \tag{5.1}$$

If A can reach v within B, $T(A,f,n,B,v,i)$ is the number of actual evaluations taken by A. However, if A cannot reach v when the maximum number of evaluations is B, $T(A,f,n,B,v,i)$ will be set as $+\infty$. Therefore, the ERT measurement here is not a pure average over r times. While A repeats solving f r times, if $T(A,f,n,B,v,i)$ is $+\infty$, the ERT measurement doesn't count it as an available evaluation, but the maximum number of evaluations B will be added to the whole evaluations.

The $ECDF$ plot visualizes the probability (y-axis) of successful times within all times r. Specifically, one successful time is A can reach v under a given number of evaluations t (x-axis) when solving f in the n dimensional environment. As shown in Eq.5.2, the $ECDF$ plot focuses on how many successful evaluations $T(A,f,n,v,i) \leqslant t$ appear during r times, when A aims to reach only one target value v on solving f.

$$ECDF(A,f,n,v,t) = \frac{1}{r}\sum_{i=1}^{r} \mathbf{1}(T(A,f,n,v,i) \leqslant t) \tag{5.2}$$

In our experiments, we study a set of target values $V = \{10^e \mid e = 2, 1.8, 1.6, ..., -8\}$, and the $ECDF$ is calculated as an average value over various v, as shown in Eq.5.3 in which $|V|$ is the number of v in V.

$$ECDF(A,f,n,V,t) = \frac{1}{r\,|V|}\sum_{v \in V}\sum_{i=1}^{r} \mathbf{1}(T(A,f,n,v,i) \leqslant t) \tag{5.3}$$

Considering the ERT plot already visually displays the performance of algorithms, the $ECDF$ measurement is preferred as a quantitative measurement. Therefore, with the help of the R programming interface in IOHanalyzer, the area under each $ECDF$ curve AUC is performed in our work.

We conclude these settings of IOHprofiler environment in Table 5.1.

Table 5.1: Settings and usages in IOHprofiler environment.

Environment	Setups	Usages
IOHexperimenter	· minimum target value: 10^{-8}.	· generate running data of each problem over multiple instances and multiple runs.
	· maximum number of evaluations: $n \times 10^4$.	
	· dimensions of problems: $n = 5, 20$.	
	· the number of instances for each problem: 5.	
	· the number of runs for each instance: 5.	
IOHanalyzer	· web-based GUI	· observe ERT plots.
	· R programming interfaces.	· calculate AUC of $ECDF$.

5.2 Summary

In this chapter, we introduce the purpose of doing practical experiments is to give practical evidence on whether these seven algorithms can safely be rewritten in our designed unified framework. Meanwhile, we also observe the performance of algorithms written in our unified framework.

We will implement algorithms and compare their performances in IOHprofiler environment. Although the IOHprofiler provides various measurements, we only focus on two measurements: ERT plots in the view of quality and AUC values of $ECDF$ in the view of quantity.

Moreover, comparisons between two samples will be further analyzed by a paired sample test statistical method — Wilcoxon signed-rank test.

Chapter 6

Experimental Results

6.1 Introduction

In this chapter, we detail how many actual experiments are to achieve the purposes (see Chapter 5) of our work. More importantly, we display the experimental results and give discussions.

Our first purpose of experiments is to know whether the performance of the algorithm in its original framework is the same as the performance of the algorithm in our unified framework, when solving the same set of optimization problems. In other words, for each algorithm of these seven selected algorithms, the comparison is between two frameworks which means the framework structure shall be the only aspect affecting the algorithm performance and any other aspects with side effects on algorithms' performance shall be eliminated. Therefore, for our first purpose of experiments, we designed two groups of experiments:

Group.1 Experiments for avoiding side effects in Section 6.2.
For each algorithm in each framework, the evaluation method can be synchronous (syncE) or asynchronous (asyncE), and the method of calculating the global best individual can be synchronous (syncG) or asynchronous (asyncG) too. In many experimental attempts at measuring the performance of algorithms, we find these two aspects that might significantly affect the algorithm performance. Therefore, we do experiments to test whether syncE/asyncE and syncG/asyncG will significantly affect the algorithm performance, and significant effects shall be eliminated.

Specially, in this group of experiments, the first comparison is between the algorithm in syncE and the same algorithm in asyncE, and the second comparison is between the algorithm in syncG and the same algorithm in asyncG. Therefore, for these comparisons between two samples, the Wilcoxon signed-rank (see Section 4.1) is used.

Group.2 Comparison between the performance 'ORIGINAL' (abbr. orig) of the algorithm in its original framework and the performance 'UNIOA' of the same algorithm in our unified framework in Section 6.3
After eliminating some aspects with side effects, the experimental results about the comparison between 'ORIGINAL' and 'UNIOA' become more convinced. Because the comparison of each algorithm is between two samples, the Wilcoxon signed-rank (see Section 4.1) is also used in this group of experiments.

Our second purpose of experiments is to understand the comparison between these performances of these seven algorithms in our unified framework, and to give an overview understanding of these algorithms. Therefore, for our second purpose of experiments, we designed one group of experiments:

Group.3 Observations among performances 'UNIOA' of these seven algorithms in Section 6.4. In this group of experiment, there is no further statistical measurement, because the purpose of this group focuses on the observation, rather than the comparison.

The summary of this chapter is displayed in Section 6.5.

6.2 Experiments for avoiding side effects

6.2.1 Results

In many experimental attempts, we doubt that the way of calculating the fitness and the way of the global best individual will significantly impact the performance of algorithms. Their impact might mislead our conclusion on whether our unified framework works correctly in replacing the original framework. According to the original implementation of each algorithm, we found that there are two ways to calculate fitness:

- asynchronous Evaluation (abbr. asyncE): each time updating one individual, this individual's fitness is immediately calculated.
- synchronous Evaluation (abbr. syncE): only after the entire population is updated, the fitness of each individual in the entire population is then calculated simultaneously.

There are also two ways to calculate the global best one individual $\mathbf{x}_g$:

- asynchronous $\mathbf{x}_g$ (abbr. asyncG): each time updating one individual, the $\mathbf{x}_g$ is updated by comparing with this one individual.
- synchronous $\mathbf{x}_g$ (abbr. syncG): only after the entire population is updated, the $\mathbf{x}_g$ is updated by comparing with the entire updated population.

Therefore, for avoid side effects from asyncE/syncE and asyncG/syncG, we constructed 33 sub-experiments [1] in which:

- Each UNIOA has two options: evaluation is synchronous or asynchronous, but $\mathbf{x}_g$ is always synchronous.
- Each ORIGINAL has three options: when the evaluation is synchronous, the $\mathbf{x}_g$ is synchronous, but when the evaluation is asynchronous, the $\mathbf{x}_g$ is synchronous or asynchronous.

