Chapter 2

General Dictionary For General Terminologies

2.1 Motivation

This chapter discusses the commonality among seven selected nature-inspired algorithms on the level of their components. Specifically, in these algorithms, various terminologies of components with their meanings are categorized into general terminologies with general definitions. **We call the catalog of general terminologies with general definitions the General Dictionary.**

Firstly, we want to introduce several common symbols in nature-inspired algorithms. In the nature-inspired algorithm researching field [1], the given objective optimization problem is formulated as a math formula $f(\vec{x})$ in which each possible solution is represented as a vector $\vec{x}$. Moreover, the number of independent elements in $\vec{x}$ is usually defined as the dimension of search space for the objective optimization problem. When seeking the optimal solution to the optimization problem, algorithms will first initialize an arbitrary set of $\vec{x}$ and then optimize this set by using iterative strategies. Same as in most studies, we also denote $\vec{x}$ as $\mathbf{x}$ and call it the objective solution. Inspired by this representation, we define that any term $\mathbf{a}$ in the bold style denotes the item that is also a vector with the same dimension as the objective solution $\mathbf{x}$. Furthermore, we want to clarify that if the bold style item $\mathbf{a}$ has a subscript i, it means this item has a one-to-one correspondence

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

with each individual $\mathbf{x}_i$. However, one item without this subscript will apply to the whole population, whether this item is in bold style or not in bold style.

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

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

2.2 Specific Swarm-based Algorithms

2.2.1 Bat-inspired Algorithm (BA)

Description When simulating the echolocation behavior of micro-bats, for simplicity, the Bat-inspired algorithm only considered the velocity $\mathbf{v}_i$ of each bat $\mathbf{x}_i$, the echo frequency $freq$ emitted by each bat $\mathbf{x}_i$, the rate r of echo pulse, and the loudness A of echo [33]. The algorithm described here follows the published implementation code [2], and explanations with math formulas are from their published paper [33].

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

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

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

After (3) evaluating fitness of each $\mathbf{x}_i$, it (4) goes to find the best individual $\mathbf{x}_*$ that is the best $\mathbf{x}_i$ the whole population has found so far. Next is (5) iteratively optimizing process in which the maximum number of iterations is $t\_max = 1000$. Under the iterative process, it firstly (6) updates A as Eq.2.3 in which $A(t = 0) = 1$ is the initial value of loundness,

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

and $\alpha = 0.97$ is a decreasing hyper-parameter. Next, (7) r is updated as Eq.2.4 in which $r_0 = 1$ is the initial value of pulse rate and $\gamma = 0.1$ is a decreasing hyper-parameter. In this algorithm, t is the iteration counter.

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

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

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

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

If (9) $rand < r(t+1)$, $\mathbf{x}_i$ will be further updated as Eq.2.6 in which $\epsilon = 0.1$.

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

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

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

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

Moreover, we want to clarify that in their implemented code, it is unreasonable to compare the previous best bat with a current unaccepted bat when updating the current best bat. The definition of the current best bat is finding the best bat in the current population. The unaccepted bat is not in the current population anymore. Therefore, it is more

reasonable to calculate the current best bat by comparing the previous best bat with the current bat who is at this same position in the current population.

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

2.2.2 Grasshopper Optimization Algorithm (GOA)

Description The Grasshopper Optimization Algorithm was proposed in 2017 [26]. They designed this algorithm by studying grasshoppers' main swarming behaviors in larval and adulthood phases, such as slow movement and small steps in the larval phase, abrupt movement and long-range steps in the adulthood phase, and food source seeking in both phases. Their final model described here is from their published code [3], and explanations with math formulas are from their published paper [26].

This model starts at (1) initializing grasshoppers $\mathbf{x}_i$ with size $N = 100$ in the d dimension environment. The initialization method in their published code can be simply re-framed as Eq.2.7 in which $up/down$ is the upper/down boundary of $\mathbf{x}_i$ [4].

