Universiteit Leiden

Master Computer Science

[insert title]

Name:	[Huilin Li]
Student ID:	[s2556057]
Date:	[dd/mm/yyyy]
Specialisation:	[insert your master's track]
1st supervisor:	[name supervisor]
2nd supervisor:	[name supervisor]

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

since two levels primitive math

Diederick: is \dv{}
Anna: is \an{}
Carola: is \ca{}
Huilin: is \hl{}

Contents

List of Figures

List of Tables

Symbols

Table 1: Some symbols in this work.

Symbol	Meaning
$\mathcal{U}(a,b)$	A single random uniform distributed number in $[a,b)$
r	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 $\lambda = a$
Min$(\{a_i\}) \rightarrow a_i$	Minimization calculator that will generate one a_i that has the minimum fitness value amongst the set $\{a_i\}$.
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 $\{a_i\}$.
Round$(a) \rightarrow a^{'}$	Rounding calculator that will round a value a to its nearest integer $a^{'}$.
Dist$(a,b) \rightarrow D_{a,b}$	Distance calculator that will calculate the Euclidean distance by $D_{a,b} = \sqrt{(a-b)^2}$.
$\mid \ \mid$	Only denotes absolute value.

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 the nature and optimization methods attracts increasing attention.

The nature evolution process, as the most successful analogy [25], inspired most of the 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 researching 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** [10, 11, 22, 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 [10], 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 of these metaheuristic algorithms [11], 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 will propose a unified framework, with primitive math knowledge, for a set of swarm-based optimization algorithms. Before proposing this unified framework, there are two principles in this work:

- *Variants of algorithms aren't 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. This is because it is widely accepted that the base algorithm is the foundation of its variants, and if the foundation is suffering from issues, the variant built upon the foundation is also not solid. Moreover, theoretically speaking, any theory and any practice derived from the base can be applied to its variants, this is because the base and its variants share the same algorithm structure. Therefore, variants can be safely ignored here, and we will only focus on the base algorithms

- *Hybrid algorithms aren't considered in this work.*
  On the one side, independent studies should be the starting of the larger study area, on the other side, it must be accepted that it is very challenging to discuss independent and hybrid algorithms together. Therefore, hybrid algorithms will be also ignored here.

To build up such a unified framework, inspired by gene terminology and operations in the Evolutionary algorithm group (EA), this work will firstly build up a generic dictionary (like gene terminology and operations) in which **different expressions with the same information will be re-defined as a generic expression (see Chapter 2)**. After analyzing these algorithms on the level of components, the unified framework seems to come out by itself. The unified framework will be built upon the level of optimization progress that is **the algorithms' progresses are also able to be re-defined in a unified framework only with primitive math knowledge (see Chapter 3)**.

Therefore, for building a unified environment to challenge these chaotic new swarm-based algorithms, the work will firstly provide a generic language with which these chaotic algorithms can be discussed on the same component level in the Chapter 2. Next, the unified framework will be proposed which can simultaneously cover these chaotic algorithms in the Chapter 3. Furthermore, we will prove the feasibility of this unified framework by benchmark experiments in the Chapter 4 and Chapter 5. Lastly in the Chapter 6, we will conclude our findings and give future research directions.

1.2 Overview

As the Figure 1.1 displayed, since unifying the entire nature-inspired algorithms is impossible in one work, we will start from 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 of this framework is also able to be employed in much more other nature-inspired algorithms.

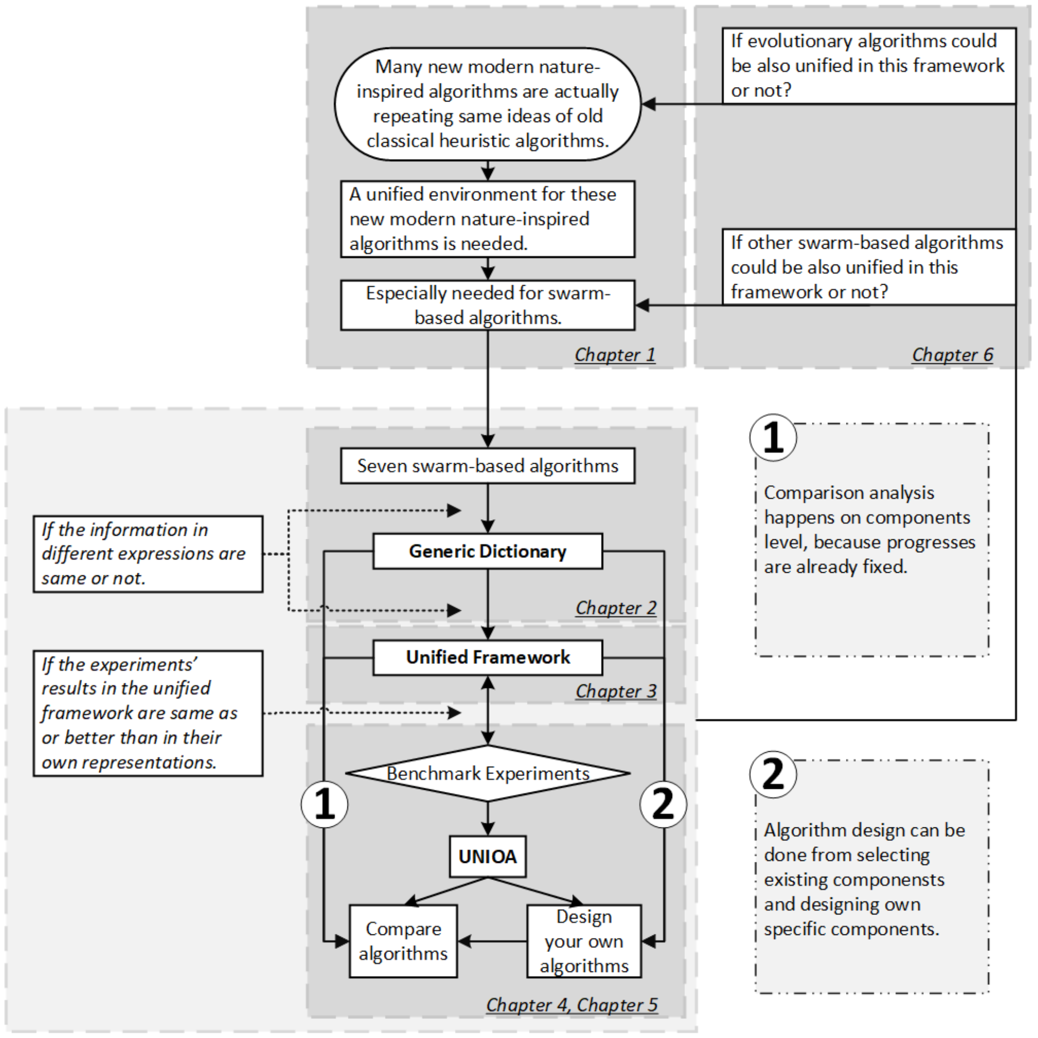
Many new modern nature-inspired algorithms are actually repeating same ideas of old classical heuristic algorithms.
A unified environment for these new modern nature-inspired algorithms is needed.
Especially needed for swarm-based algorithms.
Chapter 1
If evolutionary algorithms could be also unified in this framework or not?
If other swarm-based algorithms could be also unified in this framework or not?
Chapter 6
Seven swarm-based algorithms
Generic Dictionary
Chapter 2
Unified Framework
Chapter 3
Benchmark Experiments
UNIOA
Compare algorithms
Design your own algorithms
Chapter 4, Chapter 5
If the information in different expressions are same or not.
If the experiments' results in the unified framework are same as or better than in their own representations.
1
Comparison analysis happens on components level, because progresses are already fixed.
2
Algorithm design can be done from selecting existing componensts and designing own specific components.

Figure 1.1: Overview.

Meanwhile, 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 algorithm [3], Moth-flame optimization algorithm, Monarch butterfly optimization algorithm [28], Butterfly optimization algorithm [2].
- Old classical swarm-based algorithms:
  Particle swarm optimization [15]

This is because we hope the generic dictionary in the Chapter 2 and the unified framework in the Chapter 3 are unified enough to cover both traditional and modern nature-inspired algorithms.

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

Chapter 2

Generic Dictionary

2.1 Introduction

Before delving into these seven algorithms, we must clarify several prerequisites. The Table 2.1 lists all of them, and their detailed explanations are as follows. The first prerequisite is about some symbols commonly used in nature-inspired algorithms. In the nature-inspired algorithm researching field[1], the given objective optimization problem will be formulated as a math formula $f(\vec{x})$ in which each possible solution combination will be represented as a vector $\vec{x}$, moreover, the number of independent elements in $\vec{x}$ is normally defined as the searching space dimension to the objective optimization problem. In order to solve such an optimization problem, nature-inspired algorithms will randomly initialize a set of $\vec{x}_i$[2], then approach to the optimal solution by using iterative strategies. Same as in most studies, we also denote $\vec{x}$ as $\mathbf{x}$, and re-name its uniform name as the objective solution. Inspired by this hypothesis, we also define that any term in the bold style denotes the item is also a vector with the same dimension as the objective solution $\mathbf{x}$.

The second prerequisite is about the noun 'best'. In the papers we studied in this work, the 'best' $\mathbf{x}$ is always defined as the $\mathbf{x}$ that can reach the minimum fitness value.

The third prerequisite is the final algorithm model we will review in this work. We will not repeat the entire approach about how the natures are used to simulate mathematical

[1]Only on single objective continuous problems optimization.

[2]In nature-inspired algorithms, the set of $\vec{x}_i$ is called the entire population; the number of $\vec{x}_i$ in the set is called the size of the entire population; the value of $f(\vec{x})$ is called the fitness value of $\vec{x}$.

models. Such approaches can be found in their own original papers. We will present their final models by largely following their published implementation codes, in the meantime, the corresponding math representations and explanations will come from their published papers.

Table 2.1: Prerequisites when organizing the generic dictionary.

Prerequisite	Definition
$f(\cdot)$	The math programmed objective function to the actual optimization problem waiting to be solved.
$\mathbf{x}$	The vector $\vec{x}$ comprises all independent elements whose number is called the searching space dimension. $\mathbf{x}$ is called the objective solution in this work.
a or $\mathbf{a}$	Any non-bold term is a single number but any bold term is a vector with the same dimension as $\mathbf{x}$.
best	The best one is the $\mathbf{x}$ which can reach the minimum fitness.
final model	The published implemented codes largely determine the model reviewed in this work.

The Section 2.2 will review each algorithm: a step-by-step description about its final model, the difference between its published implemented code and its published paper, the default hyper-parameter settings in their published code or paper. The Section 2.3 will give and discuss our final generic dictionary based on these seven algorithms.

2.2 Specific Swarm-based Algorithms

2.2.1 Bat-inspired Algorithm (BA)

The Bat-inspired algorithm was inspired by the echolocation behaviour of micro-bats in 2010 [31]. The very first published paper [31] stated, for simplicity, they only considered the velocity $\mathbf{v}_i$ of each bat $\mathbf{x}_i$ and its echo frequency $freq$, the rate $rate$ of echo pulse emitted as well as the loudness A of echo. Their final algorithm model described here is strictly from their published implementation code[3], and explanations with math formulas are from their published papers [31].

Their final algorithm starts from is (1) initializing a bat population $\mathbf{x}_i$ with the size $n = 20$ in the d dimension environment and the initialization method in their published code is Eq.2.1 in which Lb/Ub is lower/upper boundary of each element in $\mathbf{x}_i$ and also (2) initializing everyone's velocity $\mathbf{v}_i$ as Eq.2.2.

$$\mathbf{x}_i = \boldsymbol{\mathcal{U}}(Lb, Ub) \tag{2.1}$$

$$\mathbf{v}_i = \boldsymbol{\mathcal{U}}(0, 0) \tag{2.2}$$

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) iteratively optimizing process in which the maximum number of iterations is $t\_max = 1000$. Next is (6) varying A as Eq.2.3 in which $A(t = 0) = 1$ and $\alpha = 0.97$ are respectively initial value and decreasing hyper-parameter. Meanwhile, (7) $rate$ is varying as Eq.2.4 in which $rate_0 = 1$ is the initial value and $\gamma = 0.1$ is the decreasing hyper-parameter. In this algorithm, t denotes the current iteration.

$$A(t+1) = \alpha \times A(t) \tag{2.3}$$

$$rate(t) = rate_0 \times (1 - e^{-\gamma \times t}) \tag{2.4}$$

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 $freq_i$.

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

Then (9) if $r < rate(t)$, $\mathbf{x}_i$ will be further updated as Eq.2.6 in which $\epsilon = 0.1$.

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

After (10) dealing with outliers in $\mathbf{x}_i(t+1)$ by replacing them by the boundary value, the BA model evaluates fitness of each $\mathbf{x}_i(t+1)$. Then (11) there is a so-called selection operator in which if the fitness of $\mathbf{x}_i(t+1)$ is better than of $\mathbf{x}_i(t)$ or $r > A$, $\mathbf{x}_i(t+1)$ will be accepted. Lastly (12) the $\mathbf{x}_*$ is updated.

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

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

2.2.2 Grasshopper Optimisation Algorithm (GOA)

The Grasshopper Optimisation Algorithm was proposed [24] in 2017. They designed this algorithm by studying grasshoppers' main swarming behaviors in both larval phase and adulthood phase, for example, slow movement and small steps in the larval phase, abrupt movement and long-range steps in the adulthood phase, as well as food source seeking in both phases. Their final model described here is from their published code[4], and explanations with math formulas are from their published paper [24].

The final 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 for $\mathbf{x}_i$.

$$\mathbf{x}_i = \boldsymbol{\mathcal{U}}(down, up) \tag{2.7}$$

As a side note, 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 4, we only consider the first case.

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 $l < max\_iteration = 100$[5]. Next is (4) updating a hyper-parameter c using Eq.2.8 in which $cMin = 0.00004/cMax = 1$ is the lower/upper boundary of c.

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

Then, the GOA model is (5) normalizing the distances between individuals into $[1,4]$. The normalization method in their published code is Eq.2.9 in which d is the Euclidean distance[6] between individual pairs.

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

The next step is (6) updating $\mathbf{x}_i$ as Eq.2.10 that is simplified version of their published original formula. Because we only consider 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 and their published paper didn't state if $\widehat{\ }$ is a special operation or not.

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

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

[5] 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.

[6] This refers to their faster version code.

Moreover, the optimization method Eq.2.10 has a special factor named S function 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 pre-setup hyper-parameters[7].

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

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 is finished, and the steps from step(4) to step(9) will be iteratively executed until the stop condition meets.

2.2.3 Crow Search Algorithm (CSA)

The crow search algorithm was designed [3] in 2016. The core metaphor is crows' behaviour of hiding excess food and retrieving this stored food when needed. 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 [3]. Their final algorithm model described here is from their published implementation code[8], and explanations with math formulas are from their published paper [3].

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 reframed as Eq.2.12 in which l/u is the lower/upper boundary.

$$\mathbf{x}_i = \boldsymbol{\mathcal{U}}(l,u) \tag{2.12}$$

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 $tmax = 5000$. Next, (4) each individual $\mathbf{x}_i$ is updated

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

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

as Eq.2.14 in which $fl = 2$ and $AP = 0.1$ are two pre-setted hyper-parameters.

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

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 is finished, and the steps from step(4) to step(7) will be iteratively executed until the stop condition meets.

2.2.4 Moth-flame Optimization Algorithm (MFO)

In 2015, the Moth-flame optimization algorithm simulated the process that moths will fly to the centre of artificial light (=flames) in the spiral path [20]. The important characteristic in MFO is that the author defined flames' positions as moths' optimization orientation, and each moth has its own dynamically changing flame. Their final algorithm model described here is strictly from their published implementation code[9], meanwhile, explanations with math formulas are from their published paper [20].

The final algorithm model starts from (1) $N = 30$ initial moths population $Moth\_pos$ in the dim dimension environment, and 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[10].

$$\mathbf{m}_i = \boldsymbol{\mathcal{U}}(lb, ub) \tag{2.16}$$

After (2) evaluating the fitness of each $\mathbf{m}_i$, the algorithm model starts iteratively optimizing each $\mathbf{m}_i$ in which the stop condition is there is a maximum number of iterations T. First

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

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

of all, (3) a threshold named $Flame\_no$ is calculated by Eq.2.17 in which t is the current number of iterations. $Flame\_no$ plays a crucial role in the following steps.

$$flame\_no = round(N - t \times \frac{N-1}{T}) \tag{2.17}$$

Next is (4) sorting the combination of the current population and the previous population[11] with respect to their fitness, and defining the first $N = 30$ moths in the combination population as $sorted\_population$. Then, a linearly decreasing parameter tt[12] is calculated by Eq.2.18.

$$\begin{aligned} a &= -1 + t \times \frac{-1}{T} \\ tt &= (a-1) \times r + 1 \end{aligned} \tag{2.18}$$

Next, (5) each element j of each moth i will be updated as follows: if $i \leqslant flame\_no$, Eq.2.19; else, Eq.2.20 in which $\mathbf{m}'_{i,j}$ denotes each updated each element j of each moth $\mathbf{m}_i$; $b = 1$ is a constant parameter used to define the shape of spiral [20].

$$\begin{aligned} d\_to\_flame &= |sorted\_population(i,j) - \mathbf{m}_{i,j}| \\ \mathbf{m}'_{i,j} &= d\_to\_flame \times e^{b \times tt} \times \cos(tt \times 2 \times \pi) + sorted\_population(i,j) \end{aligned} \tag{2.19}$$

$$\begin{aligned} d\_to\_flame &= |sorted\_population(i,j) - \mathbf{m}_{i,j}| \\ \mathbf{m}'_{i,j} &= d\_to\_flame \times e^{b \times tt} \times \cos(tt \times 2 \times \pi) + sorted\_population(flame\_no,j) \end{aligned} \tag{2.20}$$

After updating each $\mathbf{x}_i$, (6) outliers in $\mathbf{x}_i$ are replaced by the boundary values before (7) calculating its fitness. Until now, one optimization round is finished, and the steps from step(3) to step(7) will be iteratively executed until the stop condition meets.