From Figure 6.1 in which y-axis is each algorithm whose evaluation method could be synchronous or asynchronous and its corresponding AUC values obtained in IOHanalyzer are shown on x-axis, it is observed that the two distributions of 24 AUC values for syncE algorithm and asyncE algorithm are the same. Even the outliers from syncE algorithm and asyncE algorithm are also significantly near each other. Same observations can be found when dimension n is 5 and when dimension n is 20. Meanwhile, in Table 6.1, each case represents one comparison between one algorithm whose evaluation method is synchronous or is asynchronous. We find that all p values are larger than 5%. It means **the data obtained from syncE algorithm and asyncE algorithm does not reject the hull hypothesis, no matter which framework it belongs to and no matter which dimension it is in. In other words, .**

[1] Generally, for the five algorithms who use $\mathbf{x}_g$, there should be $5 \times 2 \times 2 \times 2 = 40$ sub-experiments in which each algorithm has two frameworks, and each framework has two evaluation options, and each evaluation option has two $\mathbf{x}_g$ calculation options. However, we found it was impossible to calculate $\mathbf{x}_g$ asynchronously when the evaluation was synchronous. The reason is when the evaluation is already synchronous, it means the *for loop* for $\mathbf{x}_i$ already stops, and it is impossible for the $\mathbf{x}_g$ to come back into the *for loop*. Moreover, for UNIOA, when the evaluation is asynchronous, it is meaningless to set the $\mathbf{x}_g$ asynchronous also, which is because in UNIOA, other components also have to be changed when the evaluation and $\mathbf{x}_g$ are simultaneously asynchronous, then it is impossible to decide which component exactly affects the algorithm performance. Therefore, there are only $5 \times 2 \times 2 \times 2 - 5 \times 2 \times 1 \times 1 - 5 \times 1 \times 1 \times 1 = 25$ for five algorithms who use $\mathbf{x}_g$. Meanwhile, for the two algorithms who didn't use $\mathbf{x}_g$, there will be $2 \times 2 \times 2 = 8$ experiments in which each algorithm has two framework, and each framework has two evaluations. Therefore, there are total $25 + 8 = 33$ experiments. **??** and **??** lists details about how these 33 sub-experiments are constructed.

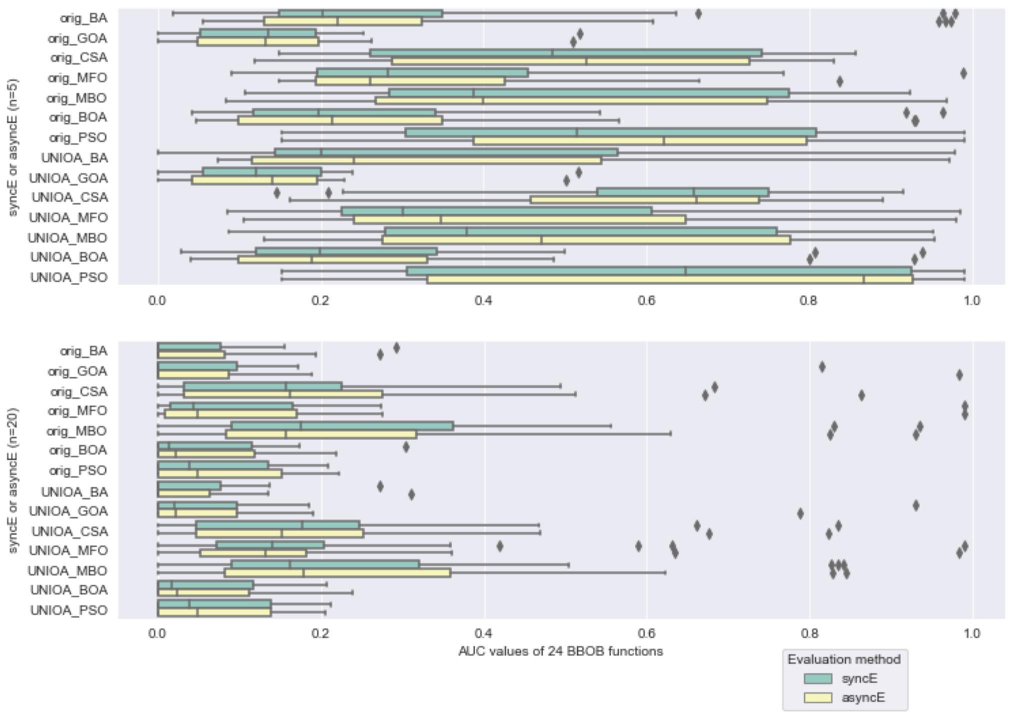

Figure 6.1: Distribution of 24 AUC values for paired algorithm performances in syncE or asyncE, when $n = 5$ or $n = 20$.

Table 6.1: P-values of Wilcoxon signed-rank test for difference on average between performances of paired algorithms in syncE or asyncE for a random optimization function, when $n = 5$ or $n = 20$.

Case	p-value (n=5)	p-value (n=20)
orig_BA_syncE_or_asyncE_syncG	0.3313349065031326	0.22886900026455015
orig_GOA_syncE_or_asyncE_syncG	0.6261109162613098	0.6673653270475619
orig_CSA_syncE_or_asyncE	0.6475683676310555	0.8862414820514412
orig_MFO_syncE_or_asyncE	0.2530979089471155	0.8409995729722781

continued . . .

...continued

Case	p-value (n=5)	p-value (n=20)
orig_MBO_syncE_or_asyncE_syncG	0.31731050786291415	0.17469182698689223
orig_BOA_syncE_or_asyncE_syncG	0.7750969621959847	0.2773986245500173
orig_PSO_syncE_or_asyncE_syncG	0.954431397113681	0.19449921074798193
UNIOA_BA_syncE_or_asyncE_syncG	0.24142655338204444	0.5968445170755943
UNIOA_GOA_syncE_or_asyncE_syncG	0.6465582592577419	0.6233436223982587
UNIOA_CSA_syncE_or_asyncE	0.9090113066460508	0.9430276454464167
UNIOA_MFO_syncE_or_asyncE	0.6475683676310555	0.38327738184229543
UNIOA_MBO_syncE_or_asyncE_syncG	0.265156633625956	0.24697273165906564
UNIOA_BOA_syncE_or_asyncE_syncG	0.09749059620220792	0.629275096131297
UNIOA_PSO_syncE_or_asyncE_syncG	0.954431397113681	0.4655179892460418

From Figure 6.2 in which y-axis is each algorithm whose $\mathbf{x}_g$ calculation method could be synchronous or asynchronous and its corresponding AUC values obtained in IOHanalyzer are shown on x-axis, it is observed that the two distributions of 24 AUC values for syncG algorithm and for asyncG algorithm are varying from algorithms. For example, the difference of paired performances between syncG BA and asyncG BA is significant when dimension n is 5 and when dimension n is 20, however, this kind of difference shown in Figure 6.2 is not significant in other algorithms. From Table 6.2 in which each case represents one comparison between each algorithm whose $\mathbf{x}_g$ calculation method is synchronous or is asynchronous, it is observed that when dimension n is 5, the data obtained from orig_GOA algorithm rejects the hull hypothesis which means the difference between syncG and asyncG orig_BA **Huilin: check the formulation with someone** algorithm's performance is significant. Such same observation is also found in orig_GOA algorithm when dimension n is 20.

Considering the difference is not significant in other cases, further analysis is needed to make the role of $\mathbf{x}_g$ explicit in unsignificant cases (orig_MBO, orig_BOA, orig_PSO) compared to significant cases (orig_BA, orig_GOA). From Table 6.3 that lists the way of each algorithm using $\mathbf{x}_g$ in ORIGINAL framework and UNIOA framework, it is observed that the effect of $\mathbf{x}_g$ is different in these algorithms, although $\mathbf{x}_g$ affects the quality of optimization in all of these algorithms. For example, $\mathbf{x}_g$ is directly added to objective solution $\mathbf{x}_i$ in BA and GOA, but in MBO, BOA and PSO, $\mathbf{x}_g$ is scaling down by multiplying a very small decimal (BOA), or by subtracting a larger value (PSO), or by reducing the probability of using $\mathbf{x}_g$ (MBO).