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

Next is (2) calculating the fitness of each individual $\mathbf{x}_i$ and (3) finding the best individual $\mathbf{T}$ that the whole population has found so far. The followings are iterative optimizing process in which the stop condition is current number of iteration l is smaller than the maximum number of iterations $max\_iteration = 100$ [5]. Under the iterative process, (4) the hyper-parameter c is firstly updated as Eq.2.8 in which $cMin = 0.00004/cMax = 1$ is the lower/upper boundary of c.

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

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

[4] 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 6, we only consider the first case.

[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.

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

$$\widetilde{d} = 2 + d \bmod 2 \tag{2.9}$$

Next is (6) updating $\mathbf{x}_i$ as Eq.2.10 that is a simplified version of their published original formula. Because we only consider one case that elements in $\mathbf{x}_i$ share the same boundary, we replace ub_d/lb_d by ub/lb. We also replace x_i^d by $\mathbf{x}_i$ and $\widehat{T_d}$ by $\mathbf{T}$ in the view of vector [7].

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

The method of updating Eq.2.10 has a function named S in which $\widetilde{d}_{i,j}$ is the input. The S function is formulated as Eq.2.11 in which $f = 0.5$ and $l = 1.5$ are two hyper-parameters [8].

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

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

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

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

2.2.3 Crow Search Algorithm (CSA)

Description The crow search algorithm was designed [2] in 2016. The core metaphor is crows' behaviour of hiding excess food and retrieving this stored food when needed. The author believes that these behaviours, including stealing others' food by tailing after them and fighting with theft by moving to another place instead of their real hiding place, are similar to an optimization process [2]. Their final algorithm model described here is

[6] It refers to their faster version code.

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

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

from their published implementation code [9], and explanations with math formulas are from their published paper [2].

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

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

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

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

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

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

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

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

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

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

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

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

2.2.4 Moth-flame Optimization Algorithm (MFO)

Description In 2015, the Moth-flame optimization algorithm simulated the process that moths will fly to the center of artificial light (that is also seen as flames) in the spiral path [22]. The core characteristic in MFO is that the author defined the positions of flames as the optimization orientation of moths, and each moth has its own dynamically changing flame. The final algorithm model described here is from their published implementation code [10], and explanations with math formulas are from their published paper [22].

This algorithm starts from (1) $N = 30$ initial moths population $Moth\_pos$ in a dim dimension environment. The initialization method in their published code can be formulated as Eq.2.16 in which $\mathbf{m}_i$ is each moth and ub/lb is the upper/lower boundary of elements in each moth [11].

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

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

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

After (3) replacing outliers in $\mathbf{x}_i$ by the boundary values, fitness of each moth is calculated. Next is (4) sorting the combination of the current population and the previous population [12] based on the ascending of their fitness, and the first $N = 30$ moths in the combination

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

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

[12] The previous population is $\emptyset$ (there is no previous population), when the current population is the initial population.

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

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

Next, (5) each element j of each moth $\mathbf{m}_i$ will be updated as follows: if $i \leqslant flame\_no$, Eq.2.19; if not, Eq.2.20 in which $\mathbf{m}^{'}_{i,j}$ denotes each newly updated element j of each moth $\mathbf{m}_i$; $b = 1$ is a constant parameter used to define the shape of spiral [22]; $t = (a-1) \times rand + 1$ is a random number related to a.

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

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

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

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

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

2.2.5 Monarch Butterfly Optimization (MBO)

Description The Monarch Butterfly Optimization simulated the migration behavior of monarch butterflies in 2015 [30]. The core metaphor is that monarch butterflies in two different habitats evolve differently. These two habitats are mimicked by dividing the whole population into two subpopulations. The author defined that the whole population

will firstly be sorted based on their fitness before dividing them into two with a ratio [30]. Then, each habitat experiences its particular evolution approach (migration operator or adjusting operator [30]). We describe their final algorithm model implemented in their published code [13], and explanations with math expressions from their published paper [30].