2.2.5 Monarch Butterfly Optimization (MBO)

The Monarch Butterfly Optimization simulated the migration behaviour of monarch butterflies in 2015 [28]. The core metaphor is that monarch butterflies in two different habitats evolve in two different ways. These two different habitats are mimicked by dividing the whole population into two subpopulations.The author defined the whole population will firstly be sorted with respect to their fitness before dividing them into

[11]The previous population could be $\emptyset$, when the current population is the initial population.

[12]Their published paper used notation t again, but we used notation tt instead of t to avoid confusion with current iteration t in the Eq.2.17.

two with a ratio [28]. Then, each habitat experiences its own special evolution approach (migration operator or adjusting operator [28]). We will describe their final algorithm model implemented in their published code[13], and explanations with math expressions from their published paper [28].

Their final algorithm starts from (1) an initial population $Population$ with size $M = 50$, and the initialization method[14] in their published code can be formulated as Eq.2.21 in which $MinParValue/MaxParValue$ is the lower/upper boundary of elements in $\mathbf{x}_i$.

$$\mathbf{x}_i = \boldsymbol{\mathcal{U}}(MinParValue, MaxParValue) \tag{2.21}$$

After (2) evaluating each $\mathbf{x}_i$, it starts iteratively optimizing each $\mathbf{x}_i$ in which the stop condition is the maximum number of iterations $Maxgen = 50$. Then (3) $Keep = 2$ best $\mathbf{x}_i$ are kept in $chromKeep$[15]. Next is (4) dividing the entire $Population$ into 2 sub-population with ratio $partition = \frac{5}{12}$. The 2 sub-populations $Population1/Population2$ will be updated by two 'Migration operator'/'Adjusting operator' respectively. In the $Population1$, (5) if $r \times period \leqslant partition$ 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 kth element and $population1/population2$ is one random moth from $Population1/Population2$.

In $Population2$, (6) if $r \leqslant partition$, $\mathbf{x}_{i,k}^{t+1} = \mathbf{x}_{best,k}^{t}$ in which k is the kth 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 kth element and $population2$ is one random moth from $Population2$, however, under this situation, if $r > BAR$ 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 $\mathrm{d}x$ is a walk step as Eq.2.24.

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

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

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

where Levy function in their published code can be formulated as Eq.2.25.

$$\begin{gathered} Stepsize \sim Exp(2 \times Maxgen) \\ \mathrm{d}x_k = \sum^{Stepsize} \tan(\pi \times r) \end{gathered} \tag{2.25}$$

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

[14] 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

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

Lastly, after (7) updating each subpopulation and combining them into one entire population, meanwhile, the outliers will be replaced by the boundary value (8) there is a so-called selection process in which the last $Keep = 2$ worst $\mathbf{x}_i$ will be replaced by $chromKeep$.

Until now, one optimization round is finished, and the steps from step(3) to step(8) will be iteratively executed until the stop condition meets.

2.2.6 Butterfly Optimization Algorithm (BOA)

The Butterfly Optimization Algorithm was modelled in 2018 [2]. The authors stated that they mimicked butterflies' behaviour of sensing/analyzing smell in the air to seek food/mates. We will step-by-step describe their final pseudocode with their final mathematical formulas in their published paper [2] and published code[16].

In their published pseudocode, they defined a dim dimensional environment in which each individual is $\mathbf{x}_i$ and the fitness of $\mathbf{x}_i$ is $f(\mathbf{x}_i)$. Firstly (1) they initialized a population of n $\mathbf{x}_i$ by Eq.2.26 in which Lb/Ub is lower/upper bound[17].

$$\mathbf{x}_i = \boldsymbol{\mathcal{U}}(Lb, Ub) \tag{2.26}$$

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

Next steps are the iterative optimization process in which the stop condition is $t < N\_iter$. Then for each $\mathbf{x}_i$, (4) it calculated its own fragrance that is related to its own fitness value as Eq.2.27.

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

Then (5) if a single uniformly distributed random value $rand > p$, 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 random neighbors around $\mathbf{x}_i$.

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

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

[17] The initialization method wasn't shown in their published paper/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.

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

Then before (7) replacing the outliers by the boundary values, (8) only if the updated $\mathbf{x}_i$ is better than the previous one according to their fitness, the update $\mathbf{x}_i$ will be accepted. Next steps are (9) updating g^* and (10) updating $sensory\_modality$ using Eq.2.30.

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

Until now, one optimization round is finished, and the steps from step(4) to step(8) will be iteratively executed until the stop condition meets.

2.2.7 Particle Swarm Optimization (PSO)

Particle Swarm Optimization was firstly introduced in 1995 [15]. The main idea comes from the movements of the animal swarm. By simulating their behaviour that the swarm dynamically moves to a 'roost' [15], the extremely simple algorithm, even with a small swarm size (15 to 30 agents), boasted impressive performance on continuous optimization functions. The very first paper defined the position of each agent in its swarm was controlled by two velocities $\mathbf{X}$ and $\mathbf{Y}$, moreover, each agent can remember its own best position $pbest$ and know the global best position $gbest$ in their swarm. Although the very first paper also provided the algorithm model with formulas, we preferred to reference a clear published pseudo-code whose algorithm model is as same as the very first paper possible. The final PSO model utilized in this work is from a tutorial of this method [19] and its setups are from empirical experiences. **Huilin: the pseudo-code is from the lecture slide actually, so, how to state this, that this model is stated by following the model in one lecture slide?**

This published pseudocode started from (1) an initial swarm $\mathbf{x}_i$ with the size $N = 25$. The initialize method was not detailed in the published paper, however, it is widely acceptable that the method can be mathematically formulated as Eq.2.31 in which lb/ub is the lower/upper boundary of $\mathbf{x}_i$.

$$\mathbf{x}_i = \boldsymbol{\mathcal{U}}(lb, ub) \qquad 2.31$$

Moreover, (2) each velocity $\mathbf{v}_i$ is also initialized by Eq.2.32 in which lb_v/ub_v is the boundary of velocity and (3) each $pbest$ is initialized by Eq.2.33.

$$\mathbf{v}_i = \boldsymbol{\mathcal{U}}(lb_v, ub_v) \qquad 2.32$$

$$pbest_i = \mathbf{x}_i \qquad 2.33$$

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. The published pseudo-code next 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 [19].

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

After updating (7) each agent by 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) $gbest$.

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

Until now, one optimization round is finished, and the steps from step(6) to step(9) will be iteratively executed until the stop condition meets. As a side note, the very first published paper did not directly point out the notation for searching 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 also needs a dimension concept. Meanwhile, the stop condition and the method to deal with the outliers also must exist, and the method to correct outliers is the most common method in which outliers will be corrected by the boundary value.

2.3 Generic Dictionary

2.3.1 Conclusion

According to the information carried by these different representations, the entire information that appears in these seven algorithms is able to be categorized into 20 different detailed components. The Table 2.2 lists these 20 generic components with notations we defined. Meanwhile, as the Figure 2.1 displayed, these 20 detailed components are able to be further categorized into two groups: **compulsory components** and **selective components**, which means **each of these seven algorithms is able to be made up of the entire compulsory components and some of the selective components**.

To the information that is compulsory to complete a complete optimization model, first of all, 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 are extra information carried by the $f()$. (see *Information 1, 3, 4*).

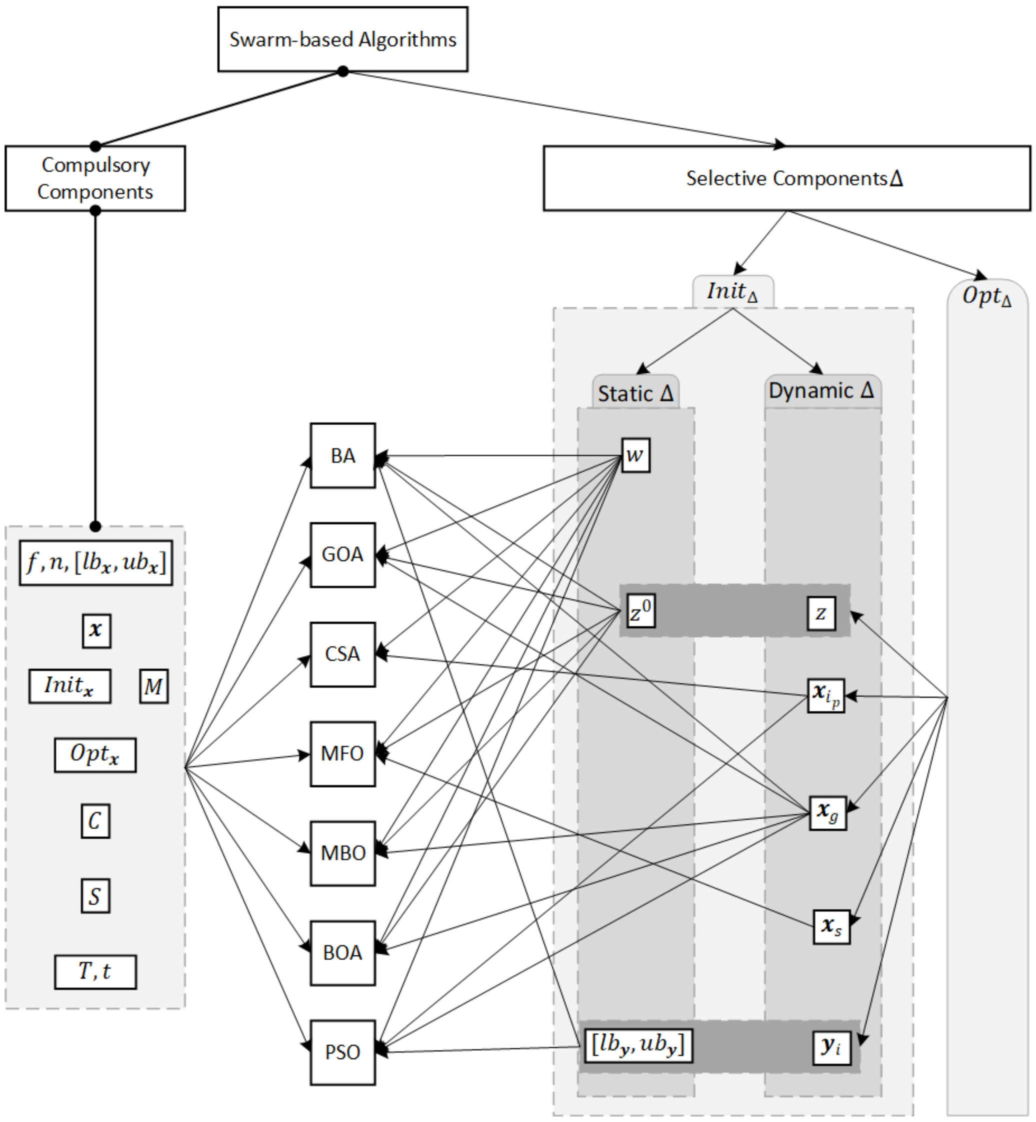

Figure 2.1: What components exist in these seven algorithms.

Next, 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 searching 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. (see *Information 2, 5, 13, 15*).

Moreover, 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 [6], the method S related to sampling and the stop condition T must exist. Moreover, these algorithms will solve the problem in a limited searching space, there must also be a method to deal with outliers outside the searching space. (see *Information 17, 18, 19, 20*).

Therefore, until now, we conclude that $f()$, $\mathbf{x}$, M, T and methods $Init_{\mathbf{x}}$, $Opt_{\mathbf{x}}$, C, S must exist in a swarm-based optimization algorithm.

Besides this compulsory information that appears in all of these seven algorithms, several information only appears in some of these seven algorithms, **for this kind of information that is selective to help achieve a higher quality of optimization, we name it as the influencing factor** Δ in this work. (see *Information 6-12*).

After delving into these influencing factors, we concluded there are two types of influencing factors as follows:

(1) Static influencing factor. See *Information 11*.
Most static numerical influencing factors w are able to be understood as common hyper-parameters in most algorithms. They are numbers, are set before running algorithms and are unchanged when approaching algorithms.

(2) Dynamic influencing factor. See *Information 6, 7, 8, 9, 12*.

  (a) Dynamic numerical influencing factor. See *Information 12*.
  The dynamic numerical influencing factor z will be changing over the iteration t during the optimization process.

  (b) Dynamic vector influencing factor.
  As its name indicates, this kind of influencing factor is a vector with the same dimension as $\mathbf{x}$.

    i. $\mathbf{x}$-relative vector influencing factor. See *Information 6, 7, 8*.

    ii. assisting vector influencing factor. See *Information 9*.

Because the dynamic influencing factors are changing during the optimization process, there must be an initialization status and a changing process. Therefore, we also defined $Init_{\Delta}$ and Opt_{Δ} to achieve the initialization status and the changing process for dynamic influencing factors, in which the $Init_{\Delta}$ method can also include the set-up of static influencing factor.

Lastly, after summarizing these similarities on the component level, the next Chapter 3 will display their similarities on the optimization progress level.

2.3.2 Generic Dictionary in Any Other Swarm-based Algorithms.

This section will display the possibility of applying this generic dictionary for any other swarm-based algorithms. First of all, the *Information 1-5, 7, 11-13, 15, 17-20* are easily detected in any one swarm-based algorithms, 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*s.

The place where it is most likely to confuse users is **how to detect the dynamic vector influencing factors: Why we think one vector influencing factor is a $\mathbf{x}$-relative vector not an assisting vector $\mathbf{y}$?** This question will 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). Could we directly assign them the assisting vector influencing factor? The answer is nope, although it is the most intuitive way. **The reason is that the information carried by these nouns is the main principle used to assign them different component name.** For example:

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

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

- 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 $\langle flame_i \rangle(t+1) = \textbf{Sort}(\{\mathbf{x}_i(t)\} \cup$

$\{\mathbf{x}_i(t+1)\}), i = 1 \ldots M$. 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 $\mathbf{x}$ all the time, rather than the assisting vector $\mathbf{y}_i$.

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

- 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 $\mathbf{x}$.

Therefore, the *Information 6, 8, 9, 10, 14, 16* might also be most likely clearly detected in any other swarm-based algorithms according to the previous observations.

Table 2.2: Generic dictionary: all information appearing in these seven algorithms is able to be categorized into 20 components.

Information	Different Representation	Generic Notation
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 $f()$ searching space.
4.The boundary of each independent element in $\mathbf{x}_i$[18].	•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 element 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 dynamic influencing factor 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 dynamic influencing factor Δ.

[18]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.Some special vector influencing factors are 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 dynamic influencing factor Δ.
9.Some special vector influencing factors are 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 $\mathbf{v}_i$.	$\mathbf{y}_i$: assisting influencing factor Δ.
10.The boundary of each independent 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 $\mathbf{y}_i$.
11.Influencing factors that are 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: influencing factor Δ.
12.Influencing factors 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: influencing factor 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 dynamic vector influencing factors.	•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 dynamic influencing factors.	•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. •MBO: Eq.2.23,step(5). •BOA: step(3),Eq.2.30. •PSO: Eq.2.3.•PSO: Eq.2.35,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(6). •MBO: step(7). •BOA: step(7). •PSO: common method.	C: clip outliers in $\mathbf{x}_i$.
18.How to decide if the new generated $\mathbf{x}_i$ would be accepted.	•BA: step(11). •GOA: none. •CSA: none. •MFO: none. •MBO: step(8). •BOA: step(8). •PSO: none.	S: selection method on new generated population.
19.The current iteration.	•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: the budget.

Chapter 3

Unified Framework

3.1 Unified Nature-Inspired Optimization Algorithm

The Table 2.2 displayed 20 components are sufficient to make up any one of these seven algorithms. Therefore, we unified these seven algorithms into the UNIOA framework whose math representation is Eq.3.1 and pseudo-code is Algorithm.1. The Section 3.1 will only give very intuitive findings in these seven algorithms based on the UNIOA framework. Later the Section 3.3 will conclude the UNIOA framework with more details after detailing how to re-frame each algorithm in the Section 3.2.

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

where

(1) Objective problem: f.
f represents the math programmed actual optimization problem. f also provides the information about the searching space, such as the dimension n, the constraint $[lb_{\mathbf{x}}, ub_{\mathbf{x}}]$. It must exist.

(2) Objective solution: $\mathbf{x}$.
$\mathbf{x}$ is one possible optimal solution obtained from the algorithm. It must exist.

(3) Population size: M.
The population size M is a very important component in nature-inspired algorithms [23]. It must exist.