Therefore, we conclude that **syncG and asyncG algorithms have different performance on average for a random optimization BBOB function in ORIGINAL framework, no matter which dimension it is in**. **Huilin: add a conclusion: which version do you decide to use for the rest of the paper and why**

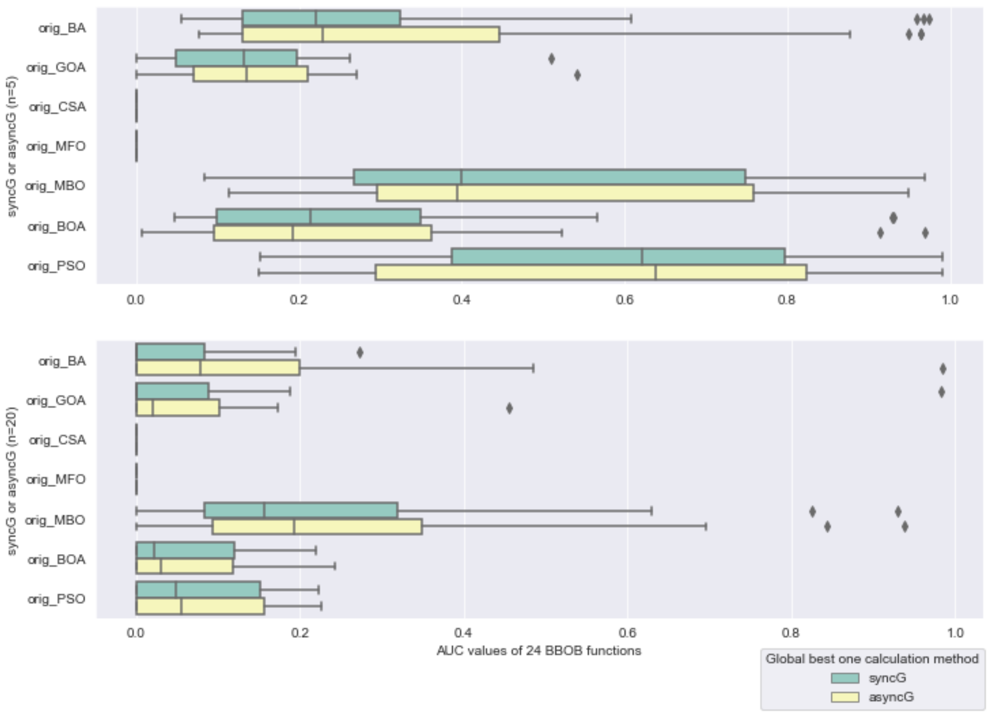

Figure 6.2: Distribution of 24 AUC values for paired algorithm performances in syncG or asyncG, when $n = 5$ or $n = 20$.

Table 6.2: P-values of Wilcoxon signed-rank test for difference on average between performances of paired algorithms in syncG or asyncG for a random optimization function, when $n = 5$ or $n = 20$.

Case	p-value (n=5)	p-value (n=20)
orig_BA_asyncE_syncG_or_asyncG	0.22459133607647186	0.004638292309835263
orig_GOA_asyncE_syncG_or_asyncG	0.0042199644589382525	0.17318594569671153
CSA	NO USE $\mathbf{x}_g$	
MFO	NO USE $\mathbf{x}_g$	
orig_MBO_asyncE_syncG_or_asyncG	0.19854279368666194	0.07411601304083099
orig_BOA_asyncE_syncG_or_asyncG	0.8192020334011836	0.2773986245500173
orig_PSO_asyncE_syncG_or_asyncG	0.44045294529422474	0.964388671614557

Table 6.3: How each algorithm uses $\mathbf{x}_g$ in ORIGINAL framework and UNIOA framework.

H_0	Algorithm	$\mathbf{x}_g$ role in ORIGINAL	$\mathbf{x}_g$ role in UNIOA
reject	BA	Eq.2.5, Eq.2.6	Eq.4.6
reject	GOA	Eq.2.10	Eq.4.15
-	CSA	NO USE $\mathbf{x}_g$	
-	MFO	NO USE $\mathbf{x}_g$	
not reject	MBO	Step (6)	Eq.4.39
not reject	BOA	Eq.2.28	Eq.4.47
not reject	PSO	Eq.2.34	Eq.4.54

6.2.2 Conclusion

According to the experimental results in Section 5.2, we can conclude that the different effects from asyncE and syncE could be ignored, but whether the way of calculating $\mathbf{x}_g$ is synchronous or asynchronous much likely has an impact on measuring the performance of algorithms. In other words, when the performance of UNIOA is the same as the performance of ORIGINAL, it is much likely because different ways of calculating $\mathbf{x}_g$ force their performance to behave same, but not because the unified framework itself does not affect the algorithm performance.

Therefore, as shown in Table 6.4, we modify asyncG to syncG, but not change their ways of evaluation. Here, considering the way of calculating $\mathbf{x}_g$ in UNIOA is synchronous, we choose to change asyncG to syncG, but not change syncG to asyncG. This is because we want to avoid misleading our conclusions about whether UNIOA performs same as ORIGINAL. For example, when the performance of UNIOA is different from the performance of ORIGINAL, we hope the reason is the fault of our unified framework, but not the way of calculating $\mathbf{x}_g$.**Huilin: not understandable**

Table 6.4: When reproducing their original framework, if the way of calculating the Evaluation and the $\mathbf{x}_g$ changes.

Algorithm	Evaluation		$\mathbf{x}_g$	
	in original code	in this work	in original code	in this work
orig_BA	asyncE	asyncE	asyncG	syncG
orig_GOA	syncE	syncE	syncG	syncG
orig_CSA	syncE	syncE	no use	no use
orig_MFO	asyncE	asyncE	no use	no use
orig_MBO	syncE	syncE	syncG	syncG
orig_BOA	asyncE	asyncE	asyncG	syncG
orig_PSO	asyncE	asyncE	asyncG	syncG

6.3 Comparing UNIOA to the original framework

6.3.1 Results

In Section 5.2, we eliminate side effects from $\mathbf{x}_g$ calculation method when reproducing these algorithms in ORIGINAL framework. Therefore, it is much safer now to discuss the difference between ORIGINAL and UNIOA.

From Figure 6.3 in which y-axis is each algorithm whose framework is ORIGINAL or UNIOA and its corresponding AUC values obtained in IOHanalyzer are shown on x-axis, it is observed that the two distributions of 24 AUC values for ORIGINAL algorithm and for UNIOA algorithm are the same. Even the outliers from ORIGINAL algorithm and from UNIOA algorithm are also significantly near to each other. Same observations can be found when dimension n is 5 and when dimension n is 20.