This 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 $\mathbf{x}_i$.

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

After (2) evaluating each $\mathbf{x}_i$, the entire population is sorted from the best to the worst. Then (3) it starts iteratively optimizing each $\mathbf{x}_i$ in which the stop condition is the maximum number of iterations $Maxgen = 50$. Under the iterative process, (4) $Keep = 2$ best $\mathbf{x}_i$ are first kept in $chromKeep$ [15]. Next is (5) dividing the entire $Population$ into two sub-populations with a ratio $partition = \frac{5}{12}$. The two sub-populations ($Population1$, $Population2$) will be updated by two operators ('Migration operator', 'Adjusting operator') respectively. (6) In the $Population1$, if $rand \times period \leqslant partition$ in which $period = 1.2$, $\mathbf{x}_{i,k}^{t+1} = \mathbf{x}_{population1,k}^{t}$; else $\mathbf{x}_{i,k}^{t+1} = \mathbf{x}_{population2,k}^{t}$, in which k is the $k-$th element of $\mathbf{x}_i$, and $population1/population2$ is one arbitrary moth from previous $Population1/Population2$. (7) Next is fixing outliers in this newly generated $Population1$ by replacing them with the boundary value, before evaluating this newly generated $Population1$.

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

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

[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.

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

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

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

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

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

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

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

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

2.2.6 Butterfly Optimization Algorithm (BOA)

Description The Butterfly Optimization Algorithm was modeled in 2018 [3]. Butterflies' behaviors of searching for food and mates are mimicked to solve optimization problems. The algorithm model described here is from their published code [16], and explanations with math formulas are from their published paper [3].

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

In this algorithm, they define a dim dimensional environment in which each individual is $\mathbf{x}_i$ and the fitness of $\mathbf{x}_i$ is $f(\mathbf{x}_i)$. (1) The algorithm first initializes a population $n = 50$ of $\mathbf{x}_i$ by Eq.2.26 in which Lb/Ub is lower/upper bound [17].

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

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

Next steps are the iterative optimization process in which the stop condition is the current iteration t is smaller than the maximum number of iterations N_iter. Under the iterative process, for each $\mathbf{x}_i$, (4) its own fragrance FP_i is first calculated as Eq.2.27 in which $f(\mathbf{x}_i)$ is the fitness value of $\mathbf{x}_i$ [18].

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

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

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

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

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

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

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

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

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

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

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

2.2.7 Particle Swarm Optimization (PSO)

Description Particle Swarm Optimization was firstly introduced in 1995 [17]. The main idea comes from the movements of the animal swarm. By simulating their behaviors that the swarm dynamically moves to a 'roost' [17], the extremely simple algorithm, even with a small swarm size (15 to 30 agents), boasted impressive performance on solving continuous optimization functions. In the very first paper, the position of each agent in its swarm was controlled by two velocities $\mathbf{X}$ and $\mathbf{Y}$. Moreover, each agent can remember its best position and know the global best position in its swarm. Although the very first paper also provided the algorithm model with formulas, we preferred to reference clearer published pseudocode whose algorithm model is as same as the very first paper possible. The final PSO model described in this work is from a tutorial of this method [21], and the hyper-parameter settings are from an application of this method [5].

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

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

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

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

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

After (4) calculating the fitness of each agent $\mathbf{x}_i$, the algorithm obtains (5) the best agent $gbest$ that is the whole swarm has found so far in the initial swarm. The following steps are about (6) iterative optimization process with a stopping condition. Under the iterative process, the published pseudo-code first updates each velocity $\mathbf{v}_i$ by Eq.2.34 in which $w = 0.73$ is an adjusting parameter, $c_1 = 1.49$ and $c_2 = 1.49$ is respectively the cognitive and social coefficient [21].

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

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

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

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

As a side note, the very first published paper did not directly point out the notation for 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 model also needs a concept of dimension. Meanwhile, the stop condition and the method to deal with the outliers also must exist. In this work, the method of fixing outliers is the most common way that replaces outliers by boundary values of $\mathbf{x}_i$.