(4) Objective solution initialization method: $Init_{\mathbf{x}}$.
The Eq.3.2 is the first step in all swarm-based algorithms. It must exist.

$$\mathbf{x}_i(t=0) = \boldsymbol{\mathcal{U}}(lb_{\mathbf{x}}, ub_{\mathbf{x}}), i = 1, 2, \ldots, M \qquad 3.2$$

(5) Objective solution optimization method: $Opt_{\mathbf{x}}$.
The optimization method is an important feature that determines the optimization ability of swarm-based algorithms. It is different in different algorithms. It must exist.

(6) Dealing with outliters method: C.
In these seven algorithms, there are two methods to deal with outliers in $\mathbf{x}_i$ after obtaining new generated $\hat{\mathbf{x}}_i$[1]. The Eq.3.3 is commonly used in most algorithms, for example, except the CSA model, other algorithms in this work all use the Eq.3.3 to deal with outliers. In this work, the CSA model especially uses the Eq.3.4. It must exist[2].

$$\mathbf{x}_{i,n}^{\text{fixed}}(t+1) = \begin{cases} ub_x & , \mathbf{x}_{i,n}(t+1) > ub_{\mathbf{x}} \\ \mathbf{x}_{i,n}(t+1) & , \text{o.w} \\ lb_x & , \mathbf{x}_{i,n}(t+1) < lb_{\mathbf{x}} \end{cases} \tag{3.3}$$

$$\mathbf{x}_{i,n}^{\text{fixed}}(t+1) = \begin{cases} \mathbf{x}_{i,n}(t) & , \mathbf{x}_{i,n}(t+1) < lb_{\mathbf{x}} \text{ or } \mathbf{x}_{i,n}(t+1) > ub_{\mathbf{x}} \\ \mathbf{x}_{i,n}(t+1) & , \text{o.w} \end{cases} \tag{3.4}$$

(7) Stop condition: T.
The T determines if the iterative optimization process would stop or not. It must exist.

(8) Selection method: S.
The method Eq.3.5 determines if the new generated $\hat{\mathbf{x}}_i(t+1)$ would be accepted and would go to the next optimization round. This method differs from different special conditions. It might exist. As a side note, some algorithms, such as MBO, might use very specific method to select from the new generated population, therefore, such methods might not be able to be involved in the Eq.3.5.

$$\mathbf{x}_i(t+1) = \begin{cases} \hat{\mathbf{x}}_i(t+1), & \text{special conditions} \\ \mathbf{x}_i(t), & \text{o.w} \end{cases} \tag{3.5}$$

(9) Influencing factors initialization method: $Init_{\Delta}$.
Influencing factors Δ are selective factors that play a crucial role in $Opt_{\mathbf{x}}$. There are three types of influencing factors as we studied in the Table 2.2 and they might exist or not.

[1]In this work, $\hat{\mathbf{x}}(t+1)$ is the new solutions generated before Selection S. However, $\mathbf{x}(t+1)$ is the new solutions generated after Selection S, which is also the final new updated solutions after one complete optimizarion round.

[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.

(a) Static numerical influencing factor: $w, z^0, [lb_{\mathbf{y}}, ub_{\mathbf{y}}]$.
This kind of influencing factor is fixed after setting-up at the beginning. 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 dynamic numerical influencing factors and the assisting vector influencing factor.

(b) Dynamic influencing factor: z, $\mathbf{x}_{i_p}$, $\mathbf{x}_g$, $\mathbf{x}_s$, $\mathbf{y}_i$.

i. Dynamic numerical influencing factor: z.
The z is a kind of hyper-parameter that will be changing as the iteration t during the optimization process. Its initialization method is varying.

ii. Dynamic vector influencing factor: $\mathbf{x}_{i_p}$, $\mathbf{x}_g$, $\mathbf{x}_s$, $\mathbf{y}_i$.

A. $\mathbf{x}$-relative vector influencing factor: $\mathbf{x}_{i_p}$, $\mathbf{x}_g$, $\mathbf{x}_s$.

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

$$\mathbf{x}_{i_p}(t=0) = \mathbf{x}_i(t=0), i = 1 \dots M \qquad 3.6$$

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

$$\mathbf{x}_g(t=0) = \textbf{Min}(\{\mathbf{x}_i(t=0)\}), i = 1 \dots M \qquad 3.7$$

- the $\mathbf{x}_s$ denotes one kind of special vector influencing factor that has a strong connection with the $\mathbf{x}$. It depends on different algorithms, so it is varying.

B. assisting vector influencing factor: $\mathbf{y}_i$.
In this work, there is only one kind of initialization method as Eq.3.8. In this work, its initialization method is fixed, however, it could be customized in the future.

$$\mathbf{y}_i(t=0) = \boldsymbol{\mathcal{U}}(lb_{\mathbf{y}}, ub_{\mathbf{y}}), i = 1 \dots M \qquad 3.8$$

(10) Influencing factors optimization method: Opt_{Δ}.
As mentioned above, dynamic influencing factor will be changing during the optimization process, therefore, the Opt_{Δ} is the method to determine how they are changing during the optimization process. Mostly, it differs in different algorithms,

however, for $\mathbf{x}_{i_p}$ and $\mathbf{x}_g$ whose initialization method is fixed, their optimization method is also fixed respectively as Eq.3.8 and Eq.3.10. It might exist or not.

$$\mathbf{x}_{i_p}(t+1) = \textbf{Min}(\{\mathbf{x}_{i_p}(t), \mathbf{x}_i(t+1)\}) \tag{3.9}$$

$$\mathbf{x}_g(t+1) = \textbf{Min}(\mathbf{x}_g(t) \cup \{\mathbf{x}_i(t+1)\}), i = 1 \dots M \tag{3.10}$$

Algorithm 1 Generic Nature-Inspired Optimization Algorithm

```
1:  $t \leftarrow 0$
2:  $Init_{\mathbf{x}}, M$                                          ▷ Initialization Process
3:  $f$                                                            ▷ Evaluation
4:  $Init_{\Delta}$: $\mathbf{y}_i, \mathbf{x}_{i_p}, \mathbf{x}_g, \mathbf{x}_s, z, w$
5:  while termination criteria are not met do
6:      $\{Opt_{\Delta} \rightarrow \mathbf{y}(t+1)\ ,\ \emptyset\}$           ▷ Optimization Process
7:      $Opt_{\mathbf{x}}, C \rightarrow \hat{\mathbf{x}}_i(t+1)$
8:      $f(\hat{\mathbf{x}}_i(t+1))$                                   ▷ Evaluation
9:      $S \rightarrow \mathbf{x}_i(t+1)$                                ▷ Selection
10:     $t \leftarrow t+1$
11:     $\{Opt_{\Delta} \rightarrow (\mathbf{x}_{i_p}(t+1), \mathbf{x}_g(t+1), \mathbf{x}_s, z(t+1)), \emptyset\}$
12: end while
```

The math representation Eq.3.1 displays **what components are used to construct these seven algorithms**, then, the pseudo-code Algorithm.1 will display **how these components are constructed to build up these seven algorithms**, which is in the view of algorithms' optimization progress. As the Algorithm.1 described, every algorithm will start from the initialization process in which the population must be evaluated after being initialized, moreover, different algorithms will utilize $Init_{\Delta}$ differently. In the optimization process, before meeting the stop condition, some algorithms might firstly optimize the assisting vector influencing factor $\mathbf{y}_i$ before updating $\mathbf{x}_i$. After dealing with outliers in the new generated $\hat{\mathbf{x}}_i(t+1)$, $\hat{\mathbf{x}}_i(t+1)$ must be evaluated again before deciding if the new generated $\hat{\mathbf{x}}_i(t+1)$ will be accepted by using the selection method. Lastly before moving to the next optimization round, other dynamic influencing factors will be updated in some algorithms.

3.2 Re-framed Nature-inspired Algorithms

3.2.1 Re-framed BA

As discussed in the Table 2.2, the BA model utilized three kinds of dynamic influencing factors and several static influencing factors to achieve the optimization on the objective solutions. These three dynamic influencing factors are Δ: $\mathbf{y}_i$, $\mathbf{x}_g$ and z.

The re-framed BA model, in the initialization process, will specially initialize $\mathbf{y}_i$ as Eq.3.11, $\mathbf{x}_g$ as Eq.3.7, z_1 as Eq.3.12, z_2 as Eq.3.13. The static influencing factors will be also set at this moment.

$$\mathbf{y}_i(t=0) = \boldsymbol{\mathcal{U}}(lb_{\mathbf{y}}, ub_{\mathbf{y}}), i = 1 \ldots M \qquad 3.11$$

$$z_1(t=0) = z_1^0 \times (1 - e^{-w_1 \times t}) \qquad 3.12$$

$$z_2(t=0) = z_2^0 \times w_2 \qquad 3.13$$

where $[lb_{\mathbf{y}}, ub_{\mathbf{y}}] = [0,0]$, $z_1^0 = 1$, $z_2^0 = 1$, $w_1 = 0.1$ and $w_2 = 0.97$.

Next in the optimization process, before the stop condition T meets, the re-framed BA model will firstly generate new $\mathbf{y}_i(t+1)$ as Eq.3.14, then generate $\hat{\mathbf{x}}_i(t+1)$ as Eq.3.15 with the help of dealing with outliers C as Eq.3.3.

$$\mathbf{y_i}(t+1) = \mathbf{y_i}(t) + \mathcal{U}(lb_{w_4}, ub_{w_4}) \times (\mathbf{x}_i(t) - \mathbf{x}_g(t)) \qquad 3.14$$

$$\hat{\mathbf{x}}_i(t+1) = \begin{cases} \mathbf{x}_g(t) + w_3 \times \boldsymbol{\mathcal{N}}(\mathbf{0},\mathbf{1}) \times z_2(t), & r < z_1(t) \\ \mathbf{x}_i(t) + \mathbf{y}_i(t+1), & \text{o.w} \end{cases} \qquad 3.15$$

where $[lb^{w_4}, ub^{w_4}] = [0,2]$,$w_3 = 0.1$.

Lastly after evaluating the fitness of $\hat{\mathbf{x}}_i(t+1)$, the special condition used in the selection method is $r > z_2 \cap f(\hat{\mathbf{x}}_i(t+1)) < f(\mathbf{x}_i(t))$. At this moment, the re-framed BA model has obtained the final generated new solutions after one round optimization. And finally other dynamic influencing factors ($\mathbf{x}_g$, z_1, z_2) are going to be updated ($\mathbf{x}_g$ as Eq.3.10, z_1 as Eq.3.16, z_2 as Eq.3.17) before the next optimization round.

$$z_1(t+1) = z_1^0 \times (1 - e^{-w_1 \times t}) \qquad 3.16$$

$$z_2(t+1) = z_2(t) \times w_2 \qquad 3.17$$

Till now, all symbols that are identified after the 'where' statement are involved in the static influencing factors.

Algorithm 2 Bat Algorithm

```
 1: $t \leftarrow 0$
 2: $Init_{\mathbf{x}}, M$                                    ▷ Initialization Process
 3: $f$                                                       ▷ Evaluation
 4: $Init_{\Delta} : \mathbf{y}_i, \mathbf{x}_g, z_1, z_2,$
 5: while termination criteria are not met do
 6:     $Opt_{\Delta} \rightarrow \mathbf{y}_i(t+1)$                  ▷ Optimization Process
 7:     $Opt_{\mathbf{x}}, C \rightarrow \hat{\mathbf{x}}_i(t+1)$
 8:     $f(\hat{\mathbf{x}}_i(t+1))$                              ▷ Evaluation
 9:     $S \rightarrow \mathbf{x}_i(t+1)$                             ▷ Selection
10:     $t \leftarrow t+1$
11:     $Opt_{\Delta} \rightarrow \mathbf{x}_g(t+1), z_1(t+1), z_2(t+1)$
12: end while
```

3.2.2 Re-framed GOA

As discussed in the Table 2.2, the GOA model utilized two kinds of dynamic influencing factors and several static influencing factors to achieve the optimization on the objective solutions. These two dynamic influencing factors are Δ: $\mathbf{x}_g$ and z.

The re-framed GOA model, in the initialization process, will specially initialize $\mathbf{x}_g$ as Eq.3.7, z as Eq.3.18. The static influencing factors will also be set at this moment.

$$z(t=0) = ub^z - t \times (\frac{ub^z - lb^z}{T}) \tag{3.18}$$

Next in the optimization process, before the stop condition T meets, the re-framed GOA model will firstly directly generate $\hat{\mathbf{x}}_i(t+1)$ as Eq.3.19 with the help of dealing with outliers C as Eq.3.3.

$$\begin{aligned} &\widetilde{D}_{i,j}(t) = 2 + \textbf{Dist}(\mathbf{x}_i(t), \mathbf{x}_j(t)) \, \textbf{mod} \, 2 \\ &\hat{\mathbf{x}}_i(t+1) = z(t) \times (\sum_{j=1, j\neq i}^{M} z(t) \times \frac{ub_{\mathbf{x}} - lb_{\mathbf{x}}}{2} \times (w_1 \times e^{\frac{-\widetilde{D}_{i,j}(t)}{w_2}} - e^{-\widetilde{D}_{i,j}(t)}) \times \frac{\mathbf{x}_i(t) - \mathbf{x}_j(t)}{\textbf{Dist}(\mathbf{x}_i(t), \mathbf{x}_j(t))}) + \mathbf{x}_g \end{aligned} \tag{3.19}$$

where $[ub^z, lb^z] = [0.00004, 1], w_1 = 0.5, w_2 = 1.5$.

Lastly after evaluating the fitness of $\hat{\mathbf{x}}_i(t+1)$, the special condition used in the selection method is $\forall$. At this moment, the re-framed GOA model has obtained the final generated new solutions after one round optimization. And finally other dynamic influencing factors

($\mathbf{x}_g$, z) are going to be updated ($\mathbf{x}_g$ as Eq.3.10, z as Eq.3.20) before starting the next optimization round.

$$z(t+1) = ub_z - (t+1) \times (\frac{ub_z - lb_z}{T}) \tag{3.20}$$

Till now, all symbols that are identified after the 'where' state are involved in the static influencing factors.

Algorithm 3 Grasshopper Optimization Algorithm

```
1: $t \leftarrow 0$
2: $Init_{\mathbf{x}}, M$                                  ▷ Initialization Process
3: $f$                                                   ▷ Evaluation
4: $Init_{\Delta}$: $\mathbf{x}_g, z$
5: while termination criteria are not met do
6:     $Opt_{\mathbf{x}}, C \rightarrow \hat{\mathbf{x}}_i(t+1)$            ▷ Optimization Process
7:     $f(\hat{\mathbf{x}}_i(t+1))$                          ▷ Evaluation
8:     $S \rightarrow \mathbf{x}_i(t+1)$                         ▷ Selection
9:     $t \leftarrow t+1$
10:    $Opt_{\Delta} \rightarrow \mathbf{x}_g(t+1), z(t+1)$
11: end while
```

3.2.3 Re-framed CSA

As discussed in the Table 2.2, the CSA model utilized one kind of dynamic influencing factors and several static influencing factors to achieve the optimization on the objective solutions. The only one dynamic influencing factor is $\mathbf{x}_{i_p}$.

The re-framed CSA model, in the initialization process, will specially initialize $\mathbf{x}_{i_p}$ as Eq.3.6. The static influencing factors will be also set at this moment.

Next in the optimization process, before the stop condition T meets, the re-framed CSA model will directly firstly generate $\hat{\mathbf{x}}_i(t+1)$ as Eq.3.21 with the help of dealing with outliers C as Eq.3.4.

$$\hat{\mathbf{x}}_i(t+1) = \begin{cases} \mathbf{x}_i(t) + rand \times w_2 \times (\mathbf{x}_{j_p}(t) - \mathbf{x}_i(t)) & , \quad r > w_1 \\ \boldsymbol{\mathcal{U}}(lb_{\mathbf{x}}, ub_{\mathbf{x}}) & , \quad \text{o.w} \end{cases} \tag{3.21}$$

where $w_1 = 0.1$, $w_2 = 2$. Moreover, the $\mathbf{x}_{j_p}$ is any one neighbor around the $\mathbf{x}_{i_p}$.

Lastly after evaluating the fitness of $\hat{\mathbf{x}}_i(t+1)$, the special condition used in the selection method is $f(\hat{\mathbf{x}}_i(t+1)) < f(\mathbf{x}_i(t))$. At this moment, the re-framed CSA model has obtained the final generated new solutions after one round optimization. And finally the dynamic influencing factor is going to be updated before starting the next optimization round, in which the update method to $\mathbf{x}_{i_p}$ is as Eq.3.9.

Till now, all symbols that are identified after the 'where' statement are involved in the static influencing factors.

Algorithm 4 Crow Search Algorithm

```
 1: $t \leftarrow 0$
 2: $Init_{\mathbf{x}}, M$                              ▷ Initialization Process
 3: $f$                                               ▷ Evaluation
 4: $Init_{\Delta}$: $\mathbf{x}_{i_p}$
 5: while termination criteria are not met do
 6:     $Opt_{\mathbf{x}}, C \rightarrow \hat{\mathbf{x}}_i(t+1)$          ▷ Optimization Process
 7:     $f(\hat{\mathbf{x}}_i(t+1))$                          ▷ Evaluation
 8:     $S \rightarrow \mathbf{x}_i(t+1)$                         ▷ Selection
 9:     $t \leftarrow t+1$
10:     $Opt_{\Delta} \rightarrow \mathbf{x}_{i_p}(t+1)$
11: end while
```

3.2.4 Re-framed MFO

As discussed in the Table 2.2, the MFO model utilized two kinds of dynamic influencing factors and several static influencing factors to achieve the optimization on the objective solutions. These two dynamic influencing factors are Δ: $\mathbf{x}_s$, z.