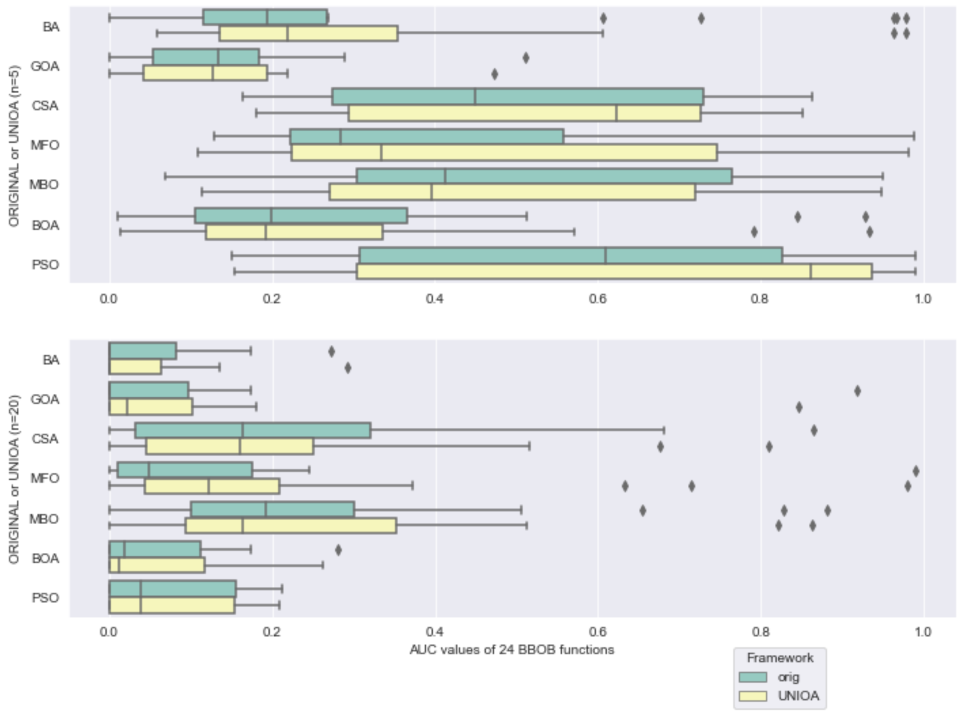

Figure 6.3: Distribution of 24 AUC values for paired algorithm performances in orig framework or UNIOA framework, when $n = 5$ or $n = 20$.

Meanwhile, from Table 6.5 in which each case represents one comparison between each algorithm whose framework is ORIGINAL or UNIOA, it is observed that when dimension n is 5, all p values are larger than 5% which means when $n = 5$, the data obtained from ORIGINAL algorithm and UNIOA algorithm does not reject the hull hypothesis, in other words, **when $n = 5$, the difference between ORIGINAL and UNIOA is not significant**.

Furthermore, in Table 6.5, it is also observed that when dimension n is 20, the difference is significant between orig_PSO and UNIOA_PSO as well as between orig_MFO and UNIOA_MFO. However, it can be acceptable if the difference direction is that UNIOA is better than ORIGINAL in our study. For example, when $n = 20$, UNIOA_PSO has higher AUC values in 16 of 24 BBOB optimization functions, and UNIOA_MFO has higher AUC values in 20 of 24 BBOB optimization functions.

Table 6.5: (1) P-values of Wilcoxon signed-rank test for difference on average between performances of paired algorithms in ORIGINAL framework or UNIOA framework for a random optimization function, when $n = 5$ or $n = 20$. (2) The number of wins orig had or UNIOA had when comparing their own AUC value throughout 24 BBOB functions.

Dimension n	Case	p-value	orig win	UNIOA win
$n = 5$	BA	0.11276741428797125	7	17
	GOA	0.0804264513256041	15	9
	CSA	0.05933346675499405	9	15
	MFO	0.17931823604537578	11	13
	MBO	0.37577150825113037	14	10
	BOA	0.8638867905449266	12	12
$n = 20$	PSO	0.006090717662464104	8	16
	BA	0.6558614582873129	3	21
	GOA	0.06569860480594888	3	21
	CSA	0.7860401269462913	12	12
	MFO	0.0001911189425107099	4	20
	MBO	0.14097231038635524	15	9
	BOA	0.8327296566664774	6	18
	PSO	0.6064080161085506	6	18

6.3.2 Conclusion

According to the experimental results in Section 6.3, we can conclude that at least in these seven algorithms, the ORIGINAL framework can be safely replaced by our unified UNIOA framework, with respect to actual algorithm performance.

For example, in Figure 6.4 showing fix-target curves, the purple line denotes the MFO in ORIGINAL, and the orange line denotes the MFO in UNIOA. We can find two lines are close to each other in most of 24 functions, which means the MFO algorithm performs same in two different frameworks when solving most of 24 optimization problems. Sometimes, the MFO in ORIGINAL needs more evaluations to reach the same target value as the MFO in UNIOA, when solving F1. Sometimes, the MFO in UNIOA can obtain much better optimization results than the MFO in ORIGINAL, when solving F2, F7, F17, F21. However, the MFO in ORIGINAL performs slightly better than the MFO in UNIOA when solving F8, F9, F15.

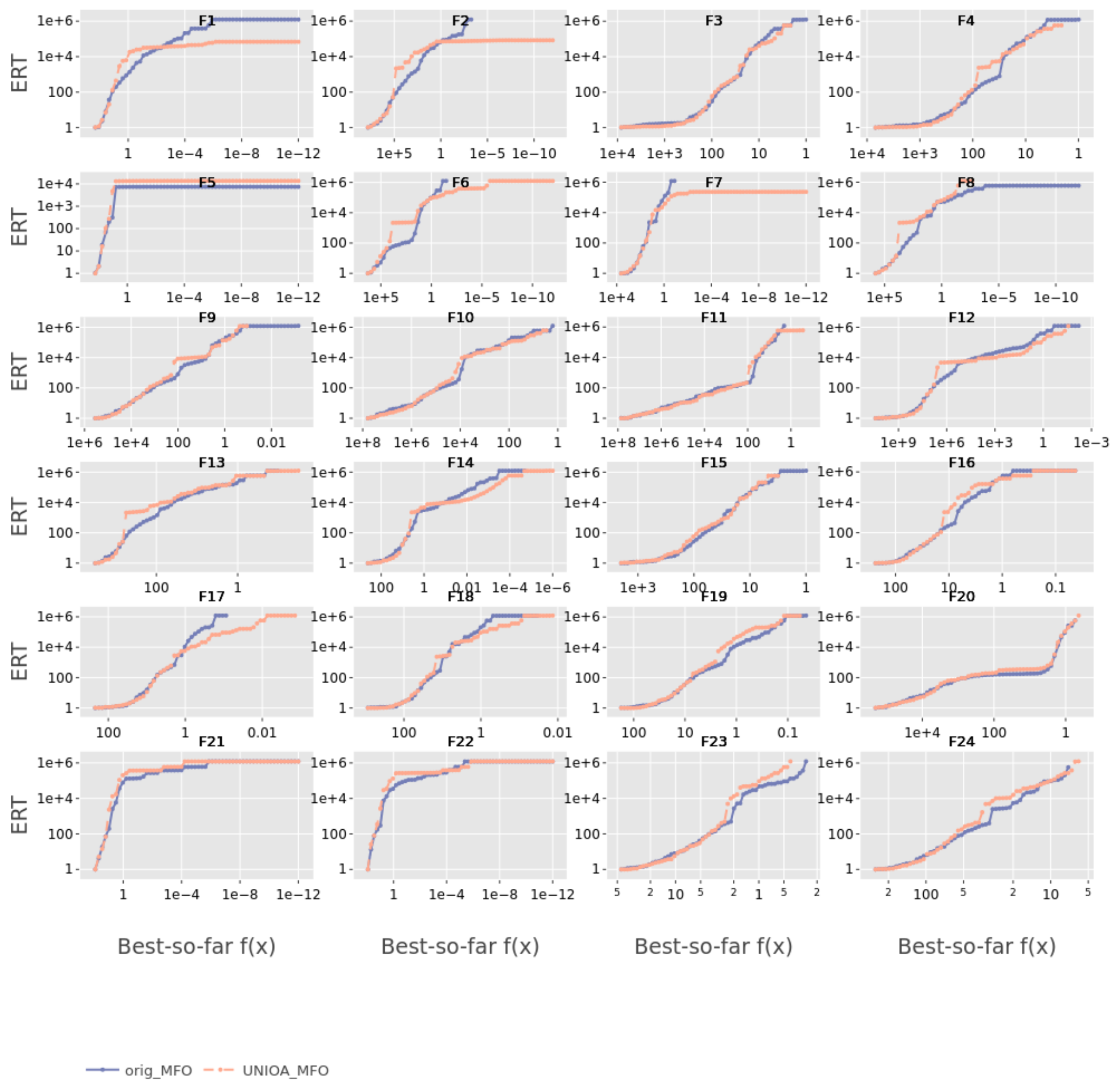

Figure 6.4: ERT values of the MFO in UNIOA and in ORIGINAL, when $n = 5$.