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

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

2.3 General Dictionary

2.3.1 Catalog of General Terminologies

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

In Table 2.1, ***Information 1 to 5, 13, 15, 17-20*** **are compulsory components**, because they must exit when modeling a nature-inspired algorithm. In other words, these components can theoretically make up a most basic nature-inspired algorithm, if we don't consider whether the algorithm performance is excellent. Meanwhile, we consider the ***Information 18*** **is also a compulsory component**, because firstly, selection process is an important process of natural evolution; secondly, it is reasonable to convert a process that does not use selection to using a special selection process that all individuals are delivered into the next generation round. More details are as follows:

- *Information 1, 3, 4*: the objective function $f()$ must exist to determine where the algorithm will happen. When the $f()$ is determined, the dimension n and the boundary $[lb_{\mathbf{x}}, ub_{\mathbf{x}}]$ are also determined which is because they exist in the $f()$.
- *Information 2, 5, 13, 15*: when searching possible solutions to $f()$, possible solutions $\mathbf{x}_i$ with the number M of $\mathbf{x}_i$ also must exist, because it determines the size of the 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.
- *Information 17, 18, 19, 20*: because these swarm-based optimization algorithms are a kind of iterative optimization heuristic algorithms in which sampling and repetition play an important role in finding optimal solutions [7], the method S related to sampling and the stop condition T must exist. Moreover, these algorithms will solve

the problem in a limited searching space, there must also be a method C to deal with outliers outside the searching space.

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

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

- *Information 11*: Static step-size $\Delta : w, z^0, [lb_{\mathbf{y}}, ub_{\mathbf{y}}]$.
  Most static numerical step-size, such as w, is able to be understood as common hyper-parameters in most algorithms. They are numbers; are set before running algorithms and are unchanged when approaching algorithms. In Figure 2.1, we find there are two groups of static step-size. The first group is w that is commonly seen as hyper-parameters. The second group is the initial value of dynamic step-size (see next item •) in which the dynamic numerical step-size z needs an initial value z^0, the $\mathbf{y}$-relative dynamic vector step size $\mathbf{y}_i$ needs an initial value $[lb_{\mathbf{y}}, ub_{\mathbf{y}}]$.
- *Information 6, 7, 8, 9, 12*: Dynamic step-size $\Delta : z, \mathbf{x}_{ip}, \mathbf{x}_g, \mathbf{x}_s, \mathbf{y}_i$.
  - *Information 12*: Dynamic numerical step-size $\Delta : z$.
    The dynamic numerical step-size z is changing over the iteration t during the optimization process.
  - Dynamic vector step-size $\Delta : \mathbf{x}_{ip}, \mathbf{x}_g, \mathbf{x}_s, \mathbf{y}_i$.
    As its name indicates, this kind of step-size is a vector with the same dimension as $\mathbf{x}$.
    - *Information 6, 7, 8*: $\mathbf{x}$-relative vector step size $\Delta : \mathbf{x}_{ip}, \mathbf{x}_g, \mathbf{x}_s$.
      This kind of vector can be generated from the target $\mathbf{x}$. Specifically, there is a straightforward formula between $\mathbf{x}$ and $\mathbf{x}$-relative vector step-size. For example, personal best position $\mathbf{x}_{ip}$, global best position $\mathbf{x}_g$. The $\mathbf{x}_s$

denotes a special operator related to $\mathbf{x}$, and it is mainly customized. For example, the sorted population in MFO.

* *Information 9*: $\mathbf{y}$-relative vector step size $\Delta : \mathbf{y}_i$.
  This kind of vector is generated without considering the target $\mathbf{x}_i$. It can be imagined as a tool that one algorithm needs it to improve algorithm's performance. The algorithm can never generate this tool by itself, besides getting it from outside of itself. In other words, it is impossible to represent this kind of vector variable by translating other existing variables in a straightforward method. Normally, this kind of vector has same size as the population of individual. For example, velocity $\mathbf{v}_i$.