The re-framed MFO model, in the initialization process, will specially initialize $\mathbf{x}_s$ as Eq.3.22, z_{1_i} as Eq.3.23, z_2 as Eq.3.24. The static influencing factors will be also set at this moment.

$$\langle \mathbf{x}_i(t=0) \rangle = \textbf{Sort}(\{\mathbf{x}_i(t=0)\}), i = 1 \dots M \tag{3.22}$$

$$z_{1_i}(t=0) = r \times (-2 - \frac{t}{T}) + 1 \tag{3.23}$$

$$z_2(t=0) = \textbf{Round}(M - t \times \frac{M-1}{T}) \tag{3.24}$$

Next in the optimization, before the stop condition T meets, the re-framed MFO model will directly generate the $\hat{\mathbf{x}}_i$ as Eq.3.25 with the help of dealing with outliers C with the help of dealing with outliers C as Eq.3.3.

$$\hat{\mathbf{x}}_i(t+1) = \begin{cases} (\mathbf{x}_{s_i}(t) - \mathbf{x}_i(t)) \times e^{w \times z_{1_i}(t)} \times \cos(2\pi \times z_{1_i}(t)) + \mathbf{x}_{s_i}(t), & i \le z_2(t) \\ (\mathbf{x}_{s_{z_2(t)}}(t) - \mathbf{x}_i(t)) \times e^{w \times z_{1_i}(t)} \times \cos(2\pi \times z_{1_i}(t)) + \mathbf{x}_{s_{z_2(t)}}(t), & \text{o.w.} \end{cases} \tag{3.25}$$

where $w = 1$.

Lastly after evaluating the fitness of $\hat{\mathbf{x}}_i$, the special condition used in the selection method is $\forall$. At this moment, the re-framed MFO model has obtained the final generated new solutions after one round optimization. And finally dynamic influencing factors ($\mathbf{x}_s$, z_{1_i}, z_2) are going to be updated ($\mathbf{x}_s$ as Eq.3.26, z_{1_i} as Eq.3.27, z_2 as Eq.3.28) before the next optimization round.

$$\langle \mathbf{x}_i(t+1) \rangle = \textbf{Sort}(\{\mathbf{x}_i(t=0)\} \cup \{\mathbf{x}_i(t+1)\}), i = 1 \dots M \tag{3.26}$$

$$z_{1_i}(t+1) = r \times (-2 - \frac{t+1}{T}) + 1 \tag{3.27}$$

$$z_2(t+1) = \textbf{Round}(M - (t+1) \times \frac{M-1}{T}) \tag{3.28}$$

Till now, all symbols that are identified after the 'where' state are involved in the static influencing factors.

Algorithm 5 Moth-flame Optimization Algorithm

```
1: $t \leftarrow 0$
2: $Init_{\mathbf{x}}, M$                                  ▷ Initialization Process
3: $f$                                                   ▷ Evaluation
4: $Init_{\Delta}$: $\mathbf{x}_s, z$
5: while termination criteria are not met do
6:     $Opt_{\mathbf{x}}, C \rightarrow \hat{\mathbf{x}}_i(t+1)$    ▷ Optimization Process
7:     $f(\hat{\mathbf{x}}_i(t+1))$                        ▷ Evaluation
8:     $S \rightarrow \mathbf{x}_i(t+1)$                   ▷ Selection
9:     $t \leftarrow t+1$
10:    $Opt_{\Delta} \rightarrow \mathbf{x}_z(t+1), z(t+1)$
11: end while
```

3.2.5 Re-framed MBO

As discussed in the Table 2.2, the MBO model utilized two kinds if dynamic influencing factors and several static influencing factors to achieve the optimization on the objective solutions. These two dynamic influencing factors are Δ: $\mathbf{x}_g$ and z.

The re-framed MBO model, in the initialization process, will specially initialize $\mathbf{x}_g$ as Eq.3.7, z as Eq.3.29.

$$z(t=0)=\frac{w_4}{t^2} \tag{3.29}$$

where $w_4=1$. Next in the optimization process, before the stop condition T meets, the re-framed MBO model firstly directly generate $\hat{\mathbf{x}}_i(t+1)$ as Eq.3.30 with the help of dealing with outliers C with the help of dealing with outliers C as Eq.3.3.

$$\begin{aligned}
\langle \mathbf{x}_i(t)\rangle &= \textbf{Sort}(\{\mathbf{x}_i(t)\}), i=1\dots M \\
{}^{strong}\hat{\mathbf{x}}_{i,n}(t+1) &= \begin{cases} \mathbf{x}_{j,n}(t)\in\langle \mathbf{x}_i(t)\rangle, j\in[1,M'] & , \; r\times w_2 \leqslant w_1 \\ \mathbf{x}_{j,n}(t)\in\langle \mathbf{x}_i(t)\rangle, j\in(M',M] & , \; \text{o.w.} \end{cases} \\
M' &= \lceil w1\times M\rceil \\
{}^{weak}\hat{\mathbf{x}}_{i,n}(t+1) &= \begin{cases} \mathbf{x}_{g,n}(t), r\geqslant w_1 \\ \begin{cases} \mathbf{x}_{j,n}(t)+z(t)\times\left(\textbf{Lévy}_{i,n}-0.5\right), j\in(M',M], r>w_3 \\ \mathbf{x}_{j,n}(t)\in\langle \mathbf{x}_i(t)\rangle, j\in(M',M], \text{o.w.} \end{cases} & ,\text{o.w.} \end{cases} \\
\{\hat{\mathbf{x}}_i(t+1)\} &= \{{}^{strong}\hat{\mathbf{x}}_i(t+1)\}\cup\{{}^{weak}\hat{\mathbf{x}}_i(t+1)\}
\end{aligned} \tag{3.30}$$

where $w_1=\frac{5}{12}$, $w_2=1.2$, $w_3=\frac{5}{12}$ and $\textbf{Lévy}_{i,n}=\textbf{Lévy}(d,n,T)$ with $d\sim Exp(2\times T)$. Lastly after evaluating the fitness of $\hat{\mathbf{x}}_i(t+1)$, the special selection method is as Eq.3.31. At this moment, the re-framed MBO model has obtained the final generated new solutions after one round optimization. And finally the dynamic influencing factors ($\mathbf{x}_g$, z) are going to be updated before starting the next optimization round, in which the update method to $\mathbf{x}_g$ is as Eq.3.10, the update method to z is as Eq.3.32.

$$\begin{aligned}
&\langle \hat{\mathbf{x}}_i(t+1)\rangle=\textbf{Sort}(\{\hat{\mathbf{x}}_i(t+1)\}), i=1\dots M \\
&\mathbf{x}_i(t+1)\in\{\langle \hat{\mathbf{x}}_i(t+1)\rangle, i=1\dots M-w_5\}\cup\{\langle \mathbf{x}_i(t)\rangle, i=1\dots w_5\}
\end{aligned} \tag{3.31}$$

where $w_5=2$.

$$z(t+1)=\frac{w_4}{(t+1)^2} \tag{3.32}$$

Till now, all symbols that are identified the 'where' statement are involved in the static influencing factors.

Algorithm 6 Monarch Butterfly Optimization Algorithm

```
1: $t \leftarrow 0$
2: $Init_{\mathbf{x}}, M$                                ▷ Initialization Process
3: $f$                                                   ▷ Evaluation
4: $Init_{\Delta}$: $\mathbf{x}_g, z$
5: while termination criteria are not met do
6:     $Opt_{\mathbf{x}}, C \rightarrow \hat{\mathbf{x}}_i(t+1)$                ▷ Optimization Process
7:     $f(\hat{\mathbf{x}}_i(t+1))$                       ▷ Evaluation
8:     $S \rightarrow \mathbf{x}_i(t+1)$                   ▷ Selection
9:     $t \leftarrow t+1$
10:    $Opt_{\Delta} \rightarrow \mathbf{x}_g(t+1), z(t+1)$
11: end while
```

3.2.6 Re-framed BOA

As discussed in the Table 2.2, the BOA model utilized two kinds if dynamic influencing factors and several static influencing factors to achieve the optimization on the objective solutions. These two dynamic influencing factors are Δ: $\mathbf{x}_g$ and z.

The re-framed BOA model, in the initialization process, will specially initialize $\mathbf{x}_g$ as Eq.3.7, z as Eq.3.33.

$$z(t=0) = z^0 \tag{3.33}$$

where $z^0 = 0.01$.

Next in the optimization process, before the stop condition T meets, the re-framed BOA model will firstly firstly directly generate $\hat{\mathbf{x}}_i(t+1)$ as Eq.3.34 with the help of dealing with outliers C with the help of dealing with outliers C as Eq.3.3.

$$\hat{\mathbf{x}}_i(t+1) = \begin{cases} \mathbf{x}_i(t) + (r^2 \times \mathbf{x}_g(t) - \mathbf{x}_i(t)) \times z_1(t) \times f\left(\mathbf{x}_i(t)\right)^{w_1} & , \quad r > w_2 \\ \mathbf{x}_i(t) + (r^2 \times \mathbf{x}_j(t) - \mathbf{x}_k(t)) \times z_1(t) \times f\left(\mathbf{x}_i(t)\right)^{w_1} & , \quad \text{o.w} \end{cases} \tag{3.34}$$

where $w_1 = 0.1$, $w_2 = 0.8$. Moreover, $\mathbf{x}_j$ and $\mathbf{x}_k$ are any two neighbors around $\mathbf{x}_i$.

Lastly after evaluating the fitness of $\hat{\mathbf{x}}_i(t+1)$, the special condition used in the selection method is $f(\hat{\mathbf{x}}_i(t+1)) < f(\mathbf{x}_i(t))$. At this moment, the re-framed BOA model has obtained the final generated new solutions after one round optimization. And finally the dynamic influencing factors are going to be updated before starting the next optimization round, in which the update method to $\mathbf{x}_g$ is as Eq.3.10, the update method to z is as Eq.3.35.

$$z(t+1) = z(t) + \frac{0.025}{z(t) \times T} \tag{3.35}$$

Till now, all symbols that are identified after the 'where' statement are involved in the static influencing factors.

Algorithm 7 Butterfly Optimization Algorithm

```
1:  $t \leftarrow 0$
2:  $Init_{\mathbf{x}}, M$                              ▷ Initialization Process
3:  $f$                                              ▷ Evaluation
4:  $Init_{\Delta}$: $\mathbf{x}_g, z$
5:  while termination criteria are not met do
6:      $Opt_{\mathbf{x}}, C \rightarrow \hat{\mathbf{x}}_i(t+1)$          ▷ Optimization Process
7:      $f(\hat{\mathbf{x}}_i(t+1))$                         ▷ Evaluation
8:      $S \rightarrow \mathbf{x}_i(t+1)$                          ▷ Selection
9:      $t \leftarrow t+1$
10:     $Opt_{\Delta} \rightarrow \mathbf{x}_g(t+1), z(t+1)$
11: end while
```

3.2.7 Re-framed PSO

As discussed in the Table 2.2, the PSO model utilized three kinds of dynamic influencing factors and several static influencing factors to achieve the optimization on the objective solutions. These two dynamic influencing factors are Δ: $\mathbf{y}_i$, $\mathbf{x}_{i_p}$, $\mathbf{x}_g$.

The re-framed PSO model, in the initialization process, will specially initialize $\mathbf{y}_i$ as Eq.3.8, $\mathbf{x}_{i_p}$ as Eq.3.6, $\mathbf{x}_g$ as Eq.3.7. The static influencing factors will be also set at this moment.

Next in the optimization, before the stop condition T meets, the re-framed PSO model will firtsly generate the $\mathbf{y}_i$ as Eq.3.36, then generate $\hat{\mathbf{x}}_i(t+1)$ as Eq.3.37 with the help of dealing with outliers C with the help of dealing with outliers C as Eq.3.3.

$$\mathbf{y}_i(t+1) = w_1 \times \mathbf{y}_i(t) + \mathcal{U}(0, w_2) \times (\mathbf{x}_{i_p}(t) - \mathbf{x}_i(t)) + \mathcal{U}(0, w_3) \times (\mathbf{x}_g(t) - \mathbf{x}_i(t)) \tag{3.36}$$

where $w_1 = 0.73$, $w_2 = 1.49$, $w_2 = 1.49$.

$$\hat{\mathbf{x}}_i(t+1) = \mathbf{x}_i(t) + \mathbf{y}_i(t+1) \tag{3.37}$$

Lastly after evaluating the fitness of $\hat{\mathbf{x}}_i$, the special condition used in the selection method is $\forall$. At this moment, the re-framed PSO model has obtained the final generated new solutions after one round optimization. And finally dynamic influencing factors ($\mathbf{x}_{i_p}$, $\mathbf{x}_g$) are going to be updated ($\mathbf{x}_{i_p}$ as Eq.3.6, $\mathbf{x}_g$ as Eq.3.10) before starting the next optimization round.

Till now, all symbols that are identified after the 'where' state are involved in the static influencing factors.

Algorithm 8 Particle Swarm Optimization

```
1:  $t \leftarrow 0$
2:  $Init_{\mathbf{x}}, M$                                   ▷ Initialization Process
3:  $f$                                                     ▷ Evaluation
4:  $Init_{\Delta}$: $\mathbf{y}_i, \mathbf{x}_{i_p}, \mathbf{x}_g$
5:  while termination criteria are not met do
6:      $Opt_{\Delta} \rightarrow \mathbf{y}_i(t+1)$            ▷ Optimization Process
7:      $Opt_{\mathbf{x}}, C \rightarrow \hat{\mathbf{x}}_i(t+1)$
8:      $f(\hat{\mathbf{x}}_i(t+1))$                          ▷ Evaluation
9:      $S \rightarrow \mathbf{x}_i(t+1)$                       ▷ Selection
10:     $t \leftarrow t+1$
11:     $Opt_{\Delta} \rightarrow \mathbf{x}_{i_p}(t+1), \mathbf{x}_g(t+1)$
12: end while
```

3.3 Conclusion

Re-framed psuedo-codes[3] in the Section 3.2 show that **these seven algorithms can share a complete same model template Algorithm.1, no matter how different their original models are**. The Section 3.3 will firstly describe this unified framework in the high level overview, before detailing these broad observations.

In the high level overview, the Figure 3.1 illustrates that in the unified model template, there are two processes (Initialization Process and Optimization Process). One process has only one evaluation f. The f in the Initialization Process happens immediately after $Init_{\mathbf{x}}, M$, which is similar to that the f in the Optimization Process happens immediately after $Opt_{\mathbf{x}}, C$, because both of them generate a population. The main update strategy happens in the $Opt_{\mathbf{x}}, C$ in the Optimization Process, and algorithms are mainly distinguished to each other here. There are two main aspects (Δ, S) that will seriously affect the quality of optimization in the $Opt_{\mathbf{x}}$, in which the Δ impact is the

[3]BA: Algorithm.2. GOA: Algorithm.3. CSA: Algorithm.4. MFO: Algorithm.5. MBO: Algorithm.13. BOA: Algorithm.7. PSO: Algorithm.8.

Link-1 to 3,5 to 16, the S impact is the *Link-4,17*. Sometimes, some algorithms, such as BOA, MFO, will also utilize f fitness in their $Opt_{\mathbf{x}}$ method, as the *Link-18*.