6.4 Observations among algorithms in UNIOA

In this section, we want to understand the performance of these seven algorithms in UNIOA, which solving a set of standardized benchmark problems. In Figure 6.5, when the dimension increases (from $n = 5$ to $n = 20$), the probability that the algorithm will reach the target value under limited evaluations is reducing, no matter which algorithm it is.

When the dimension is 5, the PSO in UNIOA is much more likely to generate more target values on solving most of 24 BBOB problems. Meanwhile, the CSA, MBO and MFO in UNIOA have similar performances on solving 24 problems. The same observation can also be found between the BA and BOA in UNIOA. Lastly, the GOA in UNIOA performs obviously worst compared to the other six algorithms when solving 24 problems.

Figure 6.6 also displays the performance of these seven algorithms when the dimension is 5. When solving F3 and F4, the MBO in UNIOA can generate much more optimal solutions before reaching the maximum evaluations. However, other six algorithms obtain much worse solutions, although the maximum evaluations have been finished. Furthermore, same observations can be found in solving F16, F17, F18, F19, but the CSA in UNIOA is the winner this time.

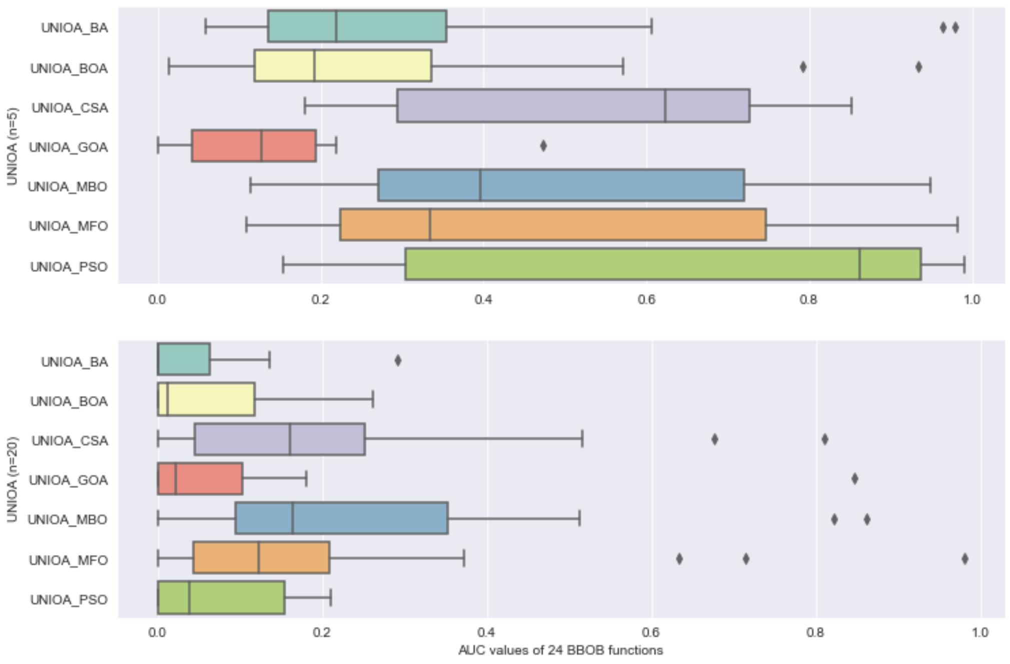

Figure 6.5: Distribution of 24 AUC values for UNIOA algorithms' performance, when $n = 5$ (top) or $n = 20$ (bottom).

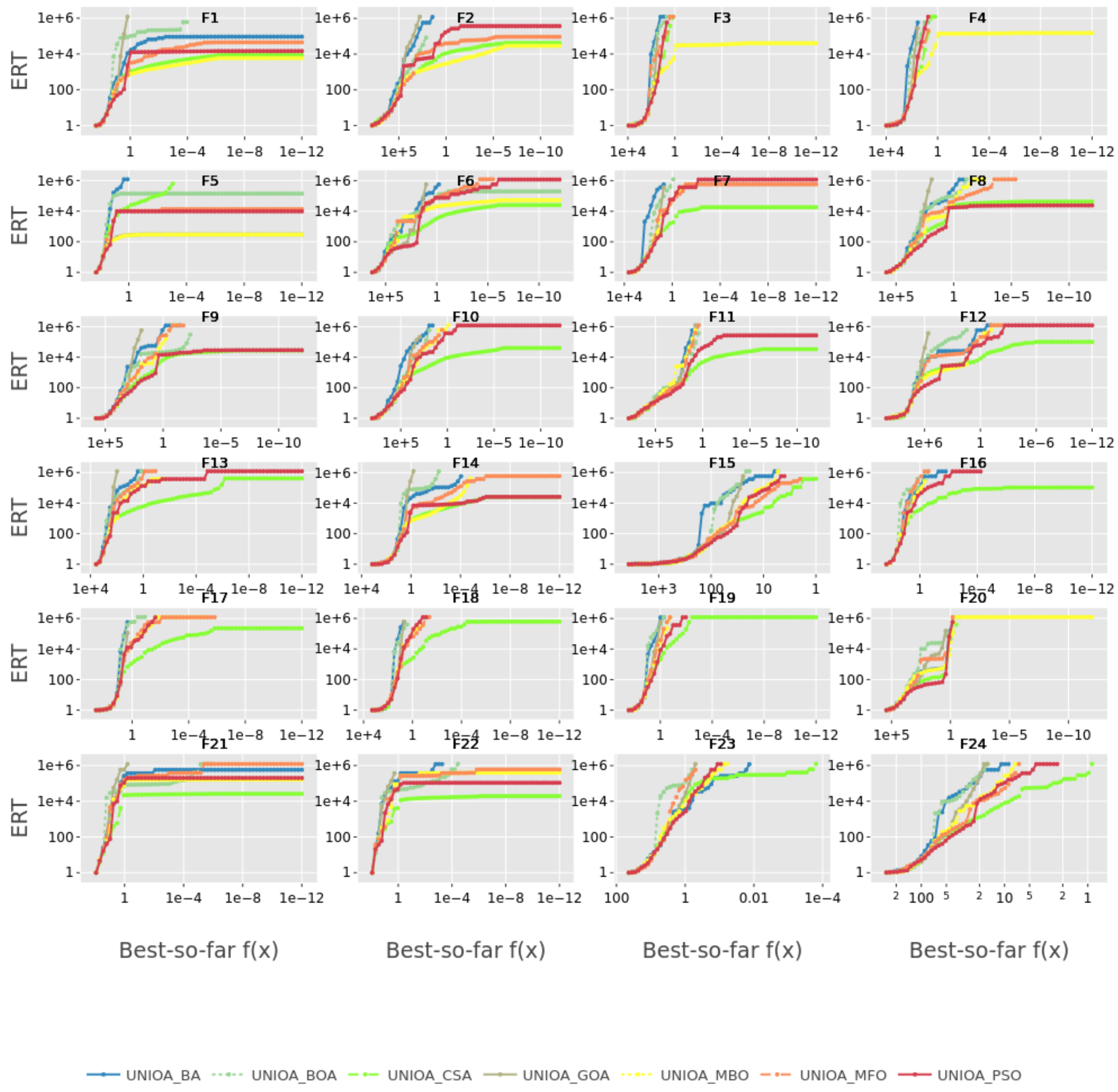

Figure 6.6: ERT values of seven algorithms in UNIOA, when $n = 5$.