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

2.3.2 Extension in Other Swarm-based Algorithms.

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

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

In most of swarm-based algorithms, besides the animal $\mathbf{x}$, another noun is also always existing. This noun could be the velocity (in BA, PSO), the memory (in CSA) or the flame (in MFO). We are not allowed to directly assign them the $\mathbf{y}$-relative vector step-size, although it is the most straightforward way. **The reason is that the information**

carried by the noun is the only principle used to assign them different general component name. For example:

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

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

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

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

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

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

2.4 Summary

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

In Subsection 2.3.1, we detail the classification of these 20 general components. Every algorithm must have the target problem f, the search space n and $[lb_{\mathbf{x}}, ub_{\mathbf{x}}]$, the possible target solution $\mathbf{x}_i$, the population size M, the stop condition T, the initialization method $Init_{\mathbf{x}}$ to $\mathbf{x}_i$, the optimization method $Opt_{\mathbf{x}}$ to $\mathbf{x}_i$, the method C to deal with outliers and the method S to select which individuals could be delivered into the next generation. However, selective components are more flexible, and these selective components are all related to determine how far the newly generated individual will move. Therefore, we name selective components step-size Δ. These selective components can be further categorized into static step-size $\Delta : w, z^0, [lb_{\mathbf{y}}, ub_{\mathbf{y}}]$ and dynamic step-size $\Delta : z, \mathbf{x}_{i_p}, \mathbf{x}_g, \mathbf{x}_s, \mathbf{y}_i$. What is interesting is the dynamic step-size also has two categories: dynamic numeric step-size $\Delta : z$ and dynamic vector step-size $\Delta : \mathbf{x}_{i_p}, \mathbf{x}_g, \mathbf{x}_s$ that are $\mathbf{x}$-relative vector step-size and $\Delta : \mathbf{y}_i$ that is $\mathbf{y}$-relative vector step-size.

In Subsection 2.3.2, we discuss how to extend this general dictionary into any other swarm-based algorithms. Furthermore, we resolve a confusion of the difference between $\mathbf{x}$-relative vector step size and $\mathbf{y}$-relative vector step size.

In conclusion, there is a general dictionary in which we can discuss these seven selected algorithms on the level of totally same components.