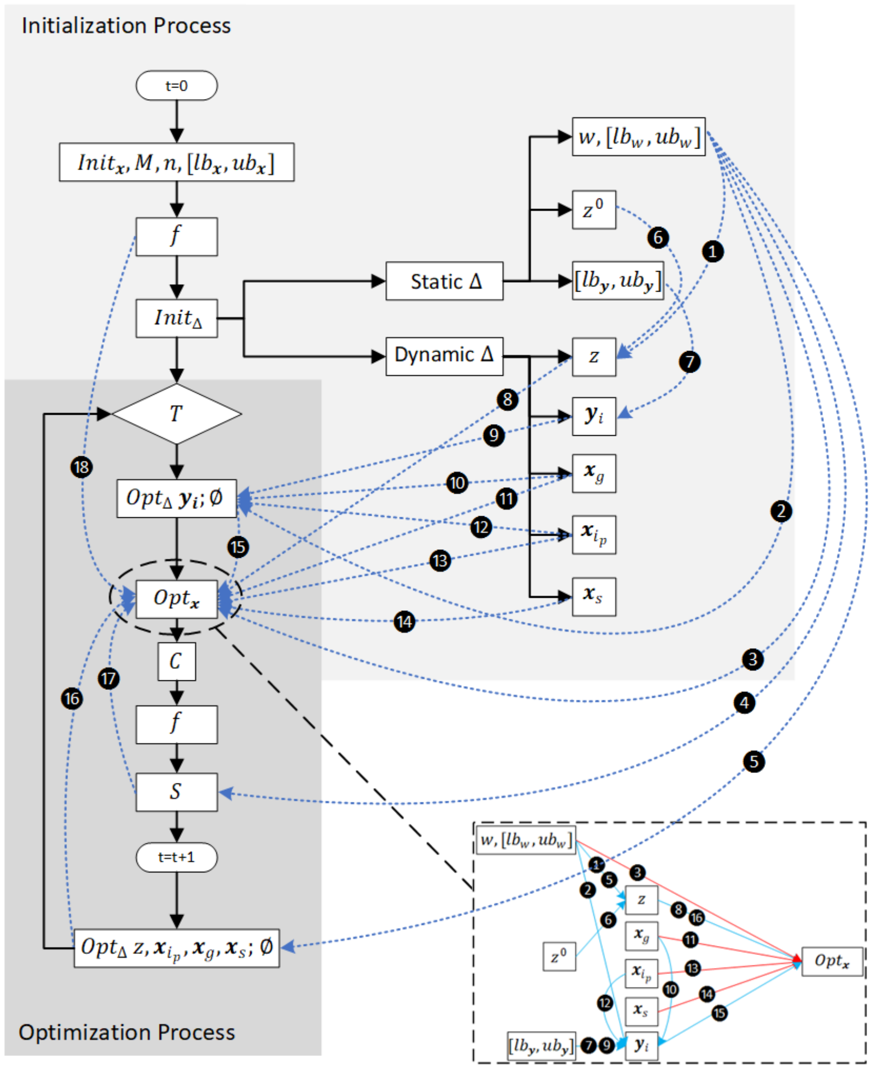
Initialization Process
t=0
$Init_{\mathbf{x}}, M, n, [lb_{\mathbf{x}}, ub_{\mathbf{x}}]$
f
$Init_{\Delta}$
Static Δ
Dynamic Δ
$w, [lb_w, ub_w]$
z^0
$[lb_{\mathbf{y}}, ub_{\mathbf{y}}]$
z
$\mathbf{y}_i$
$\mathbf{x}_g$
$\mathbf{x}_{i_p}$
$\mathbf{x}_s$
T
$Opt_{\Delta}\ \mathbf{y}_i; \emptyset$
$Opt_{\mathbf{x}}$
C
f
S
t=t+1
$Opt_{\Delta}\ z, \mathbf{x}_{i_p}, \mathbf{x}_g, \mathbf{x}_s; \emptyset$
Optimization Process

Figure 3.1: Unified framework with relationships amongst their components

The Figure 3.1, especially the small sub-picture in the bottom right corner, also illustrates that different algorithms will prepare different influencing factors Δ to participate in their optimization strategy $Opt_{\mathbf{x}}$. **Some participation is direct (bright red links)**, such as *Link-3, 11, 13, 14* in which some static Δ: $w, [lb_w, ub_w]$(commonly known as hyper-parameters) and some dynamic Δ: $\mathbf{x}_{i_p}, \mathbf{x}_g, \mathbf{x}_s$, are directly added in the $Opt_{\mathbf{x}}$. **Some participation is indirect (bright blue links)**, such as *Link-1, 5, 2, 6, 7, 9* in which these static Δ indirectly impact $Opt_{\mathbf{x}}$ by impacting some dynamic Δ: $z, \mathbf{y}$. Meanwhile, besides direct impact, dynamic Δ: $\mathbf{x}_{i_p}, \mathbf{x}_g$ can also impact $Opt_{\mathbf{x}}$ by impacting the dynamic Δ: $\mathbf{y}$, such as *Link-10, 12*.

Moreover, besides the Δ impact, the selection method S will also impact the $Opt_{\mathbf{x}}$, such as *Link-4, 17* in which static Δ firstly impacts the selection method, then the selection method impacts $Opt_{\mathbf{x}}$.

In the minor details level, the Table 3.1 illustrates the entire methods existed in these seven algorithms for each components in the unified framework, and points out whether these methods are allowed to be customized in the future. For example, $Init_{\mathbf{x}}, M$, f, $Init_{\Delta}$: $\mathbf{x}_{i_p}$, $\mathbf{x}_g$ and Opt_{Δ}: $\mathbf{x}_{i_p}$, $\mathbf{x}_g$ are not allowed to be customized, and they have to be kept same in any one of these seven algorithms. However, other components have much freedom in designing a customized method, such as $Init_{\Delta}$: $\mathbf{y}_i$, $\mathbf{x}_s$, z, w, Opt_{Δ}: $\mathbf{y}_i$, $\mathbf{x}_s$, z, $Opt_{\mathbf{x}}$, C and S, in which C has two options $C1$, $C2$, and S has four options $S1$, $S2$, $S3$, $S4$.

Until now, we can conclude, as the Table 3.2 displayed, at least in these seven algorithms, **algorithms are different because of the types of components they utilized and the design of these components, but the total types of components that can be selected and the order in which the components are arranged are totally same**.

A general utilization of this conclusion is to design an auto-designer for algorithms. For example, there are many open Python libraries [4,5,8,12,21,27] for designing evolutionary and genetic algorithms (EA), in which various 'crossover', 'mutation' and 'selection' are freely combined with each other to generate new algorithms. However, this researching road might not be such smooth in the UNIOA for swarm-based algorithms, which means **it might be not such intuitive to select several existing methods to combine a new algorithm in the UNIOA**. This is because swarm-based algorithms have more complicated mechanisms compared to EA, for example, the gene operations are limited but social activities in the real world are very various. In more details, in the most well-known EA common framework, the population will walk through 'crossover', 'mutation', 'selection'. One interesting observation to note here is that *you definitely will obtain a complete updated population even only using one of them, if ignoring the convergence*. However, in the swarm-based algorithm, it is hard to organized such a similar framework. For example, in the Table 3.1, in the *Index-4-$Init_{\Delta}$-z*, there are five different methods, however, it doesn't mean the method for BA is able to replace the method for GOA. One reason is the number of z is different, another reason is the positive impact of z for

BA might be negative impact for GOA in the $Opt_{\mathbf{x}}$. How to use *Index-4*, *Index-6* and *Index-11* only depend on how the *Index-7-$Opt_{\mathbf{x}}$* is designed.

Therefore, compared to freely packaging a new algorithm by selecting from existing methods in the EA framework, **the UNIOA emphasizes creating new methods from source**, which means the $Opt_{\mathbf{x}}$ is the beginning point to select or create other accessories. Moreover, compared to the rich selection pool in the EA framework, **the UNIOA emphasizes creating new $Opt_{\mathbf{x}}$ and its accessories** which means learning ideas from existing methods is more important than directly using existing methods.

Moreover, because C and S meet the condition that *you definitely will obtain a complete updated population even only using one of them, if ignoring the convergence*, $C1$, $C2$ and $S1$, $S2$, $S3$, $S4$ are also allowed to be freely utilized in any one algorithm. The details about these methods are detailed as follows:

- Two options $C1$, $C2$ to deal with outliers.

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

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

- Four options $S1$, $S2$, $S3$, $S4$ to select updated population from temporary generated population.

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

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

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

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

Table 3.1: In the unified framework, for each component, the existing methods collected in these seven algorithms and whether they could be customized in the future.

	Index	Components		Existing Method	Customization
Initialization	1	t=0			
	2	$Init_{\mathbf{x}}, M$		$\mathbf{x}_i(t=0) = \boldsymbol{\mathcal{U}}(lb_{\mathbf{x}}, ub_{\mathbf{x}}), i = 1, 2, \ldots, M$	no
	3	$f(\mathbf{x}_i(t=0)), i = 1, 2, \ldots, M$			no
	4	$Init_{\Delta}$	$\mathbf{y}_i$	$\mathbf{y}_i(t=0) = \boldsymbol{\mathcal{U}}(lb_{\mathbf{y}}, ub_{\mathbf{y}}), i = 1, 2, \ldots, M$	yes
			$\mathbf{x}_{i_p}$	$\mathbf{x}_{i_p}(t=0) = \mathbf{x}_i(t+1), i = 1, 2, \ldots, M$	no
			$\mathbf{x}_g$	$\mathbf{x}_g(t=0) = \mathbf{Min}(\{\mathbf{x}_i(t=0)\}), i = 1, 2, \ldots, M$	no
			$\mathbf{x}_s$	Eq.3.22(MFO)	yes
			z	Eq.3.12, Eq.3.13(BA). Eq.3.18(GOA). Eq.3.23, Eq.3.24(MFO). Eq.3.29(MBO). Eq.3.33(BOA)	yes
			w	commonly known as hyper-parameters	yes
Optimization	5	Stop condition T			
	6	Opt_{Δ}	$\mathbf{y}_i(t+1)$	Eq.3.14(BA). Eq.3.36(PSO)	yes
	7	$Opt_{\mathbf{x}}, C \rightarrow \hat{\mathbf{x}}_i(t+1)$		Eq.3.15(BA). Eq.3.19(GOA). Eq.3.25(MFO). Eq.3.30(MBO). Eq.3.34(BOA). Eq.3.37(PSO) $C1$ Eq.3.21(CSA) $C2$	yes
	8	$f(\hat{\mathbf{x}}_i(t+1))$			no
	9	$S \rightarrow \mathbf{x}_i(t+1)$		$S1$(GOA, CSA, MFO, PSO). $S2$(BOA). $S3$(BA). $S4$(MBO)	yes
	10	t=t+1			
	11	Opt_{Δ}	$z(t+1)$	Eq.3.16, Eq.3.17(BA). Eq.3.20(GOA). Eq.3.27, Eq.3.28(MFO). Eq.3.32(MBO). Eq.3.35(BOA)	yes
			$\mathbf{x}_{i_p}(t+1)$	$\mathbf{x}_{i_p}(t+1) = \mathbf{Min}(\mathbf{x}_{i_p}(t), \mathbf{x}_i(t+1), i = 1, 2, \ldots, M$	no
			$\mathbf{x}_g(t+1)$	$\mathbf{x}_g(t+1) = \mathbf{Min}(\mathbf{x}_g(t) \cup \{\mathbf{x}_i(t+1)\}), i = 1, 2, \ldots, M$	no
			$\mathbf{x}_s(t+1)$	Eq.3.26(MFO)	yes

Table 3.2: How these components in Table 2.2 are structured in these seven algorithms.

Progress	Index	Components		BA	GOA	CSA	MFO	MBO	BOA	PSO
Initialization	1	t=0								
	2	$Init_{\mathbf{x}}, M$		Eq.3.2						
	3	$f(\mathbf{x}(t))$								
	4	$Init_{\Delta}$	$\mathbf{y}_i$	Eq.3.11	×	✓	×	×	×	Eq.3.8
			$\mathbf{x}_{i_p}$	×	×	Eq.3.6	×	×	×	Eq.3.6
			$\mathbf{x}_g$	Eq.3.7	Eq.3.7	×	×	Eq.3.7	Eq.3.7	Eq.3.7
			$\mathbf{x}_s$	×	×	×	Eq.3.22	×	×	×
			z	Eq.3.12, Eq.3.13	Eq.3.18	×	Eq.3.23, Eq.3.24	Eq.3.29	Eq.3.33	×
			w	✓	✓	✓	✓	✓	✓	✓
Optimization	5	Stop condition T								
	6	Opt_{Δ}	$\mathbf{y}_i(t+1)$	Eq.3.14	×	×	×	×	×	Eq.3.36
	7	$Opt_{\mathbf{x}}, C \to \hat{\mathbf{x}}_i(t+1)$		Eq.3.15, $C1$	Eq.3.19, $C1$	Eq.3.21, $C2$	Eq.3.25, $C1$	Eq.3.30, $C1$	Eq.3.34, $C1$	Eq.3.37, $C1$
	8	$f(\hat{\mathbf{x}}_i(t+1))$								
	9	$S \to \mathbf{x}_i(t+1)$		$S3$	$S1$	$S1$	$S1$	$S4$	$S2$	$S1$
	10	t=t+1								
	11	Opt_{Δ}	$z(t+1)$	Eq.3.16, Eq.3.17	Eq.3.20	×	Eq.3.27, Eq.3.28	Eq.3.32	Eq.3.35	×
			$\mathbf{x}_{i_p}(t+1)$	×	×	Eq.3.9	×	×	×	Eq.3.9
			$\mathbf{x}_g(t+1)$	Eq.3.10	Eq.3.10	×	×	Eq.3.10	Eq.3.10	Eq.3.10
			$\mathbf{x}_s(t+1)$	×	×	×	Eq.3.26	×	×	×

Chapter 4

Benchmark Environment

4.1 Why is IOHprofiler

To prove the generic dictionary and framework suit these seven algorithms (see Chapter 2) correctly, except on the theoretical view (Chapter 3), the more convincing evidence is that the algorithms in both representations perform , to a large extent, in actual optimization problems. Moreover, instead of observing whose performance is the best among the algorithms, it is more significant to prove how far the algorithms in two representations can be close to each other.

Therefore, rather than a set of optimization problems used to test which algorithm is the winner, it's more reasonable to test the algorithms in both representations on a set of enough various optimization problems. Meanwhile, in this work, seven algorithms are created for solving continuous optimization problems [13–17, 30, 31]. For these reasons, we expect a benchmark environment in which enough various continuous optimization problems can be used to prove the availability of the generic dictionary and framework.

The IOHprofiler can meet these mentioned requirements. The IOHprofiler is a place in which you can feed into your own iterative optimization algorithms and then collect time-series data when optimizing problems. Lastly, these generated data will be statistically analyzed in a well-established website[1] [7, 29].

A very important assumption in the IOHprofiler is that the algorithm performance is only determined by the number of evaluations happening on possible solutions [7]. In other words, the IOHprofiler environment thinks that evaluating possible solutions accounts for a huge part of the actual algorithm running time[2].

Therefore, in the IOHprofiler environment, when measuring the performance of one algorithm A on one problem f in one complete experiment r, the number of evaluations on possible solutions is regarded as the Expected Running Time ERT as Eq.4.1. In general, the result from one experiment might be random, so several experiments r are normally taken to provide a more convinced result as Eq.4.2.

$$ERT(A, f, r = 1) = Count(f) \qquad 4.1$$

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

[2] The time spent in other components of algorithms is ignored in the IOHprofiler environment.

$$ERT(A, f, r) = \frac{\sum_{i=1}^{r} Count(f)}{r} \qquad 4.2$$

Furthermore, the IOHprofiler provides 24 different single-objective continuous optimization problems called BBOB set [9]. Meanwhile, the IOHprofiler environment also allows the user to test one algorithm A on different variants of one particular problem f [7], which benefits enough various problems mentioned above. In the IOHprofiler environment, the variants of one problem f is achieved by different scaling operators as $af(\sigma(x \oplus z)) + b$ in which a, σ, $\oplus z$ and b are respectively multiplicative shift, permutation, XOP-shift and additive shift [7]. In the IOHprofiler environment, different variants of one problem f are called problem instances. Moreover, one algorithm A on one instance of one problem f is one complete experiment r, which means for one problem f, if there are five instances and each instance is tested three times, there are totally $r = 15$ experiments.

Lastly, as the IOHprofiler environment is designed for analyzing iterative optimization algorithms in which the stop condition is an important feature, the IOHprofiler environmet provides two different stop conditions that are when there are enough iterative optimization rounds called Fixed-Budget, or when the algorithm already find the optimal solution called Fixed-Target.

4.2 How IOHprofiler works in this work

The IOHprofiler involves two main parts: IOHexperimenter and IOHanalyzer. The experiments in this work will happen in these two parts as follows:

- In the IOHexperimenter:
  As described in the Section 4.1, the IOHenvironment allows you to test your own algorithm and then to collect optimization data under your own algorithm. This will happen in the IOHexperimenter environment. The IOHexperimenter provides an interactive Python environment[3] in which objective problem instances can be automatically generated and tested on particular algorithms coded by the user.

  All experiments on one problem f are recorded in the generated data file which will be analyzed in the IOHanalyzer environment. Meanwhile, in this work, we will also employ these two different stop conditions provided by the IOHexperimenter.

- In the IOHanalyzer:
  The IOHanalyzer provides various performance analyses [29], but in this work,

[3] https://iohprofiler.github.io/IOHexp/python/

we only focus on the ERT performance when the stop condition is Fixed-Target. The IOHanalyzer will provide a clear plot in which we could observe algorithm's performance on 24 optimization problems and we could conclude if plot lines in two representations were closer enough.

Furthermore, except for the view of visualisation, the IOHanalyzer also allows users to demonstrate algorithms' performance on the view of quantitative analysis. In this work, we only focus on the approximated area under (AUC) the Empirical Cumulative Distribution Function (ECDF) curve. As in the IOHanalyzer, the ECDF displays the proportion of each target value under each ERT, in which the range of ERT is from one evaluation to the customized maximum evaluations, and the range of target values is ten linearly distributed values from the customized minimum fitness[4] to the obtained worst fitness[5]. Meanwhile, we will expect if the AUC in both representations are close enough, and the higher AUC to 1 is considered the better one.

Huilin: add what is considered a good and better performance

[4]Because these are minimizing optimization problems

[5]The number ten and linearly distribution are default settings in the IOHanalyzer.

Chapter 5

Experiments

5.1 Experiments Configurations

The purpose of experiments in the Chapter 5 is to test the theoretical assumption (in Chapter 2, Chapter 3) is, to a great extent, correct: **UNIOA with its accessories works correctly in these seven algorithms**. The UNIOA is the unified framework proposed in the Chapter 3, it accessories is the 20 components organized in the Chapter 2.