6.5 Summary

In this chapter, we discuss our actual experimental results. Section 6.3 proves that the unified framework UNIOA works correctly, and it can replace the original framework of these seven selected algorithms.

Furthermore, Section 6.4 also discusses the performance of these seven algorithms when they are organized in the unified framework UNIOA.

Chapter 7

Summary

7.1 Conclusions in our studies

Within these six chapters presented in this work, we come to one observation: **seven selected swarm-based optimization algorithms can be re-written in the same unified framework whose terminologies are limited, and the procedure of algorithms is fixed**.

Chapter 1 pointed out that similarities among many swarm-based optimization algorithms mislead the innovation of developing swarm-based optimization algorithms. Representing the same concept in different terminologies does not bring a new algorithm but increases the workload of filtering algorithms. In other words, the problem we expected to solve in this work is how to bring different swarm-based optimization algorithms into a general/unified environment for future more meaningful discussion and more effective studies. **In short, we aim to build up a unified environment for swarm-based optimization algorithms**. Our work starts with seven algorithms, including old classical algorithms and new modern algorithms.

Firstly, Chapter 2 studied how to make every algorithm speak the same language in this unified environment. Although these seven algorithms come from different nature analogies, the essence of information carried by their various terminologies is the same. For example, the memory in CSA is the personal best individual in PSO. We captured the commonalities of information to classify various terminologies among these algorithms, and various terminologies with the same information are defined as unified terminology. We found that 20 unified terminologies (see Table 2.1) already can cover the entire different terminologies appearing in these seven algorithms. **In short, these selected algorithms can speak the same language – 20 unified terminologies**.

Then, Chapter 3 studied how to make every algorithm walk in the same line in this unified environment. Each algorithm follows different optimization procedures, but these procedures also have similarities in the view of their functions. For example, the update of step-size y always happens before updating objective x. We found that eight tuples (see Table 3.1) can represent all of these seven algorithms. Meanwhile, the positions of these eight tuples (see Algorithm.1) can also be the same among these seven algorithms. **In short, these selected algorithms can walk in the same line with the same steps – one unified procedure with eight unified tuples**.

Lastly, Chapter 4 displayed the unified framework $UNIOA$ by combining the findings in Chapter 2 and Chapter 3. We found that each algorithm can be re-framed into the $UNIOA$ (see Section 4.2). Meanwhile, we summarized two different treatments for outliers and four different ways of selection (see Table 4.1). The Unified Framework $UNIOA$ leads to a practical application — $UNIOA$ (see Subsection 4.3.2)— an auto-designer for swarm-based optimization algorithms. **In short, the Unified Framework $UNIOA$ is constructed well for these seven algorithms**.

Furthermore, Chapter 5 and Chapter 6 provided practical evidence that the $UNIOA$ can safely replace the original framework among these seven algorithms. **In short, the Unified Framework $UNIOA$ works correctly on these seven algorithms**.

Therefore, we summarized our contributions as follows:

- Main contributions:
  - A Unified Framework $UNIOA$. (see Chapter 4)
  - 20 unified terminologies. (see Chapter 2)
  - Eight unified tuples. (see Chapter 3)
  - One unified procedure. (see Chapter 3)
  - Primitive math is the only knowledge in our unified representations. (see Chapter 4)
- Bonus contributions:
  - A demo of auto-designer $UNIOA$ for swarm-based optimization algorithms. (see Subsection 4.3.2, `https://github.com/Huilin-Li/UNIOA.git`)

- The way of calculating the fitness dose not impact the performance of algorithms. (see Section 6.2)
- The way of calculating the global best individual dose impact the performance of algorithms. (see Section 6.2)
- A comparison of performances among these seven algorithms. (see Section 6.4)

The experimental parts of our work have organized in https://github.com/Huilin-Li/ThesisProject_Huilin.git.

7.2 Limitations in our analysis

Our work also has some possible limitations. The first limitation is how we use statistics to test our hypotheses. In Chapter 5 and Chapter 6, we studied whether one algorithm in our unified framework can perform the same as this algorithm in its original framework in solving a random problem in a set of problems. However, it is also worth testing whether one algorithm in our unified framework can perform the same as this algorithm in its original framework in solving each problem in a set of problems. In the first case, each problem has only one AUC value, and the samples for the statistic test are 24 AUC values for 24 problems. The statistic test only needs to be done once to discuss whether the $UNIOA$ can perform correctly in solving a random problem in the set of problems. In the second case, each problem has 25 values which are the number of runs of each problem. Every problem will have a statistic test, which means there shall be 24 pairs of samples, and each pair of samples is 25 AUC values for one problem. The discussion of the second case shall be on whether the $UNIOA$ can work correctly in solving Problem-1 or Problem-2 or $\cdots$ or Problem-24.

The second limitation is the auto-designer for swarm-based optimization algorithms. In our demo, the number of z-relative step-size can only be added to a new algorithm once. However, according to our observations in these seven selected algorithms, the algorithm can use multiple z-relative step-size simultaneously.

The third limitation is the definition of the global best one individual $\mathbf{x}_g$. In our work, $\mathbf{x}_g$ is the best one among the current population that is updated after the selection process and the previous population that is also generated after the selection process. However, in original implementations of some algorithms, $\mathbf{x}_g$ is the best one among the current population that is updated before the selection process and the previous population that is generated after the selection process. Although we believe that the first definition shall be correct, we keep this discussion open for further studies.

Lastly, we also expect more experiments, including more runs and more instances, for giving more convinced discussions.

7.3 Future work

A deeper and more advanced analysis of our work has been left for the future. There are some ideas we would like to try in the future.

Firstly, discussions on the level of mathematics shall also be given. In our unified framework $UNIOA$, primitive math knowledge is the only knowledge readers need to understand algorithms. The mathematics formulas determine the quality of optimization among different algorithms. Therefore, we believe there could be more exciting findings when comparing these algorithms on the math level in our unified framework.

Secondly, since we organized 20 components to cover these seven algorithms, we are also curious about how to compare these algorithms only on the level of these 20 components. For example, a comparison only on $Opt_{\mathbf{x}}$. Furthermore, it is also worth studying how to improve each of these 20 components. For example, is there a more intelligent method to initialize the population, or is there a more efficient method to update step-size?

Thirdly, we are also interested in comparing different unified frameworks for nature-inspired algorithms, such as the UF [19].

Lastly, we are interested in the extension of our unified framework. The first kind of extension is to extend our unified framework from seven selected algorithms into the whole of swarm-based optimization algorithms. Although we already slightly discussed this kind of extension in Subsection 2.3.2, we hope there will be a more assured discussion on our unified framework that can also much likely cover most of the swarm-based optimization algorithms. The second kind of extension is to extend our unified framework from swarm-based optimization algorithms into the whole of nature-inspired optimization algorithms. Specifically, we are curious whether our unified framework can also work correctly in the evolutionary algorithms.