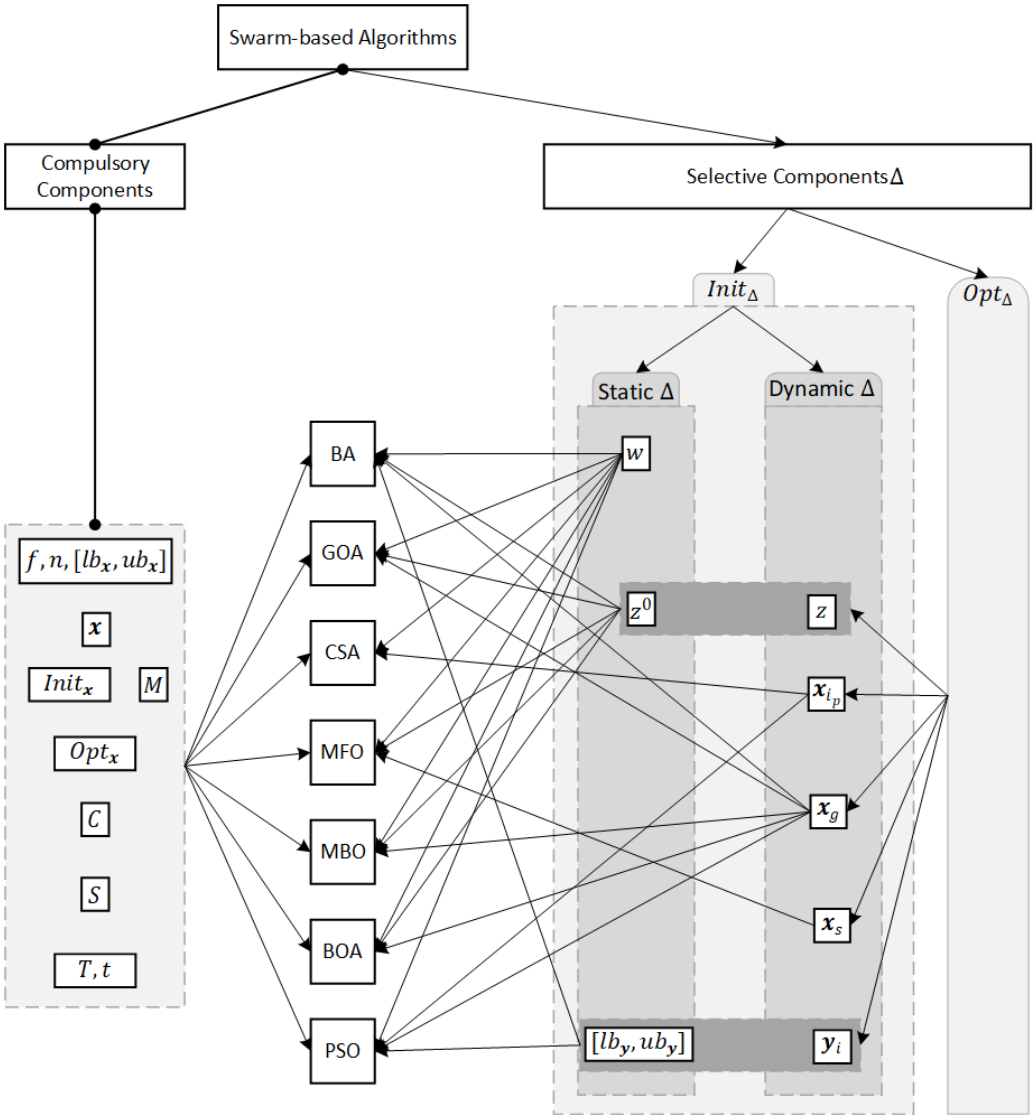

Figure 2.1: These algorithms made of the entire compulsory components and some of selective components.

Table 2.1: General dictionary: all information appearing in these seven algorithms can be categorized into 20 components. The none in gray color means one information doesn't exist in this algorithm.

Information	Different Representation	General 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 $\mathbf{x}_i$[19].	•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 step size Δ.
7.The best $\mathbf{x}_i$ that the entire population has found so far. It will change during the optimization process.	•BA: $\mathbf{x}_*$. •GOA: $\mathbf{T}$. •CSA: none. •MFO: none. •MBO: $\mathbf{x}_{best}$. •BOA: g^*. •PSO: $gbest$.	$\mathbf{x}_g$: $\mathbf{x}$-relative dynamic step size Δ.

[19]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 step sizes 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 step size Δ.
9.Some special vector step sizes 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 step size 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.step sizes 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: step size Δ.
12.step sizes 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: step size Δ.
13.How to initialize $\mathbf{x}_i$ with size M in the n dimension environment.	•BA: Eq.2.1. •GOA: Eq.2.7. •CSA: Eq.2.12. •MFO: Eq.2.16. •MBO: Eq.2.21. •BOA: Eq.2.26. •PSO: Eq.2.31.	$Init_{\mathbf{x}}$: initialization method on $\mathbf{x}$.

14.How to initialize dynamic vector step sizes.	•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 step sizes.	•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 iteration counter.	•BA: t. •GOA: l. •CSA: t. •MFO: t. •MBO: t. •BOA: t. •PSO: current iteration.	t: current iteration.
20.The maximum number of iterations.	•BA: t_max. •GOA: $max\_iteration$. •CSA: $tmax$. •MFO: T. •MBO: $Naxgen$. •BOA: N_iter. •PSO: stop condition.	T: the budget.