To achieve such experiments,

The experiments in this work have two important parts: algorithms part and benchmark environment part. For the algorithms part, besides implementing each algorithm in two representations with their default hyper-parameter settings (), it is also important to implement these algorithms in the IOHexperimenter environment (see Section 4.2). When implementing algorithms in the IOHexperimenter environment, in addition to the fixed flows of coding algorithms and collecting optimization time-series data[1], the stop condition is open and customized by the user. In this work, as described in the Section 4.1, we will consider two stop conditions as follows and when any one stop condition is met, the optimization process will stop.

- Fixed-Budget: the maximum number of iterations is $1 \times 10^4 \times n$ in which the dimension n will be varying.
- Fixed-Target: the minimum acceptable fitness value is 1×10^{-8}, since there will be minimizing optimization problems.

[1]Exact steps are displayed in the https://iohprofiler.github.io/IOHexp/python/

For the benchmark environment part, experiments will happen on the entire 24 BBOB functions whose variants will be five, furthermore, each variant will have five independent experiments.

5.2 Pre-test

Firstly, to make the discussion in the following sections clearer and easier to understand, we use 'UNIOA' denotes the algorithm in the UNIOA framework with generic representations. 'SEPARATE' (abbr. SEP) denotes the algorithm model in its own separate representation.

Meanwhile, when reproducing these seven SEP-algorithms in the same benchmark environment as the seven UNIOA-algorithms, there are two situations that might mislead the performance of future experiments:

(1) UNIOA-algorithm behave differently from SEP-algorithms because of synchronous or asynchronous evaluation (abbr. syncE, asyncE), not because of the UNIOA framework works wrong.

(2) UNIOA-algorithm behave differently from SEP-algorithms because of synchronous or asynchronous $\mathbf{x}_g$ calculation method (abbr. syncG, asyncG), not because of the UNIOA framework works wrong.

Therefore, **we need to prevent syncE/asyncE and syncG/asyncG from misleading the conclusion about if the UNIOA framework works correctly or not**. To avoid the possible negative effects from syncE/asyncE and syncG/asyncG, we made 33 sub-experiments in which[2]:

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

[2]Normally, for the five algorithms who use $\mathbf{x}_g$, there should be $5 \times 2 \times 2 \times 2 = 40$ 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, as we observed in these implementations, it is impossible to calculate $\mathbf{x}_g$ asynchronously when the evaluation is 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 the UNIOA-algorithms, when the evaluation is asynchronous, it is meaningless to set the $\mathbf{x}_g$ asynchronous also, which is because in the UNIOA framework, other components will also have to be changed when the evaluation and $\mathbf{x}_g$ are simultaneously asynchronous, then it is impossible to decide what exactly impact 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.

- Each SEP-algorithm has three options: when the evaluation is synchronous, the $\mathbf{x}_g$ is synchronous, but when the evaluation is asynchronous, the $\mathbf{x}_g$ could be synchronous or asynchronous.

The **??** and The Section C and

In the Section 5.3, we will firstly discuss benchmark performances of each algorithm in both representations. With the help of the IOHanalyzer described in the Section 4.2, we will prove algorithms in 'Generic' works as much same as in 'Specific' from two perspectives: ERT and AUC of ECDF (see Chapter 4).

Secondly, in the Section 5.4, according conclusions in the Section 5.3, we will develop the UNIOA[3] that is able to help users to design their own swarm-based algorithms with only math knowledge, without any nature/bio knowledge, moreover, give users insights in comparing algorithms on different components.

Then, in the Section 5.5, we will make an example of one comparison insight obtained in the Section 5.4.

Lastly, in the Section 5.6, we will conclude the entire findings in the Chapter 5 that will lead to final conclusions and future discussions about this work in the **??**.

5.3 Results on Generic Representations

After uploading the generated data to the IOHanalyzer, for the BA example, when the dimension $n = 5$ and the stop condition is Fixed-Target, the Figure 5.1 displays the 'Generic' works, to a great extent, as same as the 'Specific', especially in the problem functions (F2-F4, F7-F12, F15-F20, F24). Meanwhile, in the F13, F21, F22, the 'Generic' works better than the 'Specific'. However, the 'Specific' clearly is much better than the 'Generic' in the F1, F5, F6, F14, F23.

Most importantly, the conclusion in the view of visualization is very subjective. Therefore, we will clarify three principles used in this work to state the rationality of our conclusions. As seen in the Figure 5.1, the curve that is lower or has a long tail demonstrates a better performance, therefore, it is important to state how to distinguish three possible conclusions: 'Generic' is worse than 'Specific', 'Generic' is same as 'Specific' and 'Generic' is better than 'Specific'.

[3] `https://pypi.org/project/UNIOA/`

0-thesis/pics/5-ba.pdf

Figure 5.1: ERT trends on 24 functions when the dimension $n = 5$ and the stop condition is Fixed-Target. The purple curve denotes the BA algorithm in the generic representation. The orange dash curve denote the BA algorithm in its own representation.

After observing 288 plots in which each plot displays the relationship between 'Generic' and 'Specific' on one problem function on one dimension, we considered if both curves stop at the close position, the gap between two curves determines the relationship. For example, the 'Generic' loses the 'Specific' in the Figure 5.2a because the gap between two curves is large. However, in the Figure 5.2b, both curves also stop at the close position, but we thought the 'Generic' and the 'Specific' draw because of a smaller gap and a longer coincidence between curves.

0-thesis/pics/5BAF14.png

(a) For BA, the relationship between 'Generic' and 'Specific' on F14 under $n = 5$. 'Specific' wins because of the large gap between curves

0-thesis/pics/5MBOF11.png

(b) For MBO, the relationship between 'Generic' and 'Specific' on F11 under $n = 5$. 'Generic' and 'Specific' draw because of the smaller gap and a longer coincidence between curves.

0-thesis/pics/20CSAf23.png

(c) For CSA, the relationship between 'Generic' and 'Specific' on F23 under $n = 20$. 'Generic' and 'Specific' draw because of the longer coincidence between curves.

0-thesis/pics/20GOAF19.png

(d) For GOA, the relationship between 'Generic' and 'Specific' on F19 under $n = 20$.

Figure 5.2: Four examples about what kind of curve is considered the 'Generic' loses or draws or wins the 'Specific'.

If both curves didn't stop at a close position, the length of the coincident part between two curves determines if they are same or there is a winner. For example, the curves in the Figure 5.2c with a longer coincidence states the 'Generic' is same as the 'Specific', however, the curves in the Figure 5.2d with a shorter coincidence states there is a winner.

Based on these two principles in the Figure 5.2, we concluded the relationship between the 'Generic' and the 'Specific' in the entire 288 curve pairs in the Figure 5.3 in which the 'Generic' performed better than or same as the 'Specific', but the 'Generic' also would perform worse than the 'Specific', especially in the BA case.

0-thesis/pics/ryb.png

Figure 5.3: The relationship between 'Generic' and 'Specific' under Fixed-Target stop condition when $n = 5$ (left) and $n = 20$ (right). Blue cross $\times$ denotes the 'Generic' is worse than the 'Specific'. The yellow circle $\bigcirc$ denotes the 'Generic' is same as the 'Specific'. The red checkmark $\checkmark$ denotes the 'Generic' is better than the 'Specific'.

Table 5.1: AUC for ECDF when $n = 5$, $n = 20$ and aggregated dimension.

AUC	n=5		n=20		aggregated dimensions	
	'Generic'	'Specific'	'Generic'	'Specific'	'Generic'	'Specific'
BA	0.8693	0.9084	0.8693	0.9084	0.8693	0.9084
GOA	0.9307	0.9084	0.9307	0.9084	0.9307	0.9084
CSA	0.9145	0.8518	0.9145	0.8518	0.9145	0.8518
BOA	0.9130	0.8165	0.9130	0.8165	0.9130	0.8165
MBO	0.9281	0.8066	0.9281	0.8066	0.9281	0.8066
MFO	0.9045	0.8995	0.9045	0.8995	0.9045	0.8995

As mentioned in the Section 4.2, the IOHanalyzer also provides the AUC for ECDF in order to analysis more quantitatively. We concluded the AUC value for each algorithm in both representations in the Table 5.1 in which the AUC from the 'Generic' is much closer to 1 in the GOA, CSA, MBO, MFO. Although for the BA, the AUC of the 'Generic' is lower than the 'Specific', the slight difference is able to be accecpted that the 'Generic' is same as the 'Specific' to a great extent.

5.4 UNIOA Package

According to the findings in the Section 5.3, we have proved the feasibility of the generic framework described in the Chapter 3. As we discussed in the Section 3.3, based on the generic framework, the differences between algorithms is able to be discussed only on the math level without any nature/bio knowledge.

Based on these ideas:

- The math formulas totally determine the ability of different algorithm.
- The optimization flows are able to be fixed as the Algorithm.1.

We developed a small python package named '**UNIOA**' in which users are able to design their own ierative optimization algorithms only with math knowledge. Moreover, benchmark your own algorithm with other seven existing algorithms in the IOHprofiler environment. There is one detailed example in the `https://github.com/Huilin-Li/UNIOA`.

Huilin: should I also describe that example here?

5.5 Results on Comparison Insights in Generic Representations

The generic framework described in the Chapter 3 states the algorithms' components are able to be represented as math formulas, moreover, these components have their fixed position in the optimization flows. Based on the successful example in the Section 5.4,

5.6 Conclusions

Chapter 6

Conclusions and Future Work

6.1 Conclusions

The influence of generic notations/operators could be noticed when observing the development difference between EA group and SI group. The very first clear difference is that the genetic-terminology (mutation, crossover, selection) is the only one analogy in EA but SI has various analogies (bee, ant, whale and so on). Due to generic terminologies, EA group develops well in both theoretical comparison study and auto-algorithm design **Huilin: +ref**. However, SI group seems to have stopped at simulating one natural observation into one optimization process. Recent studies on SI group began to focus on analysing SI group as a whole, such as a comparison study amongst a set of SI algorithms**Huilin: +ref2**, a platform environment in which automatically calls needed algorithms**Huilin: +ref**, a unified framework that subsumes a set of population-based algorithms [18]. Inspired by these previous studies, we hope the comparison can happen in a larger set of SI, we hope auto-design can also happen in optimization operators (just like combinations of operators in EA group), we also hope primitive math knowledge could be enough to construct such a unified framework.

In the 'Generic' environment, amongst nature-inspired algorithms, we have analysed their similarities on symbols and operators in Chapter 2; we also have analysed their similarities on structures and their differences on operators in Chapter 3; moreover, experiments in Chapter 5 have proved that 'Generic' can safely replace 'Specific' to a significant extent.

Meanwhile, the Section 5.2 has a taste for population size influence on nature-inspired algorithms in the 'Generic' environment. In this **??**, we will continue theoretically discussing more insights in the view of algorithms' comparison.

If we accept that all nature-inspired algorithms can be divided into these mentioned components,

- does it mean people also can compare algorithms only on their update strategies? For example, comparisons only happen on 'Opt_X'.
- does it mean people in the future could develop new methods by developing these specific components? For example, if there is a more smart method to initialize population, if there is a more efficient method to update assisting variables?

6.2 Future Work

For the future research, we focus on discussing the possibility of using this 'Generic' to unify the entire nature-inspired algorithms including EA and SI. **Huilin: parallel space**

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

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

[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] 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.

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

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

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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

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

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

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

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

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

[22] 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.

[23] BR Rajakumar and Aloysius George. Apoga: An adaptive population pool size based genetic algorithm. AASRI Procedia, 4:288–296, 2013.

[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 Experiment results in the Chapter 5

B Re-framed Algorithm in the Chapter 3

B.1 Re-framed Bat Inspired Algorithm (Re-framed BA)

- **Objectives**:

Objective Problem	
$f(\mathbf{x}_i)$	fitness of $\mathbf{x}_i$.
n	the dimension of the search space.
$[lb_{\mathbf{x}}, ub_{\mathbf{x}}]$	the interval of objective variable $\mathbf{x}$, in our cases, it is defined in the IOHprofiler, $[lb_{\mathbf{x}}, ub_{\mathbf{x}}] = [-5, +5]$.
Objective Solution	
$\mathbf{x}_i$	it can be imagined as one individual in Swarm-Intelligence Algorithms, $\mathbf{x}_i \in R^n$.

- **Influencing factors**:

Dynamic influencing factors	
$\mathbf{x}_g$	the best position that the whole population has found so far.

$\mathbf{y}_i$	the velocity for one individual $\mathbf{y}_i$.
z_1	a kind of probability, in this case, it is called the decreasing pulse rate.
z_2	the decreasing loudness.
Static influencing factors	
T	maximum iteration, the budget in our cases, in our case, it is defined in IOHprofiler.
M	population size, $M = 20, M \in [20,40]$.
z_1^0	the initial value of pulse rate z_1, $z_1^0 = 1$, $z_1^0 \in [0,1]$.
z_2^0	the initial value of loudness z_2, $z_2^0 = 1$, $z_2^0 \in (0,+\infty)$.
w_1	used to decrease pulse rate z_1 , $w_1 = 0.1$, $w_1 \in [-1,1]$.
w_2	used to decrease loudness z_2, $w_2 = 0.97$, $w_2 \in [-1,1]$.
w_3	used to update local position, $w_3 = 0.1$, $w_3 \in [-1,1]$.
$[lb_{w_4}, ub_{w_4}]$	the interval of frequency w_4, $[lb_{w_4}, ub_{w_4}] = [0,2]$. $[lb_{w_4}, ub_{w_4}] \subset [0,+\infty]$.

- **Components**:
  - Initialization Process:
    (1) Initialize $\mathbf{x}_i(t=0)$:
$$\mathbf{x}_i(t=0) = \boldsymbol{\mathcal{U}}(lb_{\mathbf{x}}, ub_{\mathbf{x}}) \tag{6.1}$$
    (2) Initialize $\mathbf{y}_i(t=0)$:
$$\mathbf{y}_i(t=0) = \boldsymbol{\mathcal{U}}(0,0) \tag{6.2}$$
    (3) Initialize $\mathbf{x}_g(t=0)$:
$$\mathbf{x}_g(t=0) = \textbf{Min}(\{\mathbf{x}_i(t)\}), i = 1\ldots M \tag{6.3}$$

(4) Initialize $z_1(t=0)$:

$$z_1(t=0) = z_1^0 \times (1 - e^{-w_1 \times t}) \tag{6.4}$$

(5) Initialize $z_2(t=0)$:

$$z_2(t=0) = z_2^0 \times w_2 \tag{6.5}$$

– Optimization Process:

(1) Optimize the velocity $\mathbf{y}_i(t)$ to generate $\mathbf{y}_i(t+1)$:

$$\mathbf{y}_i(t+1) = \mathbf{y}_i(t) + \mathcal{U}(lb_{w_4}, ub_{w_4}) \times (\mathbf{x}_i(t) - \mathbf{x}_g(t)) \tag{6.6}$$

(2) Optimize $\mathbf{x}_i(t+1)$ to generate $\hat{\mathbf{x}}_i(t+1)$:

$$\hat{\mathbf{x}}_i(t+1) = \begin{cases} \mathbf{x}_g(t) + w_3 \times \boldsymbol{\mathcal{N}}(\mathbf{0},\mathbf{1}) \times z_2(t) & , \quad r < z_1(t) \\ \mathbf{x}_i(t) + \mathbf{y}_i(t+1) & , \quad \text{o.w} \end{cases} \tag{6.7}$$

(3) Dealing with outliers C:

$$\mathbf{x}_{i,n}^{\text{fixed}}(t+1) = \begin{cases} ub_x & , \quad \mathbf{x}_{i,n}(t+1) > ub_{\mathbf{x}} \\ \mathbf{x}_{i,n}(t+1) & , \quad \text{o.w} \\ lb_x & , \quad \mathbf{x}_{i,n}(t+1) < lb_{\mathbf{x}} \end{cases} \tag{6.8}$$

(4) Select $\mathbf{x}_i(t+1)$ from $\hat{\mathbf{x}}_i(t+1)$:

$$\mathbf{x}_i(t+1) = \begin{cases} \hat{\mathbf{x}}_i(t+1) & , \quad f(\hat{\mathbf{x}}_i(t+1)) < f(\mathbf{x}_i(t)) \text{ or } r > z_2(t) \\ \mathbf{x}_i(t) & , \quad \text{o.w} \end{cases} \tag{6.9}$$

(5) Optimize $\mathbf{x}_g(t)$ to generate $\mathbf{x}_g(t+1)$:

$$\mathbf{x}_g(t+1) = \textbf{Min}(\mathbf{x}_g(t) \cup \{\mathbf{x}_i(t+1)\}), i = 1 \ldots M \tag{6.10}$$

(6) Optimize the pulse rate $z_1(t)$ to generate $z_1(t+1)$:

$$z_1(t+1) = z_1^0 \times (1 - e^{-w_1 \times (t+1)}) \tag{6.11}$$

(7) Optimize the decreasing loudness $z_2(t)$ to generate $z_2(t+1)$:

$$z_2(t+1) = z_2(t) \times w_2 \tag{6.12}$$

Algorithm 9 Re-framed Bat Algorithm

```
1:  $t \leftarrow 0$
2:  $Init_{\mathbf{x}}(t=0), M$ as Eq.6.1                    ▷ Initialization Process
3:  $f$                                                    ▷ Evaluation
4:  $Init_{\Delta}$: $\mathbf{y}_i(t=0)$ as Eq.6.2
5:  $Init_{\Delta}$: $\mathbf{x}_g(t=0)$ as Eq.6.3
6:  $Init_{\Delta}$: $z_1(t=0)$ as Eq.6.4
7:  $Init_{\Delta}$: $z_2(t=0)$ as Eq.6.5
8:  while termination criteria are not met do
9:      $Opt_{\Delta} \rightarrow \mathbf{y}_i(t+1)$ as Eq.6.6       ▷ Optimization Process
10:     $Opt_{\mathbf{x}}, C \rightarrow \hat{\mathbf{x}}_i(t+1)$ as Eq.6.7, $C$ as Eq.6.8
11:     $f(\hat{\mathbf{x}}_i(t+1))$                               ▷ Evaluation
12:     $S \rightarrow \mathbf{x}_i(t+1)$ as Eq.6.9                  ▷ Selection
13:     $t \leftarrow t+1$
14:     $Opt_{\Delta} \rightarrow \mathbf{x}_g(t+1)$ as Eq.6.10
15:     $Opt_{\Delta} \rightarrow z_1(t+1)$ as Eq.6.11
16:     $Opt_{\Delta} \rightarrow z_2(t+1)$ as Eq.6.12
17: end while
```

B.2 Re-framed Grasshopper Optimization Algorithm (Re-framed GOA)

- **Objectives**:

Objective Problem	
$f(\mathbf{x}_i)$	fitness of $\mathbf{x}_i$, $(f : R^n \rightarrow R)$.
n	the dimensionality of the search space.
$[lb_{\mathbf{x}}, ub_{\mathbf{x}}]$	the interval of objective variable $\mathbf{x}$, in our case, it is defined in IOHprofiler, $[lb_{\mathbf{x}}, ub_{\mathbf{x}}] = [-5, +5]$.
Objective Solution	
$\mathbf{x}_i$	it can be imagined as one individual in Swarm-Intelligence Algorithms, $\mathbf{x}_i \in R^n$.