Acknowledgement

References

[1] History of optimization:lines of development, breakthroughs, applications and curiosities, and links. http://www.mitrikitti.fi/opthist.html#linx. Accessed: 20210-12-12.

[2] A novel metaheuristic method for solving constrained engineering optimization problems: Crow search algorithm. Computers and Structures, 169:1–12, 6 2016.

[3] Sankalap Arora and Satvir Singh. Butterfly optimization algorithm: a novel approach for global optimization. Soft Computing, 23:715–734, 2 2019.

[4] Ayodeji Remi-Omosowon and Yasser Gonzalez. Pyeasyga: A simple and easy-to-use implementation of a genetic algorithm library in python. https://github.com/remiomosowon/pyeasyga.

[5] Rick Boks, Hao Wang, and Thomas Bäck. A modular hybridization of particle swarm optimization and differential evolution. In Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion, pages 1418–1425, 2020.

[6] Mark A Coletti, Eric O Scott, and Jeffrey K Bassett. Library for evolutionary algorithms in python (leap). In Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion, pages 1571–1579, 2020.

[7] Carola Doerr. Theory of iterative optimization heuristics: From black-box complexity over algorithm design to parameter control., 2020.

[8] Carola Doerr, Hao Wang, Furong Ye, Sander van Rijn, and Thomas Bäck. Iohprofiler: A benchmarking and profiling tool for iterative optimization heuristics. 10 2018.

[9] Ahmed Fawzy Gad. Pygad: An intuitive genetic algorithm python library. arXiv e-prints, pages arXiv–2106, 2021.

[10] Steffen Finck, Nikolaus Hansen, Raymond Ros, and Anne Auger. Real-parameter black-box optimization benchmarking 2009: Presentation of the noiseless functions. Technical report, Citeseer, 2010.

[11] Iztok Fister Jr, Uroš Mlakar, Janez Brest, and Iztok Fister. A new population-based nature-inspired algorithm every month: is the current era coming to the end. In Proceedings of the 3rd Student Computer Science Research Conference, pages 33–37. University of Primorska Press, 2016.

[12] Simon Fong, Xi Wang, Qiwen Xu, Raymond Wong, Jinan Fiaidhi, and Sabah Mohammed. Recent advances in metaheuristic algorithms: Does the makara dragon exist? The Journal of Supercomputing, 72(10):3764–3786, 2016.

[13] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. DEAP: Evolutionary algorithms made easy. Journal of Machine Learning Research, 13:2171–2175, jul 2012.

[14] Ahmed Helmi and Ahmed Alenany. An enhanced moth-flame optimization algorithm for permutation-based problems. Evolutionary Intelligence, 13(4):741–764, 2020.

[15] Haouassi Hichem, Merah Elkamel, Mehdaoui Rafik, Maarouk Toufik Mesaaoud, and Chouhal Ouahiba. A new binary grasshopper optimization algorithm for feature selection problem. Journal of King Saud University-Computer and Information Sciences, 2019.

[16] James Kennedy and Russell Eberhart. Particle swarm optimization. In Proceedings of ICNN'95-international conference on neural networks, volume 4, pages 1942–1948. IEEE, 1995.

[17] Jonas Krause, Jelson Cordeiro, Rafael Stubs Parpinelli, and Heitor Silverio Lopes. A survey of swarm algorithms applied to discrete optimization problems. In Swarm Intelligence and Bio-Inspired Computation, pages 169–191. Elsevier, 2013.

[18] Soukaina Laabadi, Mohamed Naimi, Hassan El Amri, and Boujemâa Achchab. A binary crow search algorithm for solving two-dimensional bin packing problem with fixed orientation. Procedia Computer Science, 167:809–818, 2020.

[19] Bo Liu, Ling Wang, Ying Liu, and Shouyang Wang. A unified framework for population-based metaheuristics. Annals of Operations Research, 186:231–262, 6 2011.

[20] Federico Marini and Beata Walczak. Particle swarm optimization (pso). a tutorial. Chemometrics and Intelligent Laboratory Systems, 149:153–165, 2015.

[21] Seyedali Mirjalili. Moth-flame optimization algorithm: A novel nature-inspired heuristic paradigm. Knowledge-based systems, 89:228–249, 2015.

[22] Christian S Perone. Pyevolve: a python open-source framework for genetic algorithms. Acm Sigevolution, 4(1):12–20, 2009.

[23] Adam P Piotrowski, Jaroslaw J Napiorkowski, and Pawel M Rowinski. How novel is the "novel" black hole optimization approach? Information Sciences, 267:191–200, 2014.

[24] Shahrzad Saremi, Seyedali Mirjalili, and Andrew Lewis. Grasshopper optimisation algorithm: Theory and application. Advances in Engineering Software, 105:30–47, 3 2017.

[25] Kenneth Sörensen. Metaheuristics-the metaphor exposed. International Transactions in Operational Research, 22:3–18, 1 2015.

[26] Éric D Taillard, Luca M Gambardella, Michel Gendreau, and Jean-Yves Potvin. Adaptive memory programming: A unified view of metaheuristics. European Journal of Operational Research, 135(1):1–16, 2001.

[27] Alberto Tonda. Inspyred: Bio-inspired algorithms in python. Genetic Programming and Evolvable Machines, 21(1):269–272, 2020.

[28] Gai Ge Wang, Suash Deb, and Zhihua Cui. Monarch butterfly optimization. Neural Computing and Applications, 31:1995–2014, 7 2019.

[29] Hao Wang, Diederick Vermetten, Furong Ye, Carola Doerr, and Thomas Bäck. Iohanalyzer: Performance analysis for iterative optimization heuristic. 7 2020.

[30] Dongfang Yang, Xitong Wang, Xin Tian, and Yonggang Zhang. Improving monarch butterfly optimization through simulated annealing strategy. Journal of Ambient Intelligence and Humanized Computing, pages 1–12, 2020.

[31] Xin-She Yang. A new metaheuristic bat-inspired algorithm. 4 2010.

[32] Xin-She Yang. Nature-inspired optimization algorithms: Challenges and open problems. Journal of Computational Science, 46:101104, 2020.

"

Appendices

A Original positions of eight tuples in selected algorithms

Algorithm 10 original positions of eight tuples in BA

```
1: $t \leftarrow 0$                                              ▷ *iteration counter*
2: *Tuple.2* $Init_{\mathbf{x}}$                                 ▷ initialize population
3: *Tuple.1* $f$                                                 ▷ evaluation
4: *Tuple.3* $Init_{\Delta:w}$                                   ▷ initialize $w$-relative step-size $\Delta$
5: *Tuple.3* $Init_{\Delta:\mathbf{y}}$                          ▷ initialize $\mathbf{y}$-relative step-size $\Delta$
6: *Tuple.3* $Init_{\Delta:\mathbf{x}}$                          ▷ initialize $\mathbf{x}$-relative step-size $\Delta$
7: **while** *Tuple.8* $T$ **do**                                ▷ stop strategy
8:     *Tuple.3* $Init_{\Delta:z}$ and *Tuple.5* $Opt_{\Delta:z}$▷ initialize and update $z$-relative step-size $\Delta$
9:     *Tuple.5* $Opt_{\Delta:\mathbf{y}}$                       ▷ update $\mathbf{y}$-relative step-size $\Delta$
10:    *Tuple.4* $Opt_{\mathbf{x}}$                              ▷ update population
11:    *Tuple.6* $C$                                             ▷ outliers treatment
12:    *Tuple.1* $f$                                             ▷ evaluation
13:    *Tuple.7* $S$                                             ▷ selection
14:    *Tuple.5* $Opt_{\Delta:\mathbf{x}}$                       ▷ update $\mathbf{x}$-relative step-size $\Delta$
15:    $t \leftarrow t + 1$
16: **end while**
```

Algorithm 11 original positions of eight tuples in GOA

```
 1: $t \leftarrow 0$                                              ▷ *iteration counter*
 2: *Tuple.2* $Init_{\mathbf{x}}$                                 ▷ initialize population
 3: *Tuple.1* $f$                                                 ▷ evaluation
 4: *Tuple.3* $Init_{\Delta:w}$                                   ▷ initialize $w$-relative step-size $\Delta$
 5: *Tuple.3* $Init_{\Delta:\mathbf{x}}$                          ▷ initialize $\mathbf{x}$-relative step-size $\Delta$
 6: **while** *Tuple.8* $T$ **do**                                ▷ stop strategy
 7:     *Tuple.3* $Init_{\Delta:z}$ and *Tuple.5* $Opt_{\Delta:z}$ ▷ initialize and update $z$-relative step-size $\Delta$
 8:     *Tuple.4* $Opt_{\mathbf{x}}$                              ▷ update population
 9:     *Tuple.6* $C$                                             ▷ outliers treatment
10:     *Tuple.1* $f$                                             ▷ evaluation
11:     *Tuple.5* $Opt_{\Delta:\mathbf{x}}$                       ▷ update $\mathbf{x}$-relative step-size $\Delta$
12:     $t \leftarrow t + 1$
13: **end while**
```

Algorithm 12 original positions of eight tuples in CSA

```
 1: $t \leftarrow 0$                                              ▷ *iteration counter*
 2: *Tuple.2* $Init_{\mathbf{x}}$                                 ▷ initialize population
 3: *Tuple.1* $f$                                                 ▷ evaluation
 4: *Tuple.3* $Init_{\Delta:w}$                                   ▷ initialize $w$-relative step-size $\Delta$
 5: *Tuple.3* $Init_{\Delta:\mathbf{x}}$                          ▷ initialize $\mathbf{x}$-relative step-size $\Delta$
 6: **while** *Tuple.8* $T$ **do**                                ▷ stop strategy
 7:     *Tuple.4* $Opt_{\mathbf{x}}$                              ▷ update population
 8:     *Tuple.6* $C$                                             ▷ outliers treatment
 9:     *Tuple.1* $f$                                             ▷ evaluation
10:     *Tuple.5* $Opt_{\Delta:\mathbf{x}}$                       ▷ update $\mathbf{x}$-relative step-size $\Delta$
11:     $t \leftarrow t + 1$
12: **end while**
```

Algorithm 13 original positions of eight tuples in MFO

```
1:  $t \leftarrow 0$                                                        ▷ iteration counter
2:  Tuple.2 $Init_{\mathbf{x}}$                                            ▷ initialize population
3:  Tuple.3 $Init_{\Delta:w}$                                  ▷ initialize $w$-relative step-size $\Delta$
4:  while Tuple.8 $T$ do                                                   ▷ stop strategy
5:      Tuple.3 $Init_{\Delta:z}$ and Tuple.5 $Opt_{\Delta:z}$ ▷ initialize and update $z$-relative step-size $\Delta$
6:      Tuple.6 $C$                                                        ▷ outliers treatment
7:      Tuple.1 $f$                                                        ▷ evaluation
8:      Tuple.3 $Init_{\Delta:\mathbf{x}}$ and Tuple.5 $Opt_{\Delta:\mathbf{x}}$ ▷ initialize and update $\mathbf{x}$-relative step-size $\Delta$
9:      Tuple.4 $Opt_{\mathbf{x}}$                                         ▷ update population
10:     $t \leftarrow t + 1$
11: end while
```

Algorithm 14 original positions of eight tuples in MBO

```
1:  $t \leftarrow 0$                                                        ▷ iteration counter
2:  Tuple.2 $Init_{\mathbf{x}}$                                            ▷ initialize population
3:  Tuple.1 $f$                                                            ▷ evaluation
4:  Tuple.3 $Init_{\Delta:w}$                                  ▷ initialize $w$-relative step-size $\Delta$
5:  Tuple.3 $Init_{\Delta:\mathbf{x}}$                         ▷ initialize $\mathbf{x}$-relative step-size $\Delta$
6:  while Tuple.8 $T$ do                                                   ▷ stop strategy
7:      Tuple.4 $Opt_{\mathbf{x}}$                                         ▷ update population
8:      Tuple.1 $f$                                                        ▷ evaluation
9:      Tuple.4 $Opt_{\mathbf{x}}$                                         ▷ update population
10:     Tuple.1 $f$                                                        ▷ evaluation
11:     Tuple.7 $S$                                                        ▷ selection
12:     Tuple.5 $Opt_{\Delta:\mathbf{x}}$                          ▷ update $\mathbf{x}$-relative step-size $\Delta$
13:     $t \leftarrow t + 1$
14: end while
```

Algorithm 15 original positions of eight tuples in BOA

```
 1: $t \leftarrow 0$                                    ▷ *iteration counter*
 2: *Tuple.2* $Init_{\mathbf{x}}$                      ▷ initialize population
 3: *Tuple.1* $f$                                      ▷ evaluation
 4: *Tuple.3* $Init_{\Delta:z}$                        ▷ initialize $z$-relative step-size $\Delta$
 5: *Tuple.3* $Init_{\Delta:\mathbf{x}}$               ▷ initialize $\mathbf{x}$-relative step-size $\Delta$
 6: **while** $T$ **do**                               ▷ stop strategy
 7:     *Tuple.4* $Opt_{\mathbf{x}}$                   ▷ update population
 8:     *Tuple.6* $C$                                  ▷ outliers treatment
 9:     *Tuple.1* $f$                                  ▷ evaluation
10:     *Tuple.7* $S$                                  ▷ selection
11:     *Tuple.5* $Opt_{\Delta:\mathbf{x}}$            ▷ update $\mathbf{x}$-relative step-size $\Delta$
12:     *Tuple.5* $Opt_{\Delta:z}$                     ▷ update $z$-relative step-size $\Delta$
13:     $t \leftarrow t + 1$
14: **end while**
```

Algorithm 16 original positions of eight tuples in PSO

```
 1: $t \leftarrow 0$                                    ▷ *iteration counter*
 2: *Tuple.2* $Init_{\mathbf{x}}$                      ▷ initialize population
 3: *Tuple.1* $f$                                      ▷ evaluation
 4: *Tuple.3* $Init_{\Delta:w}$                        ▷ initialize $w$-relative step-size $\Delta$
 5: *Tuple.3* $Init_{\Delta:\mathbf{y}}$               ▷ initialize $\mathbf{y}$-relative step-size $\Delta$
 6: *Tuple.3* $Init_{\Delta:\mathbf{x}}$               ▷ initialize $\mathbf{x}$-relative step-size $\Delta$
 7: **while** $T$ **do**                               ▷ stop strategy
 8:     *Tuple.5* $Opt_{\Delta:\mathbf{y}}$            ▷ update $\mathbf{y}$-relative step-size $\Delta$
 9:     *Tuple.4* $Opt_{\mathbf{x}}$                   ▷ update population
10:     *Tuple.6* $C$                                  ▷ outliers treatment
11:     *Tuple.1* $f$                                  ▷ evaluation
12:     *Tuple.5* $Opt_{\Delta:\mathbf{x}}$            ▷ update $\mathbf{x}$-relative step-size $\Delta$
13:     $t \leftarrow t + 1$
14: **end while**
```