- **Influencing factors**:

Dynamic influencing factors	
$\mathbf{x}_g$	the best position that the whole population has found so far.
z	a decreasing coefficient.
Static influencing factors	
T	maximum iteration, the budget in our cases, in our case, it is defined in IOHprofiler.
M	population size, $M = 100$.
$[lb_z, ub_z]$	the interval of assisting variable z, $[lb_z, ub_z] = [0.00004.1]$, $[lb_z, ub_z] \subset [-\infty, +\infty]$.
w_1	the intensity of attraction, $w_1 = 0.5$.
w_2	the attractive length scale, $w_2 = 1.5$.

- **Components**:
  - Initialization Process:

    (1) Initialize $\mathbf{x}_i(t=0)$:

$$\mathbf{x}_i(t=0) = \boldsymbol{\mathcal{U}}(lb_{\mathbf{x}}, ub_{\mathbf{x}}) \tag{6.13}$$

    (2) Initialize $\mathbf{x}_g(t=0)$:

$$\mathbf{x}_g(t=0) = \textbf{Min}(\{\mathbf{x}_i(t)\}), i = 1 \dots M \tag{6.14}$$

    (3) Initialize $z(t)$:

$$z_1(t) = ub_{z_1} - t \times (\frac{ub_{z_1} - lb_{z_1}}{T}) \tag{6.15}$$

  - Optimization Process:

(1) Optimize $\mathbf{x}_i(t)$ to generate $\hat{\mathbf{x}}_i(t+1)$:

$$\begin{aligned}\widetilde{D}_{i,j}(t) &= 2 + \textbf{Dist}(\mathbf{x}_i(t), \mathbf{x}_j(t))\,\textbf{mod}\,2\\ \hat{\mathbf{x}}_i(t+1) &= z_1(t) \times (\sum_{j=1, j\neq i}^{M} z_1(t) \times \frac{ub_{\mathbf{x}} - lb_{\mathbf{x}}}{2} \times\\ &(w_1 \times e^{\frac{-\widetilde{D}_{i,j}(t)}{w_2}} - e^{-\widetilde{D}_{i,j}(t)}) \times\\ &\frac{\mathbf{x}_i(t) - \mathbf{x}_j(t)}{\textbf{Dist}(\mathbf{x}_i(t), \mathbf{x}_j(t))}) + \mathbf{x}_g\end{aligned} \qquad 6.16$$

(2) Dealing with outliers C:

$$\mathbf{x}_{i,n}^{\text{fixed}}(t+1) = \begin{cases} ub_x & , \ \mathbf{x}_{i,n}(t+1) > ub_x \\ \mathbf{x}_{i,n}(t+1) & , \ \text{o.w} \\ lb_x & , \ \mathbf{x}_{i,n}(t+1) < lb_x \end{cases} \qquad 6.17$$

(3) Select $\mathbf{x}_i(t+1)$ from $\hat{\mathbf{x}}_i(t+1)$:

$$\mathbf{x}_i(t+1) = \hat{\mathbf{x}}_i(t+1) \qquad 6.18$$

(4) Optimize $\mathbf{x}_g(t)$ to generate $\mathbf{x}_g(t+1)$:

$$\mathbf{x}_g(t+1) = \textbf{Min}(\mathbf{x}_g(t) \cup \{\mathbf{x}_i(t+1)\}), i = 1 \ldots M \qquad 6.19$$

(5) Optimize $z(t)$ to generate $z(t+1)$:

$$z(t) = ub_z - t \times (\frac{ub_z - lb_z}{T}) \qquad 6.20$$

Algorithm 10 Re-framed Grasshopper Optimization Algorithm

```
1: $t \leftarrow 0$
2: $Init_{\mathbf{x}}(t=0), M$ as Eq.6.13          ▷ Initialization Process
3: $f$                                             ▷ Evaluation
4: $Init_{\Delta}$: $\mathbf{x}_g(t=0)$ as Eq.6.14
5: $Init_{\Delta}$: $z(t=0)$ as Eq.6.15
6: while termination criteria are not met do
7:     $Opt_{\mathbf{x}}, C \rightarrow \hat{\mathbf{x}}_i(t+1)$ as Eq.6.16, $C$ as Eq.6.17   ▷ Optimization Process
8:     $f(\hat{\mathbf{x}}_i(t+1))$                ▷ Evaluation
9:     $S \rightarrow \mathbf{x}_i(t+1)$ as Eq.6.18   ▷ Selection
10:    $t \leftarrow t+1$
11:    $Opt_{\Delta} \rightarrow \mathbf{x}_g(t+1)$ as Eq.6.19
12:    $Opt_{\Delta} \rightarrow z_1(t+1)$ as Eq.6.20
13: end while
```

B.3 Re-framed Crow Search Algorithm (Re-framed CSA)

- **Objectives**:

Objective Problem	
$f(\mathbf{x}_i)$	fitness of $\mathbf{x}_i$.
n	the dimension of the search space.
$[lb_{\mathbf{x}}, ub_{\mathbf{x}}]$	the interval of objective variable $\mathbf{x}$, in our cases, it is defined in the IOHprofiler, $[lb_{\mathbf{x}}, ub_{\mathbf{x}}] = [-5, +5]$.
Objective Solution	
$\mathbf{x}_i$	it can be imagined as one individual in Swarm-Intelligence Algorithms, $\mathbf{x}_i \in R^n$.

- **Influencing factors**:

Dynamic influencing factors	
$\mathbf{x}_{i_p}$	the memory position $\mathbf{x}_{i_p}$ for one individual $\mathbf{x}_i$.

Static influencing factors	
T	maximum iteration, the budget in our cases, in our case, it is defined in IOHprofiler.
M	population size, $M = 50$.
w_1	awareness probability, $w_1 = 0.1$, $w_1 \in [0,1]$.
w_2	flight length, $w_2 = 2$, $w_2 \in (0,+\infty)$.

- **Components**:
  - Initialization Process:
    (1) Initialize $\mathbf{x}_i(t=0)$:

$$\mathbf{x}_i(t=0) = \boldsymbol{\mathcal{U}}(lb_{\mathbf{x}}, ub_{\mathbf{x}}) \qquad 6.21$$

    (2) Initialize $\mathbf{x}_{i_p}(t=0)$:

$$\mathbf{x}_{i_p}(t=0) = \mathbf{x}_i(t=0), i = 1 \dots M \qquad 6.22$$

  - Optimization Process:
    (1) Optimize $\mathbf{x}_i(t+1)$ to generate $\hat{\mathbf{x}}_i(t+1)$:

$$\hat{\mathbf{x}}_i(t+1) = \begin{cases} \mathbf{x}_i(t) + r \times w_2 \times (\mathbf{y}_j(t+1) - \mathbf{x}_i(t)) & , \quad r > w_1 \\ \boldsymbol{\mathcal{U}}(lb_{\mathbf{x}}, ub_{\mathbf{x}}) & , \quad \text{o.w} \end{cases} \qquad 6.23$$

    (2) Dealing with outliers C:

$$\mathbf{x}_{i,n}^{\text{fixed}}(t+1) = \begin{cases} \mathbf{x}_{i,n}(t) & , \quad \mathbf{x}_{i,n}(t+1) < lb_{\mathbf{x}} \text{ or } \mathbf{x}_{i,n}(t+1) > ub_{\mathbf{x}} \\ \mathbf{x}_{i,n}(t+1) & , \quad \text{o.w} \end{cases} \qquad 6.24$$

    (3) Select $\mathbf{x}_i(t+1)$ from $\hat{\mathbf{x}}_i(t+1)$:

$$\mathbf{x}_i(t+1) = \hat{\mathbf{x}}_i(t+1) \qquad 6.25$$

(4) Optimize $\mathbf{x}_{i_p}(t)$ to generate $\mathbf{x}_{i_p}(t+1)$:

$$\mathbf{x}_{i_p}(t+1) = \textbf{Min}(\{\mathbf{x}_{i_p}(t), \mathbf{x}_i(t+1)\}) \qquad 6.26$$

Algorithm 11 Re-framed Crow Search Algorithm

```
1:  $t \leftarrow 0$
2:  $Init_{\mathbf{x}}(t=0), M$ as Eq.6.21                          ▷ Initialization Process
3:  $f$                                                          ▷ Evaluation
4:  $Init_{\Delta}$: $\mathbf{x}_{i_p}(t=0)$ as Eq.6.22
5:  while termination criteria are not met do
6:      $Opt_{\mathbf{x}}, C \rightarrow \hat{\mathbf{x}}_i(t+1)$ as Eq.6.23, $C$ as Eq.6.24
7:      $f(\hat{\mathbf{x}}_i(t+1))$                             ▷ Evaluation
8:      $S \rightarrow \mathbf{x}_i(t+1)$ according to Eq.6.25       ▷ Selection
9:      $t \leftarrow t+1$
10:     $Opt_{\Delta}$: $\mathbf{x}_{i_p}(t+1)$ as Eq.6.26
11: end while
```

B.4 Re-framed Moth-flame Optimization Algorithm (Re-framed MFO)

- **Objectives**:

Objective Problem	
$f(\mathbf{x}_i)$	fitness of $\mathbf{x}_i$.
n	the dimension of the search space.
$[lb_{\mathbf{x}}, ub_{\mathbf{x}}]$	the interval of objective variable $\mathbf{x}$, in our cases, it is defined in the IOHprofiler, $[lb_{\mathbf{x}}, ub_{\mathbf{x}}] = [-5, +5]$.
Objective Solution	
$\mathbf{x}_i$	it can be imagined as one individual in Swarm-Intelligence Algorithms, $\mathbf{x}_i \in R^n$.

- **Influencing factors**:

Dynamic influencing factors	
$\mathbf{y}_i$	the best flame position for one individual $\mathbf{y}_i \in R^n$.
z_{1_i}	a weight value.
z_2	a kind of threshold.
Static influencing factors	
T	maximum iteration, the budget in our cases, in our case, it is defined in IOHprofiler.
M	population size, $M = 30$.
w	the shape of spiral, $w = 1$, $w \in (0, +\infty)$.

- **Components**:
  - Initialization Process:
    (1) Initialize $\mathbf{x}_i(t = 0)$:

$$\mathbf{x}_i(t = 0) = \boldsymbol{\mathcal{U}}(lb_{\mathbf{x}}, ub_{\mathbf{x}}) \qquad 6.27$$

    (2) Initialize $\mathbf{x}_s(t = 1)$:

$$\langle \mathbf{x}_i(t) \rangle = \textbf{Sort}(\{\mathbf{x}_i(t)\}), i = 1 \dots M \qquad 6.28$$

    (3) Initialize a weighting value $z_{1_i}(t)$:

$$z_{1_i}(t) = r \times (-2 - \frac{t}{T}) + 1 \qquad 6.29$$

    (4) Initialize the threshold $z_2(t)$:

$$z_2(t) = \textbf{Round}(M - t \times \frac{M-1}{T}) \qquad 6.30$$

  - Optimization Process:

(1) Optimize $\mathbf{x}_i(t)$ to generate $\hat{\mathbf{x}}_i(t+1)$:

$$\hat{\mathbf{x}}_i(t+1) = \begin{cases} (\mathbf{x}_{s_i}(t) - \mathbf{x}_i(t)) \times e^{w \times z_{1_i}(t)} \times \cos(2\pi \times z_{1_i}(t)) + \mathbf{x}_{s_i}(t), & i \leq z_2(t) \\ (\mathbf{x}_{s_{z_2(t)}}(t) - \mathbf{x}_i(t)) \times e^{w \times z_{1_i}(t)} \times \cos(2\pi \times z_{1_i}(t)) + \mathbf{x}_{s_{z_2(t)}}(t), & \text{o.w.} \end{cases} \tag{6.31}$$

(2) Dealing with outliers C:

$$\mathbf{x}_{i,n}^{\text{fixed}}(t+1) = \begin{cases} ub_x & , \mathbf{x}_{i,n}(t+1) > ub_{\mathbf{x}} \\ \mathbf{x}_{i,n}(t+1) & , \text{o.w} \\ lb_x & , \mathbf{x}_{i,n}(t+1) < lb_{\mathbf{x}} \end{cases} \tag{6.32}$$

(3) Select $\mathbf{x}_i(t+1)$ from $\hat{\mathbf{x}}_i(t+1)$:

$$\mathbf{x}_i(t+1) = \hat{\mathbf{x}}_i(t+1) \tag{6.33}$$

(4) Optimize $\mathbf{x}_s(t)$ to generate $\mathbf{x}_s(t+1)$:

$$\langle \mathbf{x}_i(t+1) \rangle = \textbf{Sort}(\{\mathbf{x}_i(t)\} \cup \{\mathbf{x}_i(t+1)\}), i = 1 \dots M \tag{6.34}$$

(5) Optimize $z_{1_i}(t)$ to generate $z_{1_i}(t+1)$:

$$z_{1_i}(t) = r \times (-2 - \frac{t}{T}) + 1 \tag{6.35}$$

(6) Optimize $z_2(t)$ to generate $z_2(t+1)$:

$$z_2(t) = \textbf{Round}(M - t \times \frac{M-1}{T}) \tag{6.36}$$

Algorithm 12 Re-framed moth-flame Optimization Algorithm

```
1: $t \leftarrow 0$
2: $Init_{\mathbf{x}}(t=0), M$ as Eq.6.27                          ▷ Initialization Process
3: $f$                                                             ▷ Evaluation
4: $Init_{\Delta}$: $\mathbf{x}_s(t=0)$ as Eq.6.28
5: $Init_{\Delta}$: $z_{1_i}(t=0)$ as Eq.6.29
6: $Init_{\Delta}$: $z_2(t=0)$ according to Eq.6.30
7: while termination criteria are not met do
8:     $Opt_{\mathbf{x}}, C \rightarrow \hat{\mathbf{x}}_i(y+1)$ as Eq.6.31, $C$ as Eq.6.32   ▷ Optimization Process
9:     $f(\hat{\mathbf{x}}_i(t+1))$                                 ▷ Evaluation
10:    $S \rightarrow \mathbf{x}_i(t+1))$ as Eq.6.33                 ▷ Select
11:    $t \leftarrow t+1$
12:    $Init_{\Delta} \rightarrow \mathbf{x}_s(t+1)$ as Eq.6.34
13:    $Init_{\Delta} \rightarrow z_{1_i}(t+1)$ as Eq.6.35
14:    $Init_{\Delta} \rightarrow z_2(t+1)$ as Eq.6.36
15: end while
```

B.5 Re-framed Monarch Butterfly Algorithm (Re-framed MBO)

- **Objectives**:

Objective Problem	
$f(\mathbf{x}_i)$	fitness of $\mathbf{x}_i$.
n	the dimension of the search space.
$[lb_{\mathbf{x}}, ub_{\mathbf{x}}]$	the interval of objective variable $\mathbf{x}$, in our cases, it is defined in the IOHprofiler, $[lb_{\mathbf{x}}, ub_{\mathbf{x}}] = [-5, +5]$.
Objective Solution	
$\mathbf{x}_i$	it can be imagined as one individual in Swarm-Intelligence Algorithms, $\mathbf{x}_i \in R^n$.

- **Influencing factors**:

Dynamic influencing factors	
$\mathbf{x}_g$	the best position that the whole population has found so far.
z	weighting value of individuals.
Static influencing factors	
T	maximum iteration, the budget in our cases, in our case, it is defined in IOHprofiler.
M	population size, $M = 50$.
w_1	the ratio of stronger population to weaker population, specifically named 'partition', $w_1 = 5/12$.
w_2	the migrating rate, specifically named 'period', $w_2 = 1.2$.
w_3	the adjusting rate, specifically named 'BAR', the adjusting rate, $w_3 = 5/12$.
w_4	the maximum one step size, $w_4 = 1$.
w_5	the number of elitists, $w_5 = 2$, $w_5 \in [0, M]$.

- **Components**:
  - Initialization Process:
    (1) Initialize $\mathbf{x}_i(t=0)$:
    $$\mathbf{x}_i(t=0) = \boldsymbol{\mathcal{U}}(lb_{\mathbf{x}}, ub_{\mathbf{x}}) \qquad 6.37$$
    (2) Initialize $\mathbf{x}_g(t=0)$:
    $$\mathbf{x}_g(t=0) = \textbf{Min}(\{\mathbf{x}_i(t)\}), i = 1 \dots M \qquad 6.38$$
    (3) Initialize $z(t=0)$:
    $$z(t=0) = \frac{w_4}{t^2} \qquad 6.39$$
  - Optimization Process:

(1) Optimize $\hat{\mathbf{x}}_i(t+1)$:

$$\begin{aligned}
&\langle \mathbf{x}_i(t)\rangle = \textbf{Sort}(\{\mathbf{x}_i(t)\}), i = 1 \dots M \\
&{}^{strong}\hat{\mathbf{x}}_i(t+1) = \begin{cases} \mathbf{x}_{j,n}(t), j \in [1, M^{'}] & , \quad r \times w_2 \leqslant w_1 \\ \mathbf{x}_{j,n}(t), j \in (M^{'}, M] & , \quad \text{o.w.} \end{cases} \\
&{}^{weak}\hat{\mathbf{x}}_i(t+1) = \begin{cases} \mathbf{x}_{g,n}(t), r \geqslant w_1 \\ \begin{cases} \mathbf{x}_{j,n}(t) + z(t) \times \left(\textbf{Lévy}_{i,n} - 0.5\right), j \in (M^{'}, M], r > w_3 \\ \mathbf{x}_{j,n}(t), j \in (M^{'}, M], \text{o.w.} \end{cases} & ,\text{o.w.} \end{cases} \\
&\{\hat{\mathbf{x}}_i(t+1)\} = \{{}^{strong}\hat{\mathbf{x}}_i(t+1)\} \cup \{{}^{weak}\hat{\mathbf{x}}_i(t+1)\}
\end{aligned} \quad 6.40$$

where $\mathbf{x}_{j,n}(t) \in \langle \mathbf{x}_i(t)\rangle$, $M^{'} = \lceil w1 \times M \rceil$, $\textbf{Lévy}_{i,n} = \textbf{Lévy}(d, n, T)$ with $d \sim Exp(2 \times T)$.

(2) Dealing with outliers C:

$$\mathbf{x}_{i,n}^{\text{fixed}}(t+1) = \begin{cases} ub_x & , \quad \mathbf{x}_{i,n}(t+1) > ub_{\mathbf{x}} \\ \mathbf{x}_{i,n}(t+1) & , \quad \text{o.w} \\ lb_x & , \quad \mathbf{x}_{i,n}(t+1) < lb_{\mathbf{x}} \end{cases} \quad 6.41$$

(3) Select $\mathbf{x}_i(t+1)$ from $\hat{\mathbf{x}}_i(t+1)$:

$$\begin{aligned}
&\langle \hat{\mathbf{x}}_i(t+1)\rangle = \textbf{Sort}(\{\hat{\mathbf{x}}_i(t+1)\}), i = 1 \dots M \\
&\mathbf{x}_i(t+1) \in \{\langle \hat{\mathbf{x}}_i(t+1)\rangle, i = 1 \dots M - w_5\} \cup \{\langle \mathbf{x}_i(t)\rangle, i = 1 \dots w_5\}
\end{aligned} \quad 6.42$$

(4) Optimize $\mathbf{x}_g(t)$ to generate $\mathbf{x}_g(t+1)$:

$$\mathbf{x}_g(t+1) = \textbf{Min}(\mathbf{x}_g(t) \cup \{\mathbf{x}_i(t+1)\}), i = 1 \dots M \quad 6.43$$

(5) Optimize $z(t)$ to generate $z(t+1)$:

$$z(t+1) = \frac{w_4}{(t+1)^2} \quad 6.44$$

Algorithm 13 Re-framed Monarch Butterfly Optimization Algorithm

```
 1: $t \leftarrow 0$
 2: $Init_{\mathbf{x}}(t=0), M$ as Eq.6.37                                  ▷ Initialization Process
 3: $f$                                                                    ▷ Evaluation
 4: $Init_{\Delta}$: $\mathbf{x}_g(t=0)$ as Eq.6.38
 5: $Init_{\Delta}$: $z(t=0)$ as Eq.6.39
 6: while termination criteria are not met do
 7:     $Opt_{\mathbf{x}}, C \rightarrow \hat{\mathbf{x}}_i(t+1)$ as Eq.6.40, $C$ as Eq.6.41    ▷ Optimization Process
 8:     $f(\hat{\mathbf{x}}_i(t+1))$                                        ▷ Evaluation
 9:     $t \leftarrow t+1$
10:     $S \rightarrow \mathbf{x}_i(t+1)$ as Eq.6.42                        ▷ Selection
11:     $Opt_{\Delta} \rightarrow \mathbf{x}_g(t+1)$ as Eq.6.43
12:     $Opt_{\Delta} \rightarrow z(t+1)$ as Eq.6.44
13: end while
```

B.6 Re-framed Butterfly Optimization Algorithm (Re-framed BOA)

- **Objectives**:

Objective Problem	
$f(\mathbf{x}_i)$	fitness of $\mathbf{x}_i$, $(f : R^n \rightarrow R)$.
n	the dimensionality of the search space.
$[lb_{\mathbf{x}}, ub_{\mathbf{x}}]$	the interval of objective variable $\mathbf{x}$, in our case, it is defined in IOHprofiler, $[lb_{\mathbf{x}}, ub_{\mathbf{x}}] = [-5, +5]$.
Objective Solution	
$\mathbf{x}_i$	it can be imagined as one individual in Swarm-Intelligence Algorithms, $\mathbf{x}_i \in R^n$.

- **Influencing factors**:

Dynamic influencing factors	
$\mathbf{x}_g$	the best position that the whole population has found so far.

z	sensory modality.
Static influencing factors	
T	maximum iteration, the budget in our cases, in our case, it is defined in IOHprofiler.
M	population size, $M = 5$.
z^0	the initial value of sensory modality z, $z^0 = 0.01$, $z^0 \in [0,1]$.
w_1	power exponent, $w_1 = 0.1$, $w_1 \in [0,1]$.
w_2	switch probability, $w_2 = 0.8$.

- **Components**:
  - Initialization Process:

    (1) Initialize $\mathbf{x}_i(t=0)$:

$$\mathbf{x}_i(t=0) = \boldsymbol{\mathcal{U}}(lb_{\mathbf{x}}, ub_{\mathbf{x}}) \tag{6.45}$$

    (2) Initialize $\mathbf{x}_g(t=0)$:

$$\mathbf{x}_g(t=0) = \textbf{Min}(\{\mathbf{x}_i(t)\}), i = 1 \dots M \tag{6.46}$$

    (3) Initialize $z(t=0)$:

$$z(t=0) = z^0 \tag{6.47}$$

  - Optimization Process:

    (1) Optimize $\mathbf{x}(t)$ to generate $\hat{\mathbf{x}}_i(t+1)$:

$$\hat{\mathbf{x}}_i(t+1) = \begin{cases} \mathbf{x}_i(t) + (r^2 \times \mathbf{x}_g(t) - \mathbf{x}_i(t)) \times z_1(t) \times f(\mathbf{x}_i(t))^{w_1} & , \quad r > w_2 \\ \mathbf{x}_i(t) + (r^2 \times \mathbf{x}_j(t) - \mathbf{x}_k(t)) \times z_1(t) \times f(\mathbf{x}_i(t))^{w_1} & , \quad \text{o.w} \end{cases} \tag{6.48}$$

    where $\mathbf{x}_j$ and $\mathbf{x}_k$ are any two neighbors around $\mathbf{x}_i$.

(2) Dealing with outliers C:

$$\mathbf{x}_{i,n}^{\text{fixed}}(t+1) = \begin{cases} ub_x & , \quad \mathbf{x}_{i,n}(t+1) > ub_{\mathbf{x}} \\ \mathbf{x}_{i,n}(t+1) & , \quad \text{o.w} \\ lb_x & , \quad \mathbf{x}_{i,n}(t+1) < lb_{\mathbf{x}} \end{cases} \tag{6.49}$$

(3) Select $\mathbf{x}_i(t+1)$ from $\hat{\mathbf{x}}_i(t+1)$:

$$\mathbf{x}_i(t+1) = \begin{cases} \hat{\mathbf{x}}_i(t+1) & , \quad f(\hat{\mathbf{x}}_i(t+1)) < f(\mathbf{x}_i(t)) \\ \mathbf{x}_i(t) & , \quad \text{o.w} \end{cases} \tag{6.50}$$

(4) Optimize $\mathbf{x}_g(t)$ to generate $\mathbf{x}_g(t+1)$:

$$\mathbf{x}_g(t+1) = \textbf{Min}(\mathbf{x}_g(t) \cup \{\mathbf{x}_i(t+1)\}), i = 1 \dots M \tag{6.51}$$

(5) Optimize $z(t)$ to generate $z(t+1)$:

$$z(t+1) = z(t) + \frac{0.025}{z(t) \times T} \tag{6.52}$$

Algorithm 14 Re-framed Butterfly Optimization Algorithm

```
 1: $t \leftarrow 0$
 2: $Init_{\mathbf{x}}(t=0), M$ as Eq.6.45                      ▷ Initialization Process
 3: $f$                                                        ▷ Evaluation
 4: $Init_{\Delta}$: $\mathbf{x}_g(t=0)$ as Eq.6.46
 5: $Init_{\Delta}$: $z(t=0)$ as Eq.6.47
 6: while termination criteria are not met do
 7:     $Opt_{\mathbf{x}}, C \rightarrow \hat{\mathbf{x}}_i(t+1)$ as Eq.6.48, $C$ as Eq.6.49    ▷ Optimization Process
 8:     $f(\hat{\mathbf{x}}_i(t+1))$                            ▷ Evaluation
 9:     $S \rightarrow \mathbf{x}_i(t+1)$ as Eq.6.50             ▷ Selection
10:     $t \leftarrow t+1$
11:     $Opt_{\Delta} \rightarrow \mathbf{x}_g(t+1)$ as Eq.6.51
12:     $Opt_{\Delta} \rightarrow z(t+1)$ as Eq.6.52
13: end while
```

B.7 Re-framed Particle Swarm Optimization (Re-framed PSO)

- **Objectives**:

Objective Problem	
$f(\mathbf{x}_i)$	fitness of $\mathbf{x}_i$.
n	the dimension of the search space.
$[lb_{\mathbf{x}}, ub_{\mathbf{x}}]$	the interval of objective variable $\mathbf{x}$, in our cases, it is defined in the IOHprofiler, $[lb_{\mathbf{x}}, ub_{\mathbf{x}}] = [-5, +5]$.
Objective Solution	
$\mathbf{x}_i$	it can be imagined as one individual in Swarm-Intelligence Algorithms, $\mathbf{x}_i \in R^n$.

- **Influencing factors**:

Dynamic influencing factors	
$\mathbf{x}_{ip}$	the best position that this individual $\mathbf{x}_i$ has found so far.
$\mathbf{x}_g$	the best position that the whole population has found so far.
$\mathbf{y}_i$	the velocity for one individual $\mathbf{y}_i \in R^n$.
Static influencing factors	
T	maximum iteration, the budget in our cases, in our case, it is defined in IOHprofiler.
M	population size, $M = 25$.
$[lb_{\mathbf{y}}, ub_{\mathbf{y}}]$	the interval of assisting variable $\mathbf{y}$, $[lb_{\mathbf{y}}, ub_{\mathbf{y}}] = [0, 0]$.
w_1	scaling strength, here it is specifically called inertia weight, $w_1 = 0.73$, $w_1 \in (0, +\infty)$.
w_2	here it is specifically called personal coefficient, $w_2 = 1.49$, $w_2 \in [0, +\infty)$.
w_3	here it is specifically called global coefficient, $w_3 = 1.49$, $w_3 \in [0, +\infty)$.

- **Components**:
  - Initialization Process:

    (1) Initialize $\mathbf{x}_i(t=0)$:

    $$\mathbf{x}_i(t=0) = \boldsymbol{\mathcal{U}}(lb_{\mathbf{x}}, ub_{\mathbf{x}}) \tag{6.53}$$

    (2) Initialize $\mathbf{y}_i(t=0)$:

    $$\mathbf{y}_i(t=0) = \boldsymbol{\mathcal{U}}(lb_{\mathbf{y}}, ub_{\mathbf{y}}) \tag{6.54}$$

    (3) Initialize $\mathbf{x}_{i_p}(t=0)$:

    $$\mathbf{x}_{i_p}(t=0) = \mathbf{x}_i(t) \tag{6.55}$$

    (4) Initialize $\mathbf{x}_g(t=0)$:

    $$\mathbf{x}_g(t=0) = \textbf{Min}(\{\mathbf{x}_i(t)\}), i = 1 \dots M \tag{6.56}$$

  - Optimization Process:

    (1) Optimize $\mathbf{y}_i(t)$ to generate $\mathbf{y}_i(t+1)$:

    $$\begin{aligned} \mathbf{y}_i(t+1) &= w_1 \times \mathbf{y}_i(t) + \mathcal{U}(0, w_2) \times (\mathbf{x}_{i_p}(t) - \mathbf{x}_i(t)) \\ &\quad + \mathcal{U}(0, w_3) \times (\mathbf{x}_g(t) - \mathbf{x}_i(t)) \end{aligned} \tag{6.57}$$

    (2) Optimize $\mathbf{x}_i(t)$ to generate $\hat{\mathbf{x}}_i(t+1)$:

    $$\hat{\mathbf{x}}_i(t+1) = \mathbf{x}_i(t) + \mathbf{y}_i(t+1) \tag{6.58}$$

    (3) Dealing with outliers C:

    $$\mathbf{x}_{i,n}^{\text{fixed}}(t+1) = \begin{cases} ub_x & , \quad \mathbf{x}_{i,n}(t+1) > ub_{\mathbf{x}} \\ \mathbf{x}_{i,n}(t+1) & , \quad \text{o.w} \\ lb_x & , \quad \mathbf{x}_{i,n}(t+1) < lb_{\mathbf{x}} \end{cases} \tag{6.59}$$

(4) Select $\mathbf{x}_i(t+1)$ from $\hat{\mathbf{x}}_i(t+1)$:

$$\mathbf{x}_i(t+1) = \hat{\mathbf{x}}_i(t+1) \tag{6.60}$$

(5) Optimize $\mathbf{x}_{i_p}(t)$ to generate $\mathbf{x}_{i_p}(t+1)$:

$$\mathbf{x}_{i_p}(t+1) = \textbf{Min}(\{\mathbf{x}_{i_p}(t), \mathbf{x}_i(t+1)\}) \tag{6.61}$$

(6) Optimize $\mathbf{x}_g(t)$ to generate $\mathbf{x}_g(t+1)$:

$$\mathbf{x}_g(t+1) = \textbf{Min}(\mathbf{x}_g(t) \cup \{\mathbf{x}_i(t+1)\}), i = 1 \dots M \tag{6.62}$$

Algorithm 15 Re-framed Particle Swarm Optimization

```
1: $t \leftarrow 0$
2: $Init_{\mathbf{x}}(t=0), M$ as Eq.6.53                          ▷ Initialization Process
3: $f$                                                           ▷ Evaluation
4: $Init_{\Delta}$: $\mathbf{y}_i(t=0)$ as Eq.6.54
5: $Init_{\Delta}$: $\mathbf{x}_{i_p}(t=0)$ as Eq.6.55
6: $Init_{\Delta}$: $\mathbf{x}_g(t=0)$ as Eq.6.56
7: while termination criteria are not met do
8:     $Opt_{\Delta} \rightarrow \mathbf{y}_i(t+1)$ as Eq.6.57                ▷ Optimization Process
9:     $Opt_{\mathbf{x}}, C \rightarrow \hat{\mathbf{x}}_i(t+1)$ as Eq.6.58, $C$ as Eq.6.59
10:    $f(\hat{\mathbf{x}}_i(t+1))$                                    ▷ Evaluation
11:    $S \rightarrow \mathbf{x}_i(t+1))$ as Eq.6.60                        ▷ Select
12:    $t \leftarrow t+1$
13:    $Opt_{\Delta} \rightarrow \mathbf{x}_{i_p}(t+1)$ as Eq.6.61
14:    $Opt_{\Delta} \rightarrow \mathbf{x}_g(t+1)$ as Eq.6.62
15: end while
```

C Whether synchronous or asynchronous evaluation will impact the algorithm performance?

D Whether synchronous or asynchronous $\mathbf{x}_g$ will impact the algorithm performance?